



EEZ Studio Reference guide

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

Ver. 0.22.0 – 02/2025
www.envox.eu
github.com/eez-open

Table of Contents

C1. Legal information.....	C.5
C1.1. Definitions.....	C.5
C1.2. Disclaimers.....	C.5
C1.3. Miscellaneous.....	C.6
C1.4. Contact information.....	C.6
C1.5. Revision history.....	C.7
C2. The EEZ Studio overview.....	C.9
C2.1. Introduction.....	C.9
C2.2. Main sections.....	C.9
C2.3. Known issues and issue reporting.....	C.9
C2.4. Donations.....	C.9
C3. Installation.....	C.11
C3.1. System requirements.....	C.11
C3.2. Linux.....	C.11
C3.3. Mac.....	C.11
C3.4. Windows.....	C.11
C3.5. Nix package manager.....	C.12
C3.6. Build and run from source (all operating systems).....	C.12
C3.6.1. Linux only.....	C.12
C3.6.2. Raspbian only.....	C.12
C3.6.3. All platforms.....	C.12
C3.6.4. Raspbian.....	C.12
C3.6.5. Nix.....	C.12
C3.7. USB TMC.....	C.12
C3.7.1. Windows.....	C.13
C3.7.2. Linux.....	C.13
C3.8. FAQ.....	C.13
C4. Key features.....	C.15
C4.1. General.....	C.15
C4.2. EEZ Studio Project.....	C.15
C4.3. EEZ Studio Instrument.....	C.15
C5. Menu options and Settings.....	C.17
C5.1. Home page.....	C.17
C5.2. Menu options.....	C.17
C5.2.1. File.....	C.17
C5.2.2. Edit.....	C.18
C5.2.3. View.....	C.18
C5.2.4. Help.....	C.19

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.

Software development in Envox is very dynamic, which means that differences between versions may include minor or major changes to the GUI. Since the number of images in this document is constantly growing, we are not able to keep them all up-to-date, therefore some of them will not reflect the appearance of the latest version.

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
2023-08-31	0.10.3 (M17)	Initial release
2024-04-08	0.12.0 (M18)	Home page redesign
2024-05-09	0.13.0 (M19)	Support for non-SCPI instruments and devices
2025-05-16	0.13.1	LVGL bux fixes
2024-06-07	0.14.0 (M20)	Hybrid table/tree/grid widget
2024-06-12	0.14.1	LVGL and flow improvements
2024-06-16	0.14.2	"Scrollbar mode" and "Scroll direction" property description for the LVGL widgets
2024-08-04	0.15.0 (M21)	Project scrapbook, Copy/paste between projects
2024-09-02	0.16.0 (M22)	Improvement of session-centric work with instruments and data management
2024-09-30	0.17.0 (M23)	Multimedia support/Support for networking/other Instruments related
2024-10-08	0.18.0	Various fixes and improvements (check release note on GitHub)
2024-11-04	0.19.0	Various fixes and improvements (check release note on GitHub)
2024-11-18	0.20.0	Various fixes and improvements (check release note on GitHub)
2024-12-17	0.21.0	Various fixes and improvements (check release note on GitHub)
2025-02-10	0.22.0	Migration to LVGL v9.2.2, Play Sound and FocusWidget actions

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.

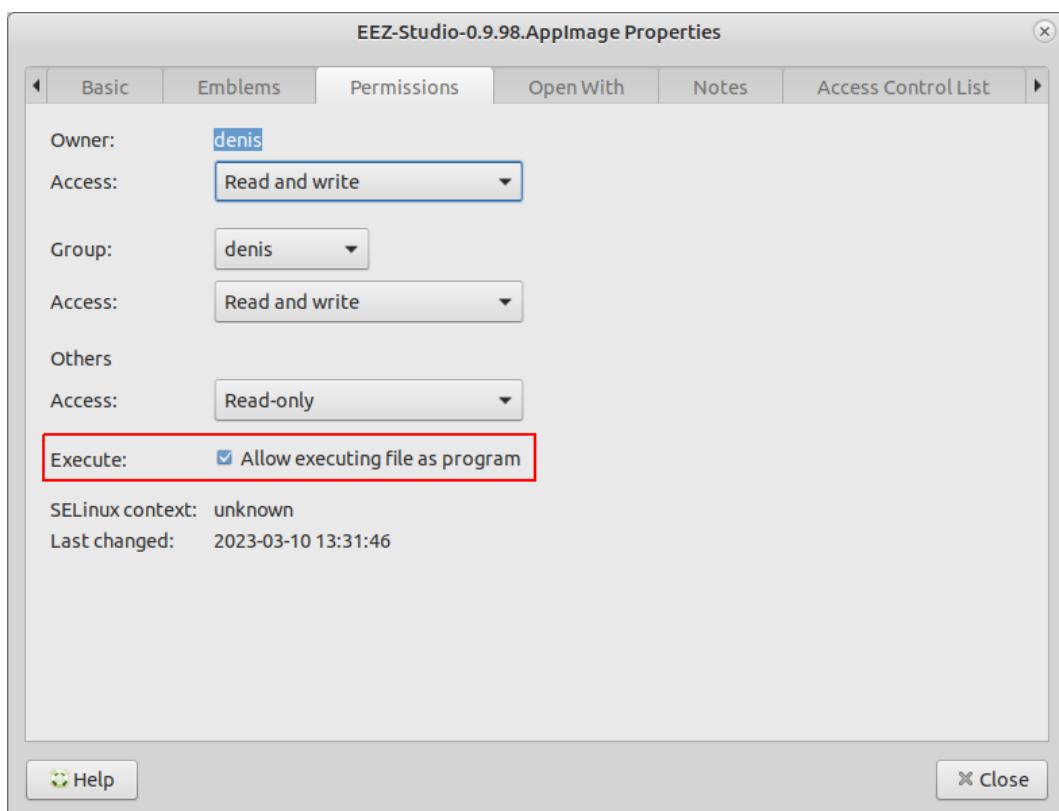


Fig. 1: .AppImage file permission

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://linode.com/install-node-js-and-npm-on-raspberry-pi/>

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

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:

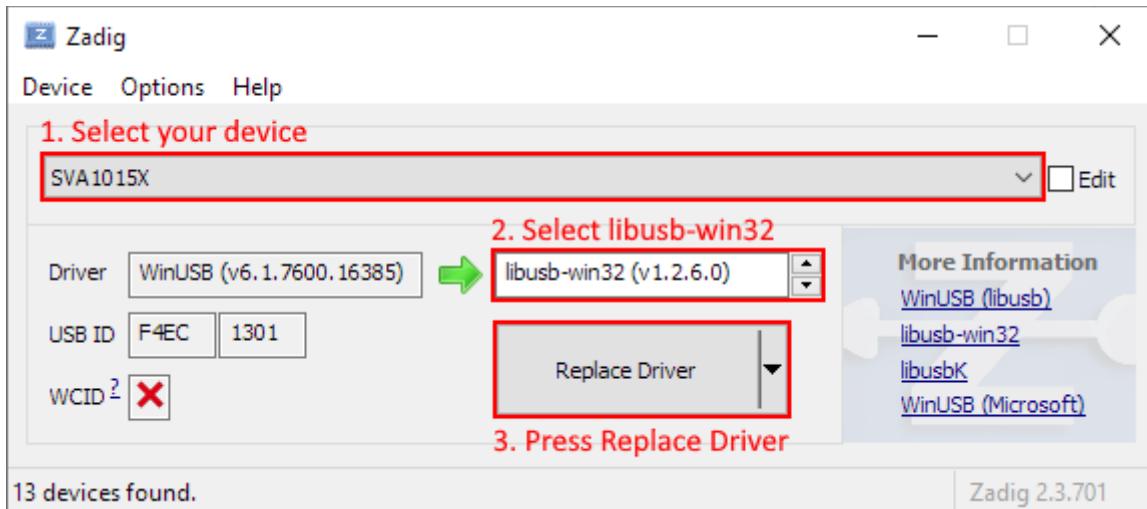


Fig. 2: Zadig driver settings

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](#))
- Free and open source, EEZ Studio license: GPL 3.0; Runtime License (user selectable): MIT, BSD 2.0, Public Domain

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 Graphics Library) support
- Multi-language support
- Support for unlimited number of Color Themes
- Support for unlimited number of Widget styles
- Support for unlimited number of user created Widgets and Actions
- Copy/paste between projects
- Project scrapbook
- Adding new functionality using Project extensions
- 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 gitea.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
- Custom, 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)

EEZ Studio Reference guide

- 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 (Fig. 3). At the top of the page there is a toolbar with options for working with Projects, Instruments, Extensions and EEZ Studio settings.

The options for working with projects (*Open*, *Create* and *Examples*) will be described in detail in the chapters that start with the prefix P. (i.e. P.1, P.2, ...), and the *Instruments* section in chapters started with prefix I.

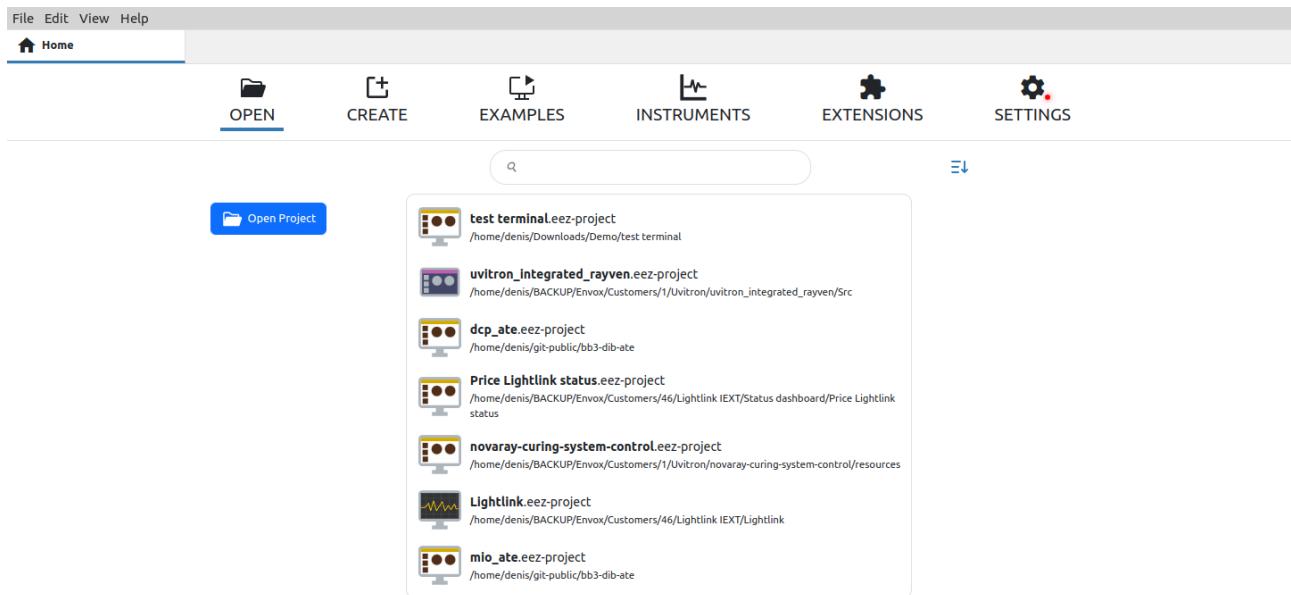


Fig. 3: Home page

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.
Reload (Projects only)	-	Reload currently selected project. If there are unsaved changes, a message will appear asking if you want to save the messages before reloading.
Load Debug Info... (Projects only)	-	Loads the debugger state and switches the project to Debug mode. <i>Note: this is a valid operation only in the project in which the debugger state file was generated.</i>

<i>Save Debug Info... (Projects only)</i>	–	When the project is in <i>Debug</i> mode, use this option to save the debugger state to a file.
<i>Import Instrument Definition...</i>	–	Import IEXT (Instrument EXTension) file.
<i>Save</i>	CTRL + S	Saving project files.
<i>Save as (Projects only)</i>	CTRL + SHIFT + S	Saving the project under a different name.
<i>Check (Projects only)</i>	CTRL + K	Opens the <i>Check</i> panel of the project.
<i>Build (Projects only)</i>	CTRL + B	Starts the build procedure and opens the <i>Build</i> panel of the project.
<i>Build Extensions (Projects only)</i>	–	Build IEXT .zip files only if the project has IEXT (Instrument EXTension) definitions.
<i>Build and Install Extensions (Projects only)</i>	–	The same as the previous option and in addition the IEXTs that have been built are installed immediately.
<i>Exit</i>	–	EEZ Studio shutdown.

C5.2.2. Edit

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

C5.2.3. View

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

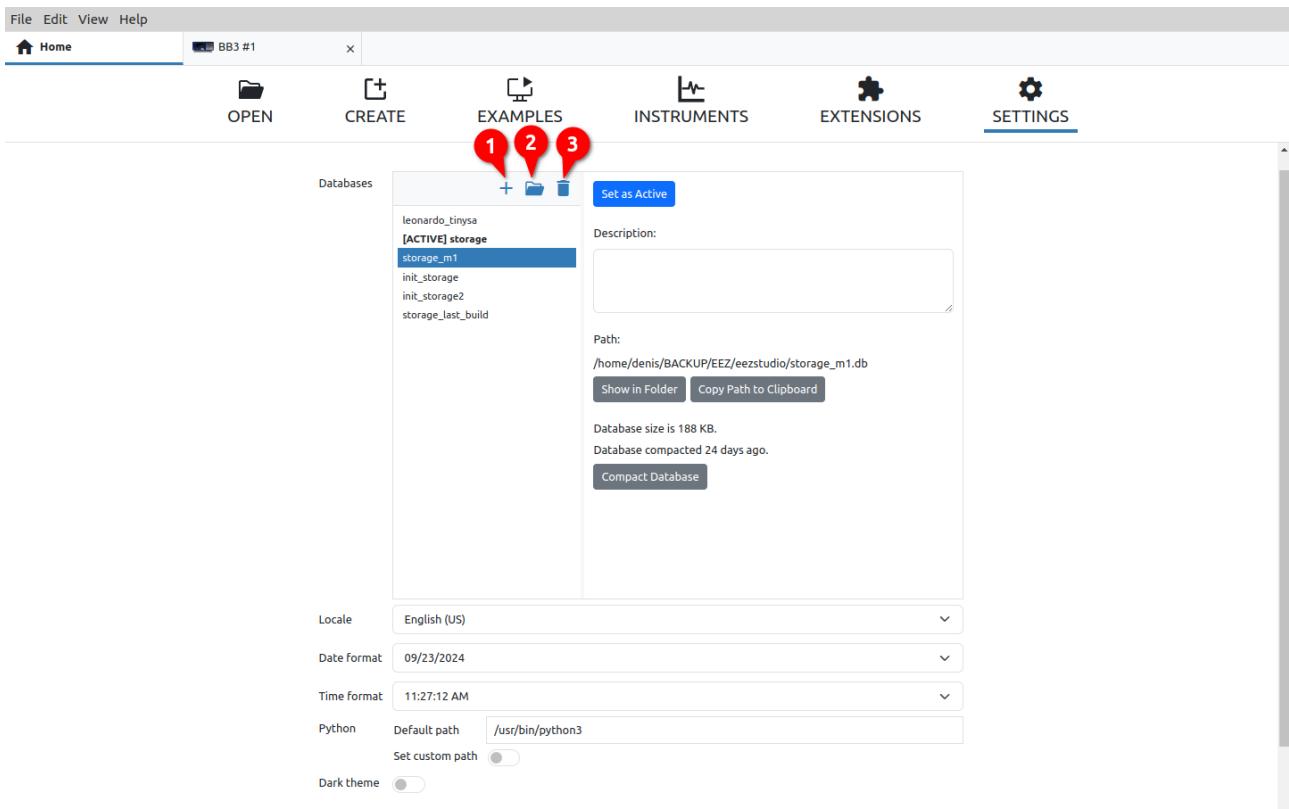


Fig. 4: Settings tab

Databases

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. Here we can also create a new database (1) that will be displayed in the list in which the currently active database has the prefix **[ACTIVE]**. To set another database as active, use the *Set as Active* button.

To display the folder on the disk where the database is located, use option (2), or to delete the currently selected database, use option (3).

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

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.

Python

Since it is possible to have multiple python versions and/or installations, this causes some confusion/problems when e.g. a user runs `pip` commands from a terminal to install dependencies and EEZ Studio runs a different python version during the build of LVGL when converting images. Here we can choose between the location of the default or custom Python installation/version.

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).

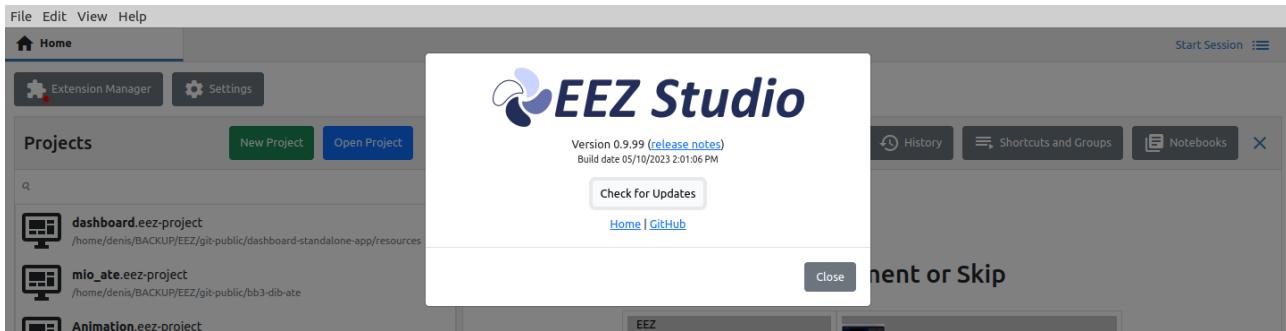


Fig. 5: About page

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

EEZ Studio Project.....	P.1
P1. Home page project sections.....	P.7
P1.1. EEZ Studio project types.....	P.7
P1.2. Create new project.....	P.8
P1.3. Project basic settings.....	P.9
P1.4. Additional steps for creating EEZ BB3 projects.....	P.9
P2. Project editor overview.....	P.13
P2.1. Project editor workspace.....	P.13
P2.2. Display of the page in the editor.....	P.14
P2.3. Panel moving and docking.....	P.14
P2.4. Border tabssets.....	P.16
P3. Project editor modes.....	P.17
P3.1. Toolbar overview.....	P.17
P3.2. Toolbar in Edit mode.....	P.18
P3.3. Feature buttons.....	P.26
P4. Project editor panels.....	P.27
P4.1. Panel items.....	P.27
P4.2. Right-click menu.....	P.28
P4.3. Edit mode panels overview.....	P.29
P4.3.1. Search and Replace.....	P.30
P4.3.2. Find and Replace.....	P.31
P4.4. Debug mode panels overview.....	P.31
P5. Project editors/viewers.....	P.33
P5.1. Editors.....	P.33
P5.1.1. Page editor.....	P.33
P5.1.2. User Actions.....	P.33
P5.1.3. User Widgets.....	P.34
P5.1.4. Font editor.....	P.34
P5.1.5. Shortcuts (EEZ-GUI only).....	P.35
P5.1.6. MicroPython (EEZ BB3 only).....	P.35
P5.1.7. Readme.....	P.35
P5.1.8. Settings.....	P.36
P5.2. Viewers.....	P.37
P5.2.1. Page viewer.....	P.37
P5.2.2. Action viewer.....	P.38
P6. EEZ Flow.....	P.39
P6.1. EEZ Flow basic concepts.....	P.39
P6.2. Flow execution.....	P.40
P6.3. Flow examples.....	P.42
P7. Project editing.....	P.43
P7.1.1. Connecting Flow components.....	P.46
P7.1.2. Copy & Paste between two projects.....	P.48
P7.2. Working with Widgets.....	P.49
P7.2.1. Widget component's items.....	P.49
P7.2.2. Creating a User Widget.....	P.50
P7.3. Working with Actions.....	P.52
P7.3.1. Action component's items.....	P.53
P7.3.2. Creating a User Action.....	P.53
P8. Variables.....	P.57
P8.1. Variables usage in the project with EEZ Flow enabled.....	P.58

P8.2. Variable types.....	P.59
P8.2.1. Basic/Primitive types.....	P.59
P8.2.2. Structures.....	P.59
P8.2.3. Enums.....	P.59
P8.2.4. Objects.....	P.59
P8.2.5. Arrays.....	P.59
P8.2.6. JSON objects.....	P.59
P8.2.7. Expressions.....	P.59
P8.2.8. Literals.....	P.60
P8.2.9. Binary Operators.....	P.61
Addition +.....	P.61
Subtraction -.....	P.61
Multiplication *	P.62
Division /.....	P.62
Remainder %.....	P.62
Left shift <<.....	P.62
Right shift >>.....	P.62
Binary AND &.....	P.62
Binary OR 	P.63
Binary XOR ^.....	P.63
P8.2.10. Logical operators.....	P.63
P8.2.11. Unary operators.....	P.63
P8.2.12. Conditional (ternary) operator.....	P.63
P8.3. Functions.....	P.63
P8.3.1. System.....	P.63
System.getTick.....	P.63
P8.3.2. Flow.....	P.64
Flow.index.....	P.64
Flow.isPageActive.....	P.64
Flow.pageTimelinePosition.....	P.64
Flow.makeValue.....	P.64
Flow.makeArrayValue.....	P.64
Flow.languages.....	P.65
Flow.translate.....	P.65
Flow.parseInteger.....	P.65
Flow.parseFloat.....	P.65
Flow.parseDouble.....	P.65
P8.3.3. Date.....	P.66
Date.now.....	P.66
Date.toString.....	P.66
Date.toLocaleString.....	P.66
Date.fromString.....	P.66
Date.getYear.....	P.66
Date.getMonth.....	P.66
Date.getDay.....	P.67
Date.getHours.....	P.67
Date.getMinutes.....	P.67
Date.getSeconds.....	P.67
Date.getMilliseconds.....	P.67
Date.make.....	P.68
P8.3.4. Math.....	P.68
Math.sin.....	P.68
Math.cos.....	P.68
Math.pow.....	P.68
Math.log.....	P.69
Math.log10.....	P.69
Math.abs.....	P.69
Math.floor.....	P.69
Math.ceil.....	P.69
Math.round.....	P.70
Math.min.....	P.70

Math.max.....	P.70
P8.3.5. String.....	P.70
String.length.....	P.70
String.substring.....	P.70
String.find.....	P.71
String.padStart.....	P.71
String.split.....	P.71
P8.3.6. Array.....	P.71
Array.length.....	P.71
Array.slice.....	P.72
Array.allocate.....	P.72
Array.append.....	P.72
Array.insert.....	P.72
Array.remove.....	P.73
Array.clone.....	P.73
P8.3.7. JSON.....	P.73
JSON.get.....	P.73
JSON.clone.....	P.73
P8.3.8. LVGL.....	P.73
LVGL.MeterTickIndex.....	P.73
P8.4. Expression Builder.....	P.74
P9. Styles and Color themes.....	P.75
P9.1. Overview.....	P.75
P9.2. Style properties.....	P.75
P9.3. Project Styles.....	P.76
P9.3.1. Creating a new Style.....	P.77
P9.4. Style hierarchy.....	P.77
P9.4.1. Setting the Style attribute color from the palette.....	P.78
P9.4.2. Setting the Style attribute color using the Color theme.....	P.79
P9.5. Style attributes.....	P.80
P9.5.1. EEZ-GUI project.....	P.80
P9.5.2. Dashboard project.....	P.81
P9.5.3. LVGL project.....	P.82
P9.5.4. Inheriting local Style attributes.....	P.84
P10. Bitmaps.....	P.85
P10.1. Adding a bitmap.....	P.85
P10.2. Bitmap properties.....	P.86
P10.3. Using a bitmap.....	P.87
P11. Fonts.....	P.89
P11.1. EEZ-GUI project fonts.....	P.89
P11.1.1. Add new font.....	P.89
P11.1.2. Add character.....	P.90
P11.2. LVGL project fonts.....	P.91
P11.2.1. Add new font.....	P.91
P11.2.2. Edit characters.....	P.93
P12. Texts.....	P.95
P12.1. Texts panel.....	P.95
P12.2. XLIFF Import/export.....	P.96
P13. Settings.....	P.99
P13.1. General.....	P.99
P13.2. Build.....	P.100
P13.2.1. Configurations.....	P.101
P13.2.2. Files.....	P.102

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 Open* section of the home page is shown in Fig. 1. which displays a searchable Recent Project List (RPL).

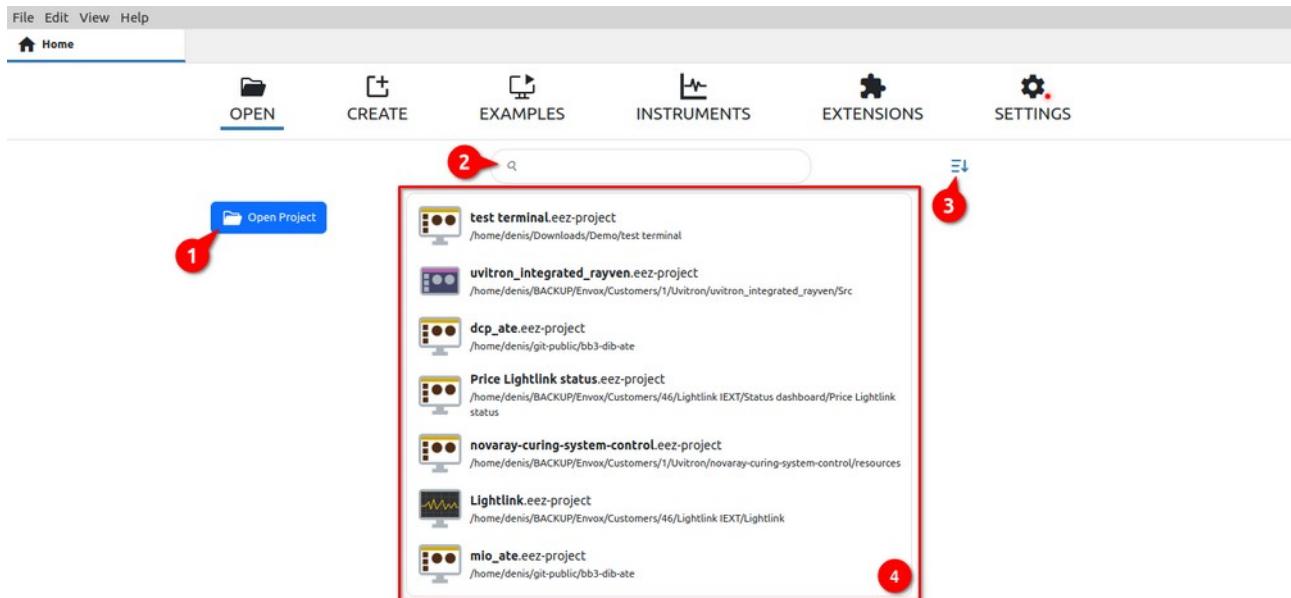


Fig. 1: Home page open project options

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

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/>

- **LVGL with EEZ Flow** – similar to the previous type embedded GUI application that uses LVGL (Light and Versatile Graphics Library) framework but with the addition of the EEZ Flow project development.
- **BB3 Applet** – GUI application that can be run on EEZ BB3. Program logic is created using EEZ Flow (flowchart-based programming).
- **BB3 MicroPython script** – GUI application that can be run on EEZ BB3. Program logic is created using MicroPython scripting.
- **Templates from gitea repository** – Various 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.

P1.2. Create new project

Creating a new project is possible by selecting the *Create* option from the Home page toolbar when the Create tab is displayed. (Fig. 2).

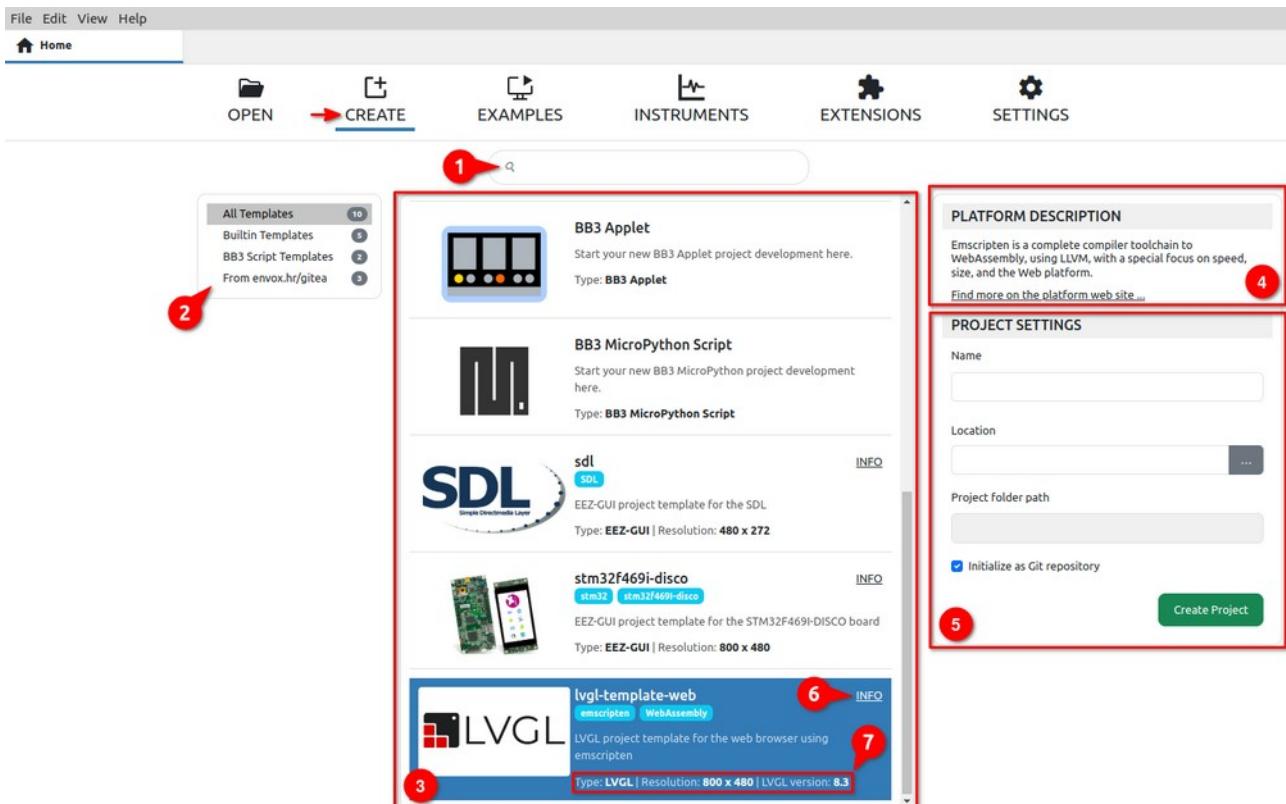


Fig. 2: Create new project

#	Option	Description
1	<i>Search</i>	Search by project name.
3	<i>Project list</i>	List of all projects within the selected category, grouped into expandable sublists.
3	<i>Project selector</i>	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.
4	<i>Platform description</i>	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.
5	<i>Project settings</i>	Project basic parameters (see below).
6	<i>Info</i>	If the project has a Git repository, this link will appear that takes you to the repository home page.

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.

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

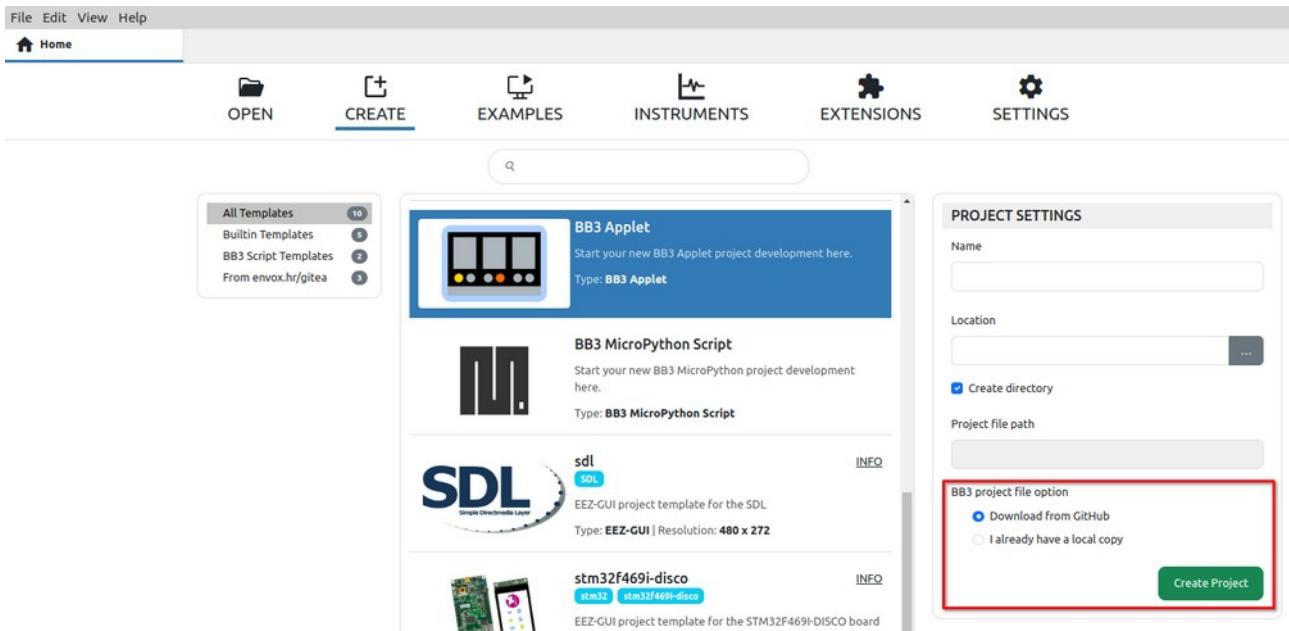


Fig. 3: EEZ BB3 applet new project additional option

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. 4).

EEZ Studio Reference Guide

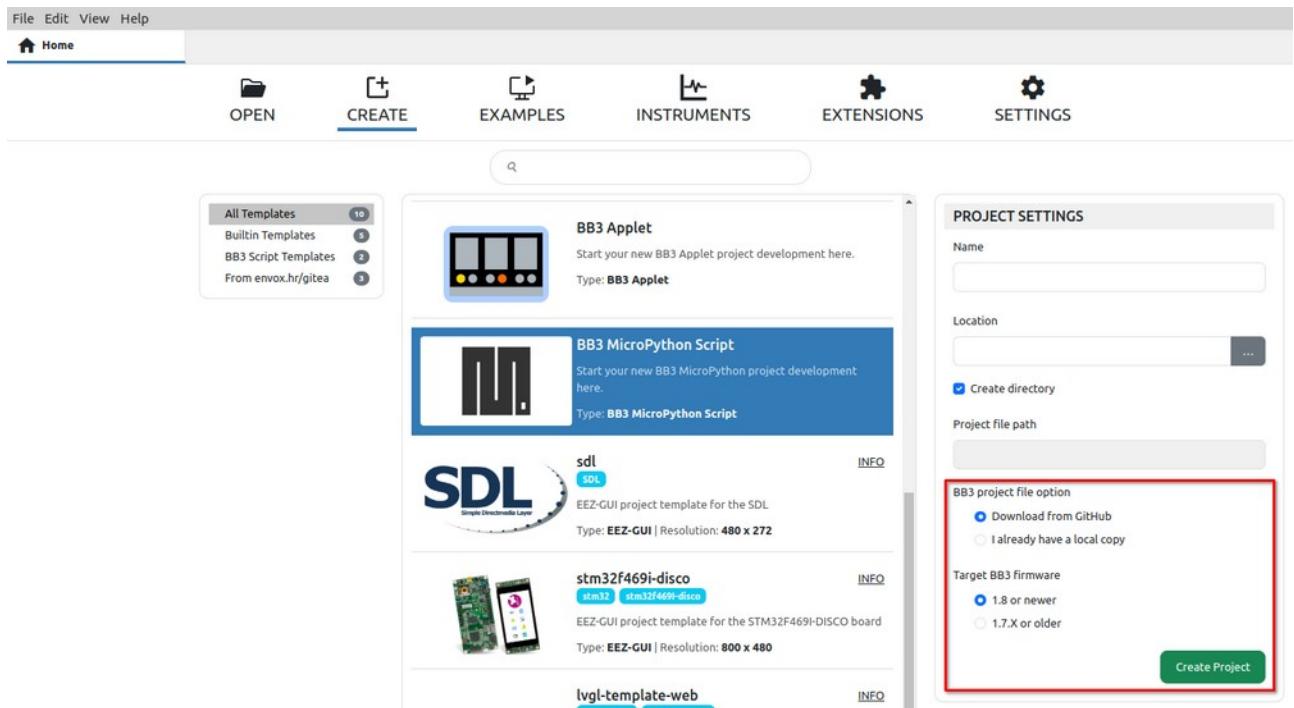


Fig. 4: EEZ BB3 MicroPython new project additional options

After all the basic parameters have been entered, the new project will be displayed in the project editor in *Edit* mode. Fig. 5 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.

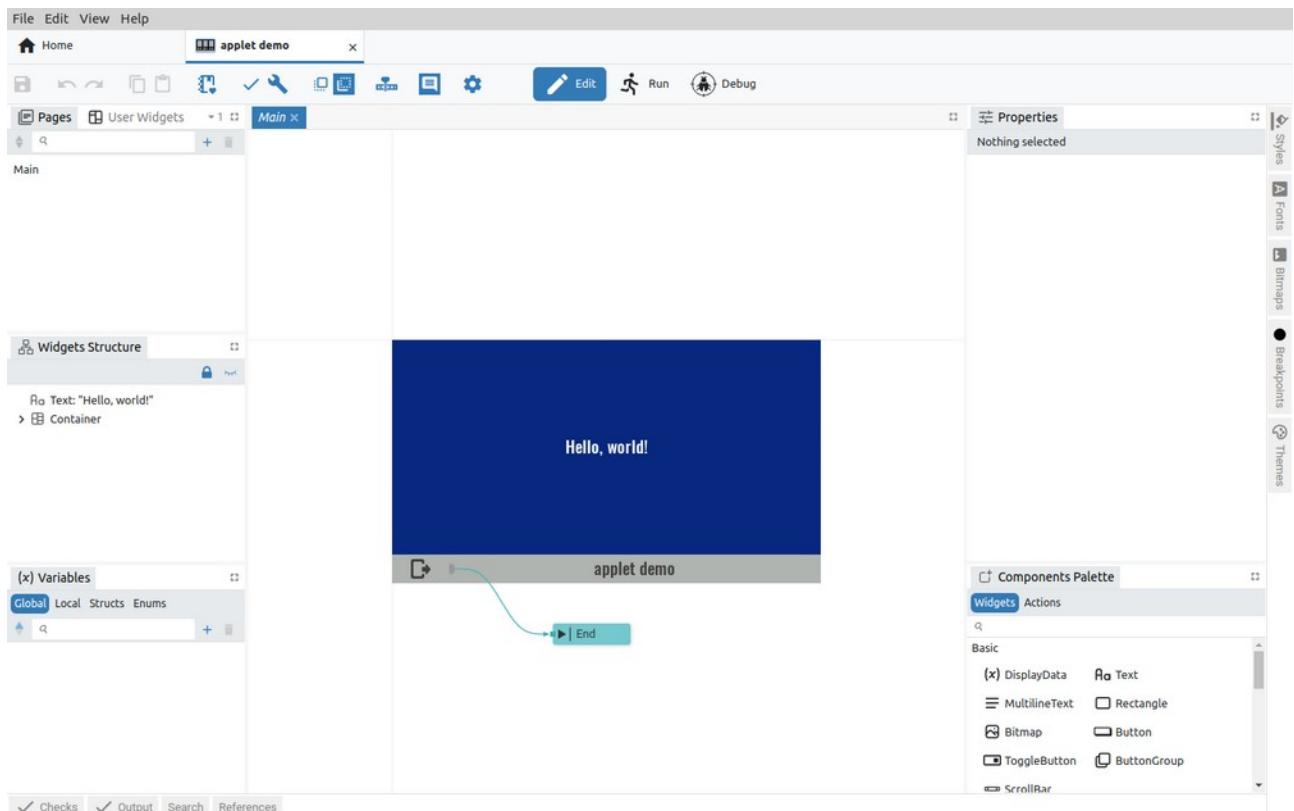


Fig. 5: Newly created project in Edit mode

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. 6.

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).

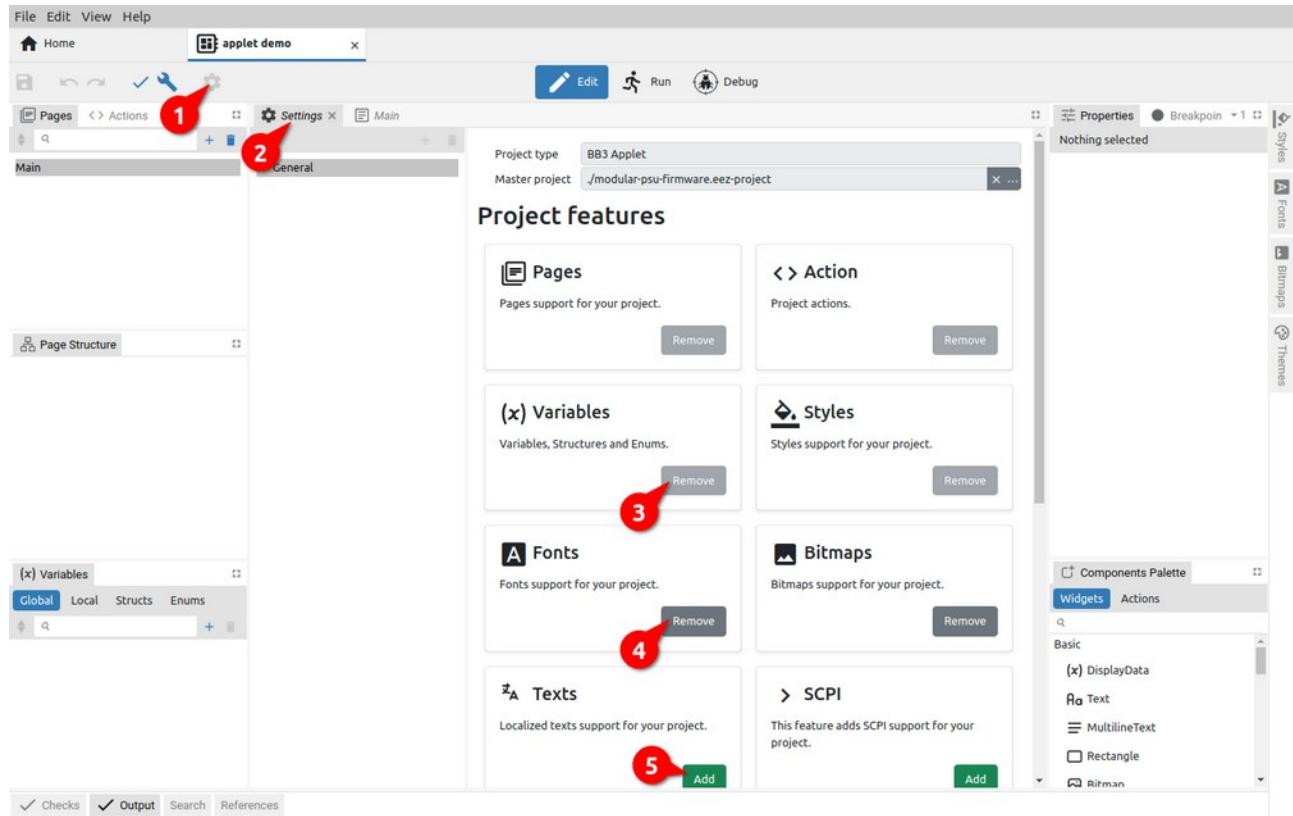


Fig. 6: Newly created project settings

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. 7 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.

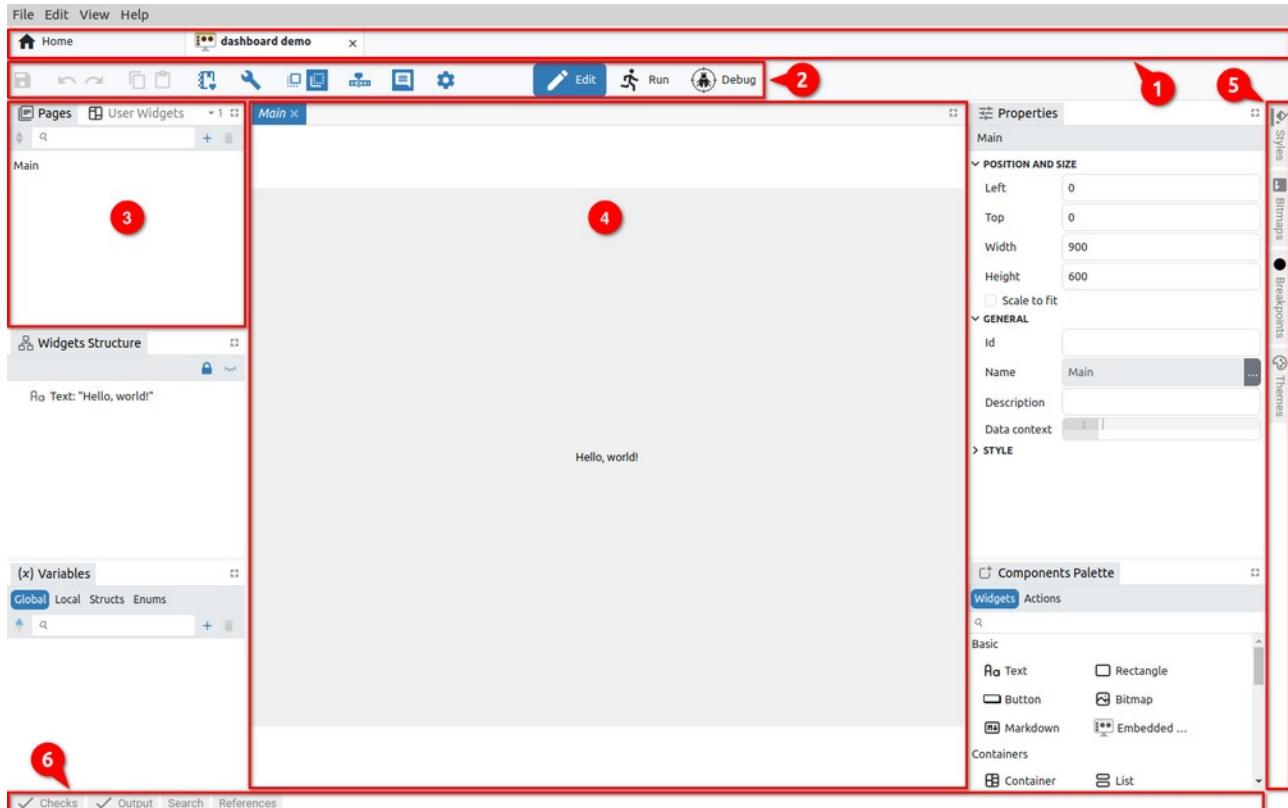


Fig. 7: Project editor sections

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

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

Panels and editors can be grouped within one or more tabssets. Tabssets are dockable and can be placed in the workspace e.g. (3) and (4) or along borders e.g. (5) and (6). The elements of the project editor shown in Fig. 7 are explained below.

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

4 Editor tabset

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

5 Right border tabset

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

6 Bottom border tabset

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

P2.2. Display of the page in the editor

In Fig. 8 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).

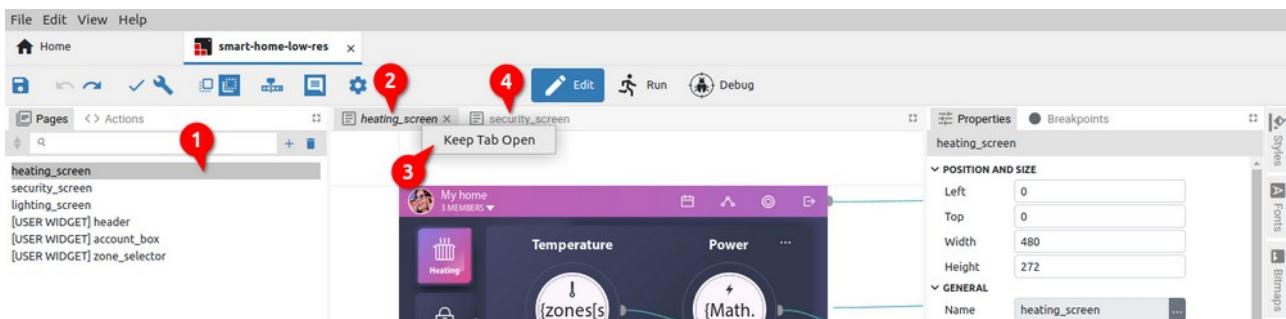


Fig. 8: Page editor tab locking

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. 9).

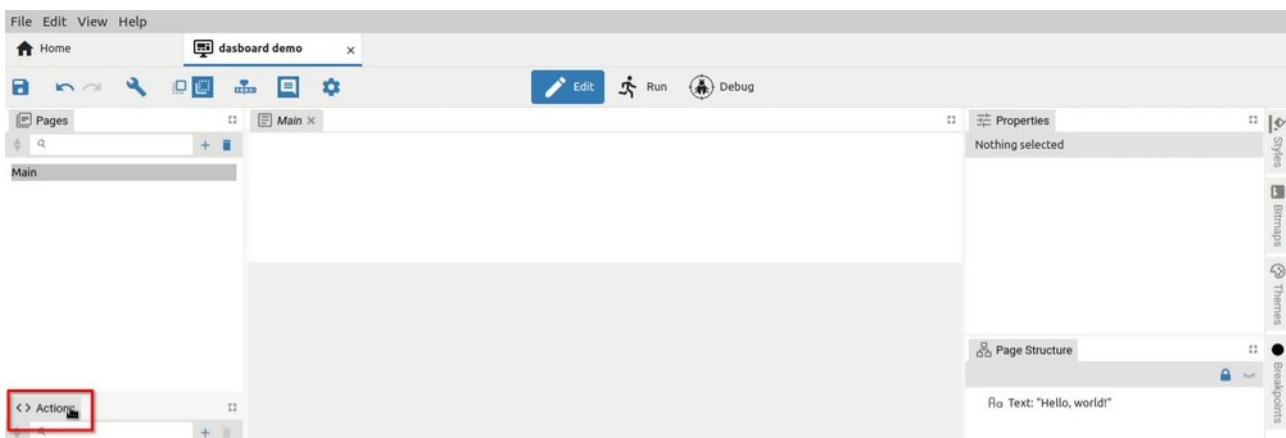


Fig. 9: Panel selection

The panel is now ready to move to another location. The cursor will change and marks will also appear on all four sides indicating the ability to dock into border tabs (Fig. 10).

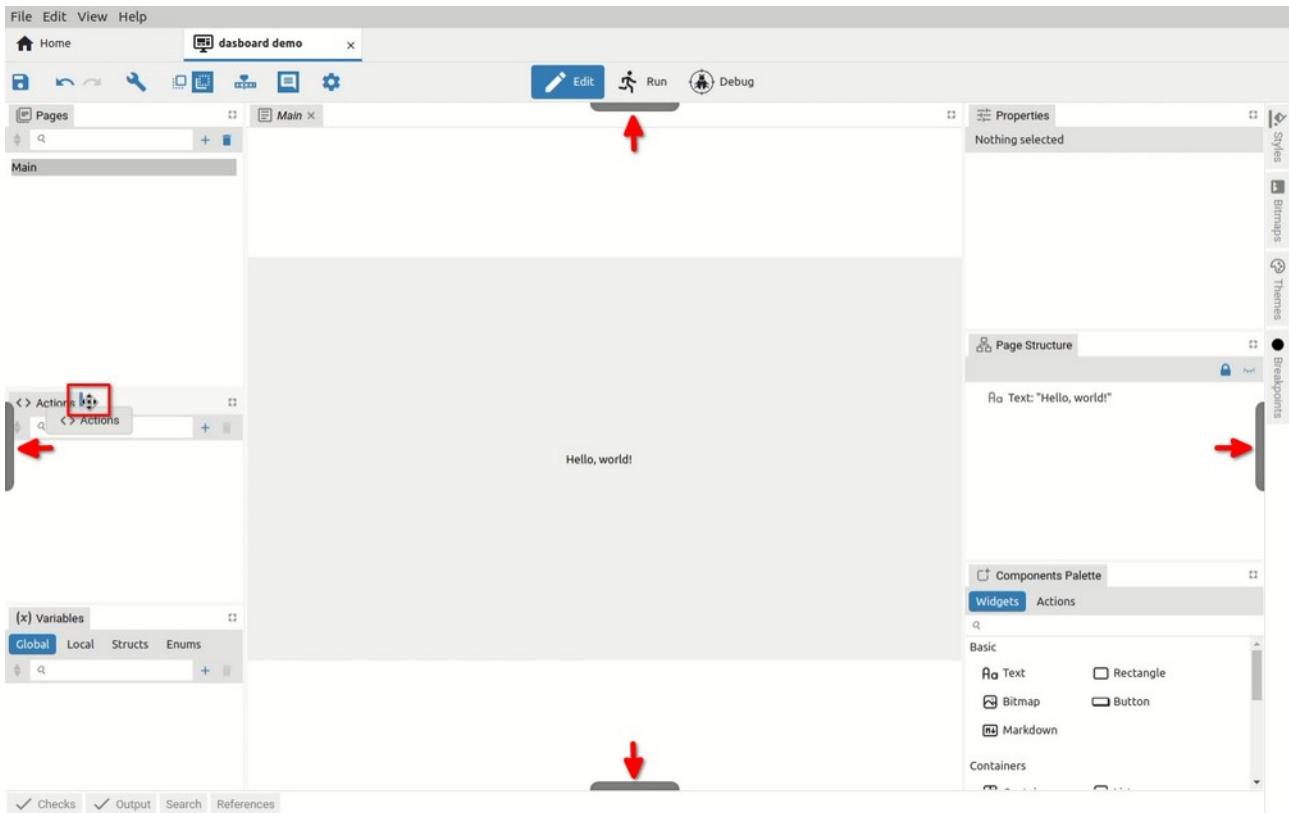


Fig. 10: Docking indicators for border tabs

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. 11. 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. 12.

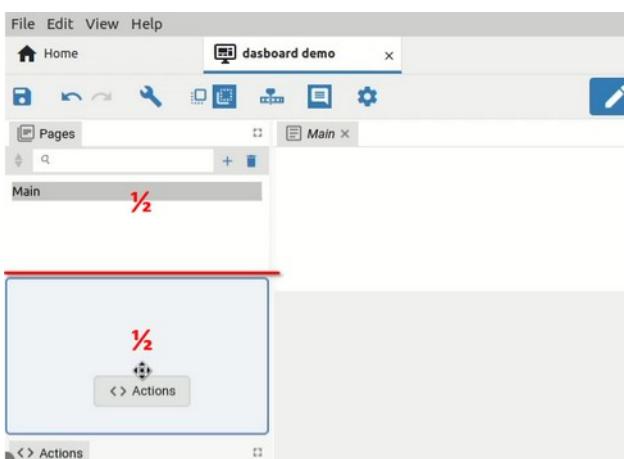


Fig. 11: Panel horizontal positioning

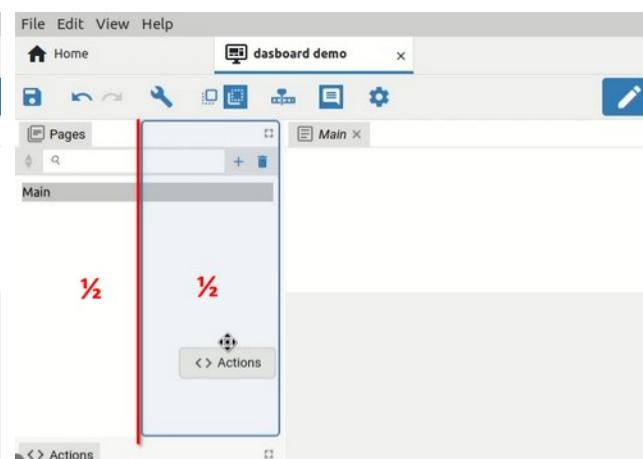


Fig. 12: Panel vertical positioning

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

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. 11 and Fig. 12.

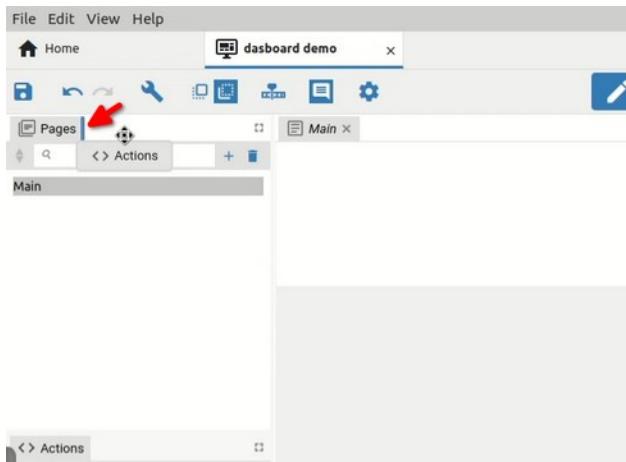


Fig. 13: Positioning in another tabset (1st method)

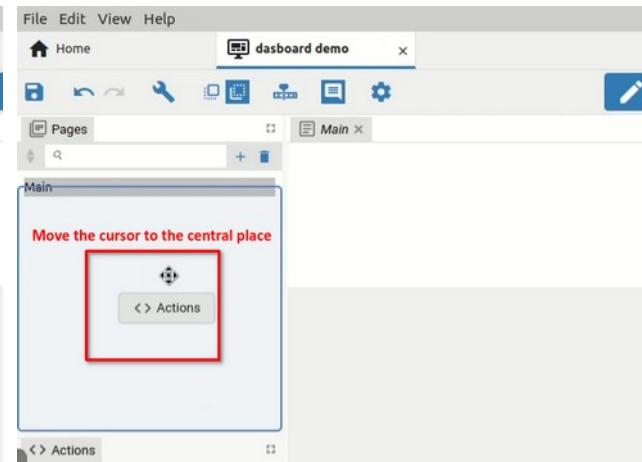


Fig. 14: Positioning in another tabset (2nd method)

Finally, when we have chosen where we want the selected panel to be displayed for docking, it will be necessary to release the mouse button. In our example, Actions will become a new tab in the tabset with Pages (Fig. 15).

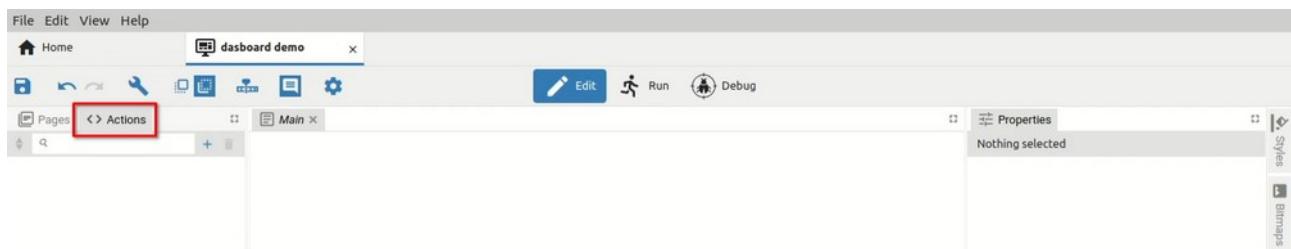


Fig. 15: Panel docking completed

P2.4. Border tabssets

Panels from border tabssets, unlike panels in the workspace, are displayed by clicking the tab (Fig. 16) and closed by clicking the tab again. Only one panel within a border tabset can be open at any time (Fig. 17).

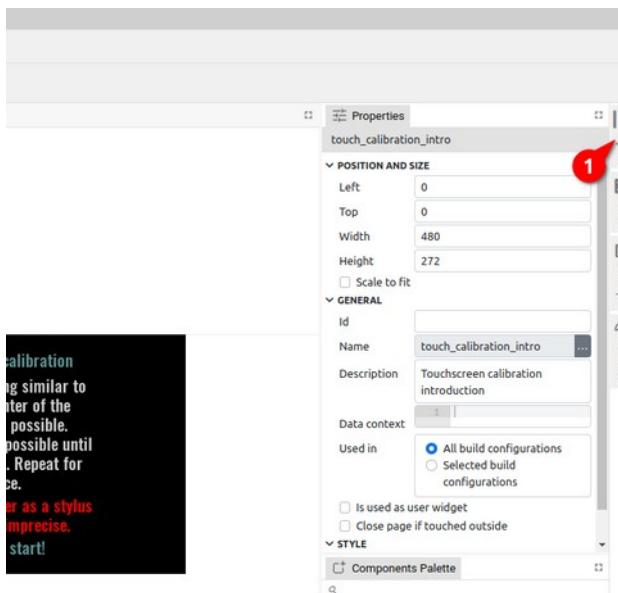


Fig. 16: Border tabs are closed

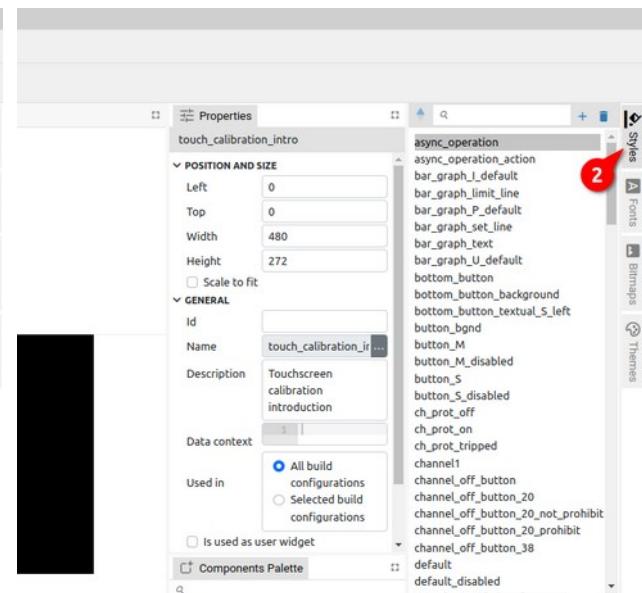


Fig. 17: The border tab is selected and opened

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

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. 18).

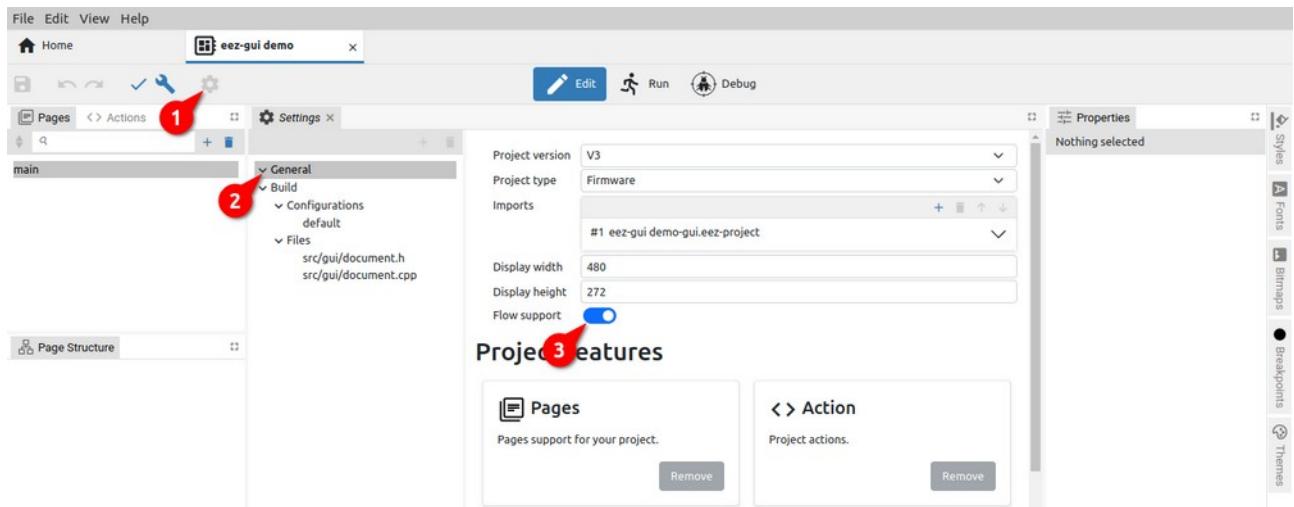


Fig. 18: Enabling EEZ Flow in project Settings

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

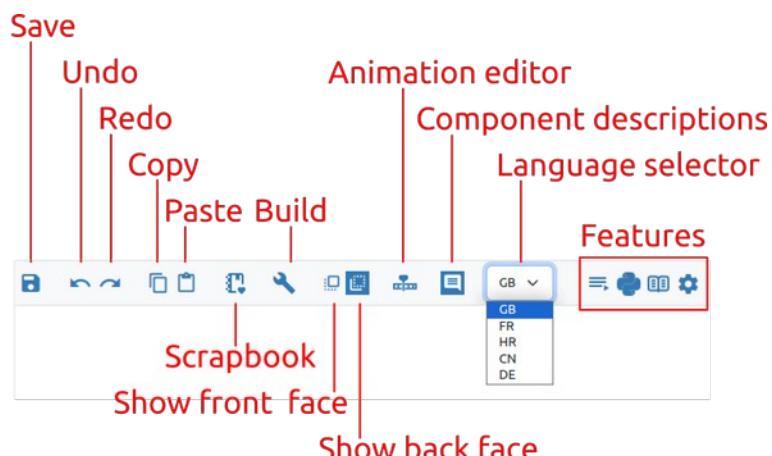


Fig. 19: Project toolbar (left part)



Fig. 20: Project toolbar (right part)

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	✓		
Copy	✓		
Paste	✓		
Scrapbook	✓		
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

Copy / Paste

The editor allows you to copy one or more project items using the *Copy* (1) and *Paste* (2) options shown in Fig. 22.

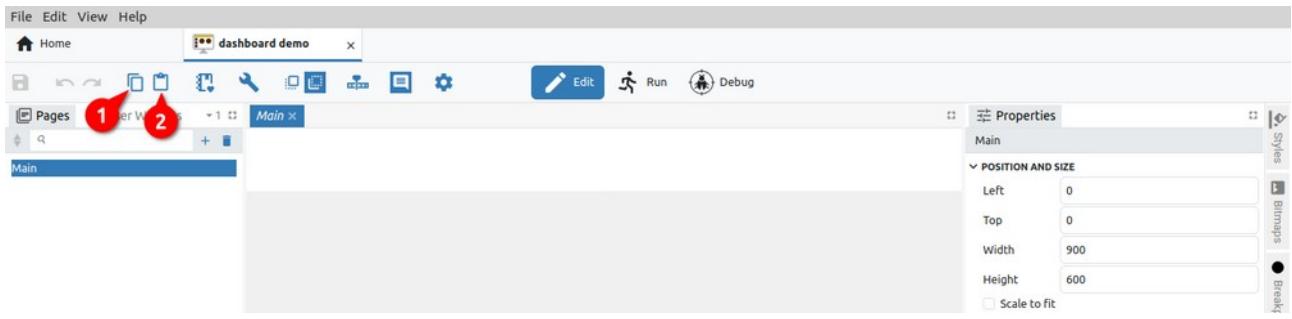


Fig. 22: Copy project item

In the example in Fig. 23, the *Main* page is selected, which will be named *Main_1* when copied (3).

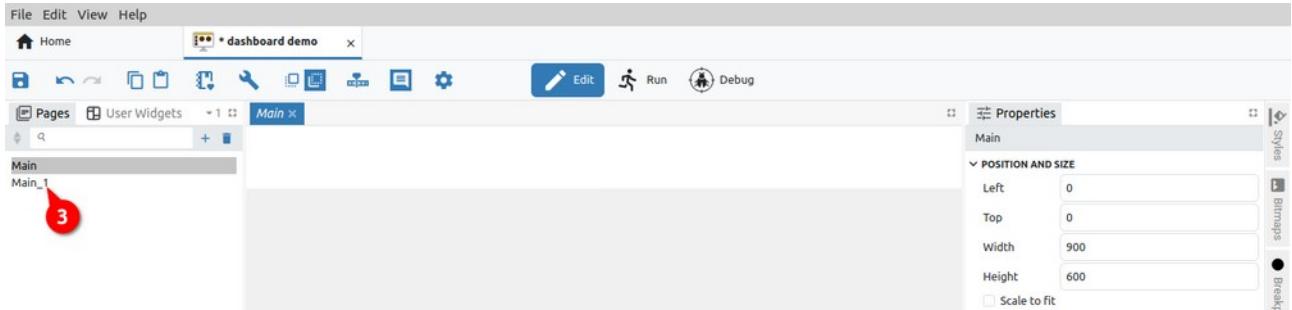


Fig. 23: Paste project item

In the event that the content of one project is copied to another project, a note will appear as shown in Fig. 24.

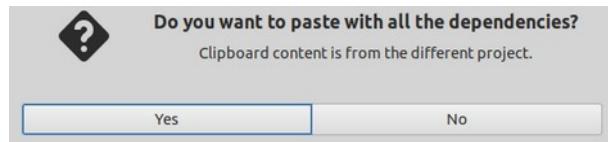


Fig. 24: Message when copying to a new project

In case we want to paste with all dependencies and there are one or more items with the same name in the destination project, a dialog box will appear to resolve the conflict. In our example, if we want to copy a variable named *global* of type *integer*, since there is a variable of the same name but of a different type (*double*) in the destination project, a conflict resolver will appear as in Fig. 25.

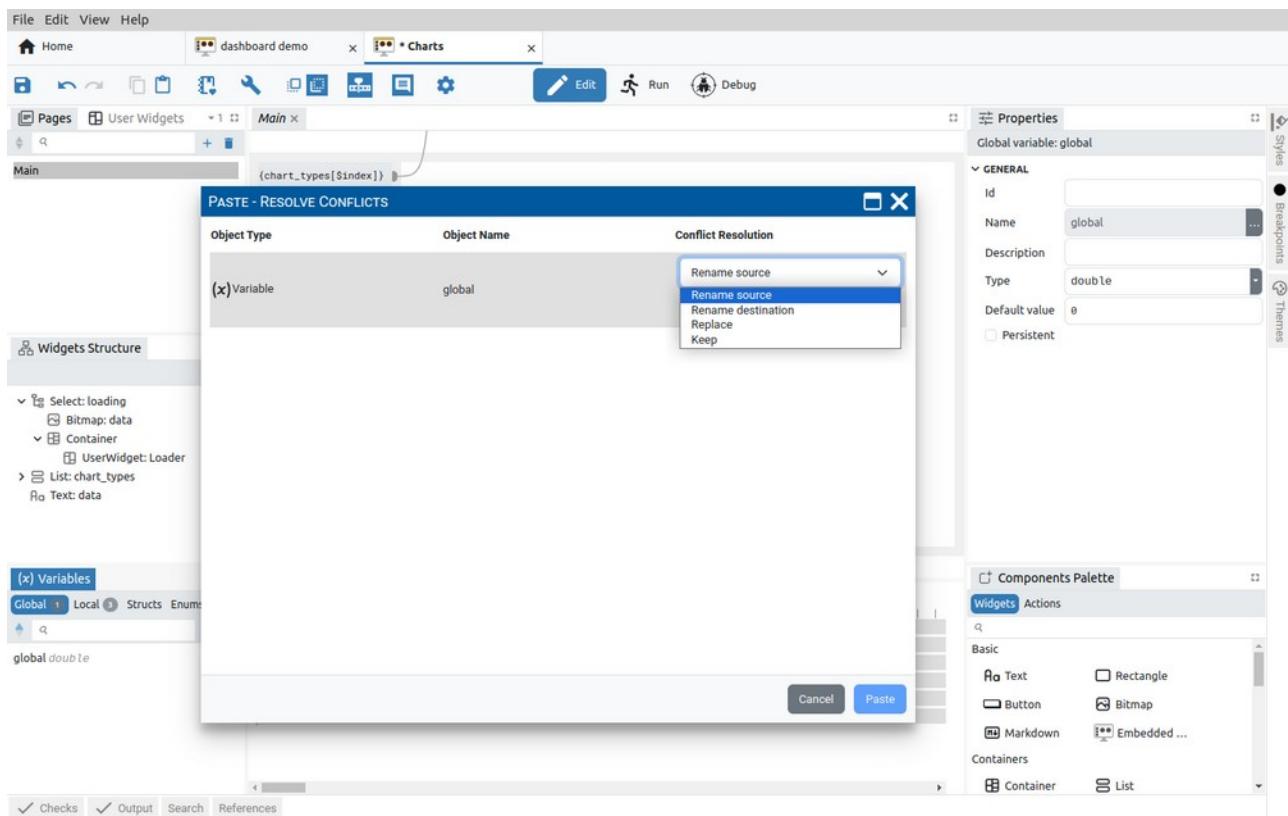


Fig. 25: Conflict resolution for paste operations

The following four options are offered to resolve the conflict:

Option	Description
<i>Rename source</i>	Use if you want to paste the item with a new name.
<i>Rename destination</i>	If we want the pasted item to keep its original name, use this option to change the existing item name in the destination project.
<i>Replace</i>	By selecting this option, the existing item in the destination project will be overwritten with the copied item.
<i>Keep</i>	In case you don't want to overwrite an existing item in the destination project with a new one, use this option.

If the question from Fig. 24 if we answer negatively, and there is an item with the same name in the destination project, a new item with a suffix in the name will be created when copying, which in our example will be `global_1`. (Fig. 26).

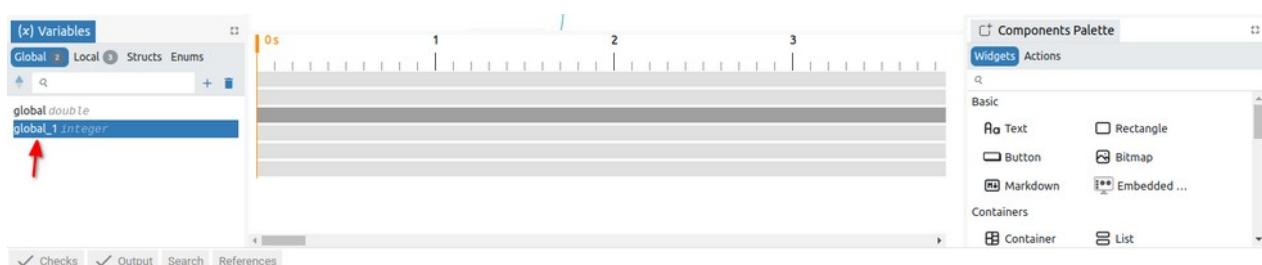


Fig. 26: Paste as a new item

Scrapbook

Copying items between projects allows them to be saved in the *Scrapbook*, where they will be saved until they are explicitly deleted, unlike the clipboard, where they will be saved only until the first restart of EEZ Studio. An additional advantage of Scrapbook over the clipboard is that it can have multiple "instances" and can contain a wide variety of content. Each Scrapbook is a separate file

(SQLite) on disk that can be freely copied, archived, etc.

It is possible to add any item of an existing project to the Scrapbook and it can contain an unlimited collection of items that can be copied to other projects and edited if necessary.

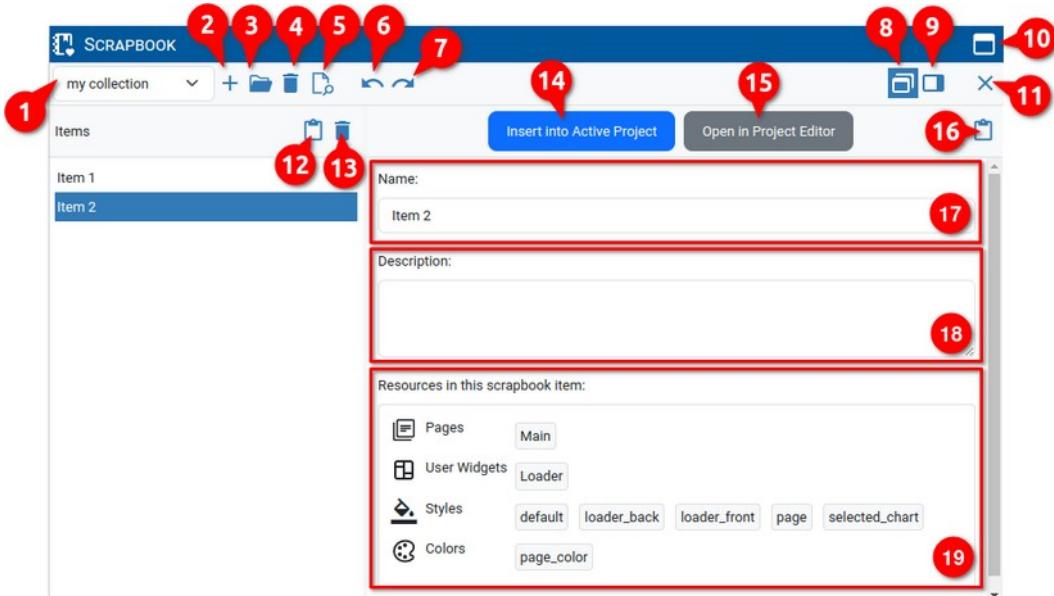


Fig. 27: Project scrapbook

#	Option	Description
1	<i>List of scrapbook files</i>	List of <i>Default</i> scrapbook and all user created scrapbook files.
2	<i>Create a new Scrapbook file</i>	Creating a new user scrapbook file.
3	<i>Open a Scrapbook file</i>	Opening an existing scrapbook file from disk (the file extension is <code>.eez-scrapbook</code>). A successfully loaded scrapbook will be added to the scrapbook list (1).
4	<i>Delete Scrapbook file</i>	Removing a scrapbook file from the list of scrapbooks (the file will not be deleted from disk).
5	<i>Show Scrapbook file in File explorer</i>	Open the folder on the disk that contains the scrapbook files.
6	<i>Undo</i>	Undo operation in scrapbook.
7	<i>Redo</i>	Redo operation in scrapbook.
8	<i>Undock into separate window</i>	Display scrapbook in free floating window (default view).
9	<i>Dock to the side</i>	Docking of the scrapbook display within the project editor.
10	<i>Maximize</i>	Maximizing the scrapbook view.
11	<i>Close</i>	Close the scrapbook view.
12	<i>Create a new Scrapbook item from the Clipboard</i>	Copying the contents of the system Clipboard into a newly created item that will be displayed in the list of items and whose name (17) and description (18) can be edited as desired.
13	<i>Delete this item</i>	Deleting the selected item from the list of items.
14	<i>Insert into Active Project</i>	Paste the content of the item into the selected project. In case there is a conflict, the conflict resolver will open (similar to Fig. 25).
15	<i>Open in Project Editor</i>	Since the scrapbook file is a valid EEZ Studio project, it can be opened for viewing and editing using this option.

16 Paste Clipboard content into selected Scrapbook item

Adding the contents of the system Clipboard to an existing item. For example, item already has the variable `global`:

Resources in this scrapbook item:

(x) Variables global

If there is `global_1` in the Clipboard, using this option, that variable will appear in the same section (Variables).

Resources in this scrapbook item:

(x) Variables global global_1

17 Name

The name of the scrapbook item.

18 Description

Description of the scrapbook item.

19 Resources in this Scrapbook item

List of all resources that the selected item contains.

Check

All project elements are checked without building the executable code. The Results are displayed in the *Output* panel (Fig. 28).

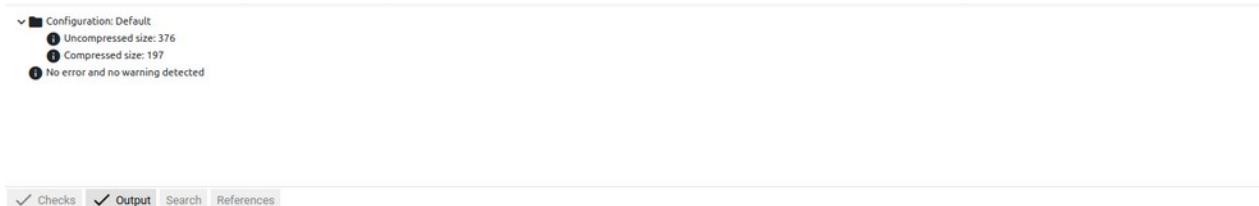


Fig. 28: Results of project checking

Build

Build the executable code after checking all the elements of the project. The results are displayed in the *Output* panel (Fig. 29).

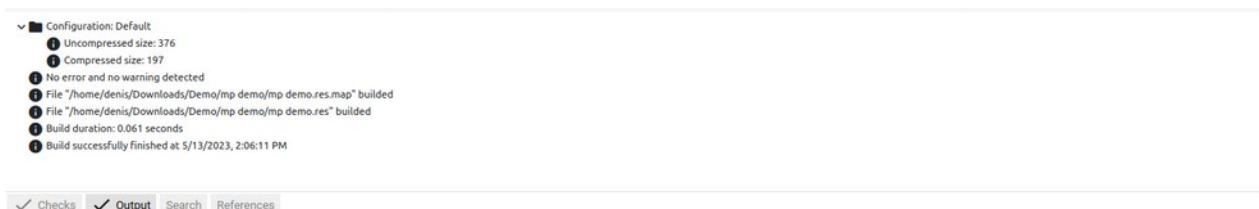


Fig. 29: Project build results

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.

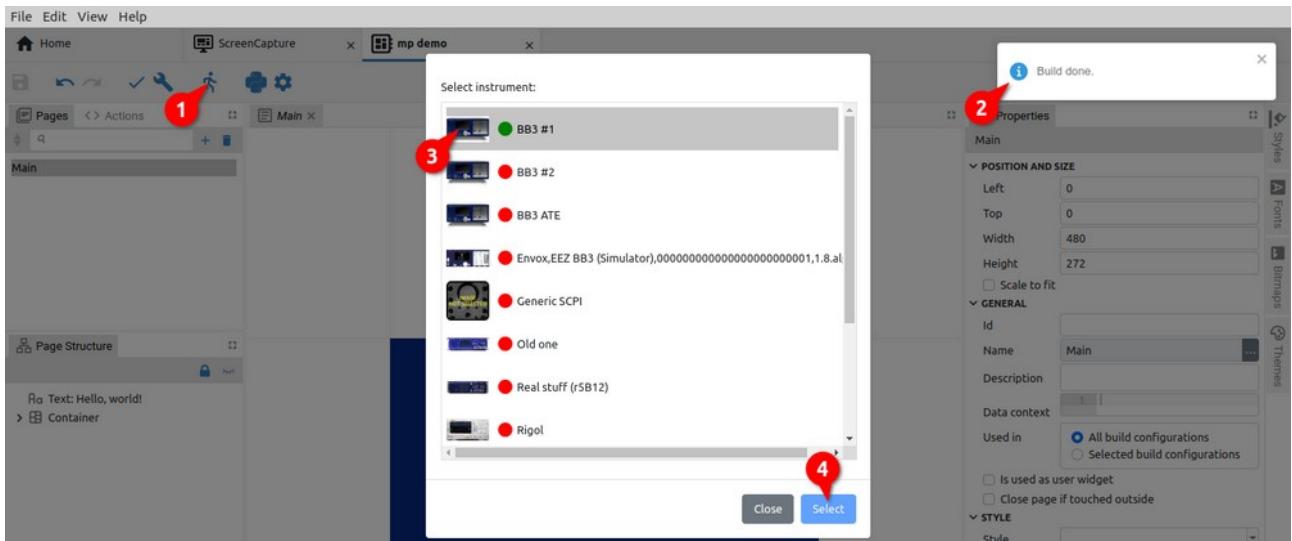


Fig. 30: Selection of target EEZ BB3 for project execution

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. 31 the serial interface is selected.

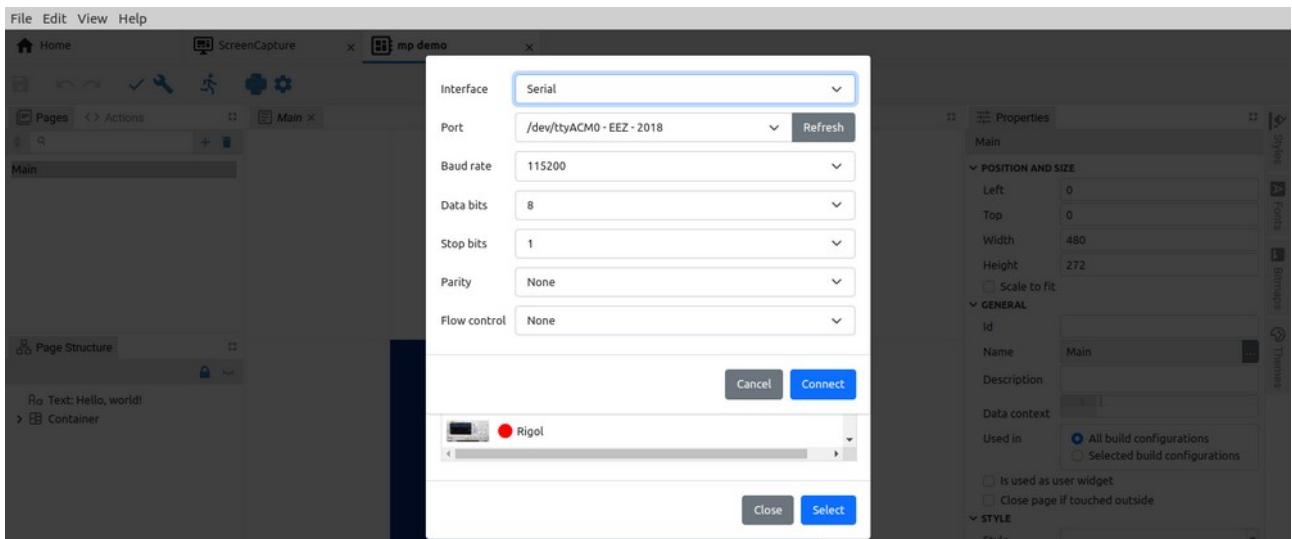


Fig. 31: Selection of target EEZ BB3 for project execution

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. 32).



Fig. 32: 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. 33).

EEZ Studio Reference Guide

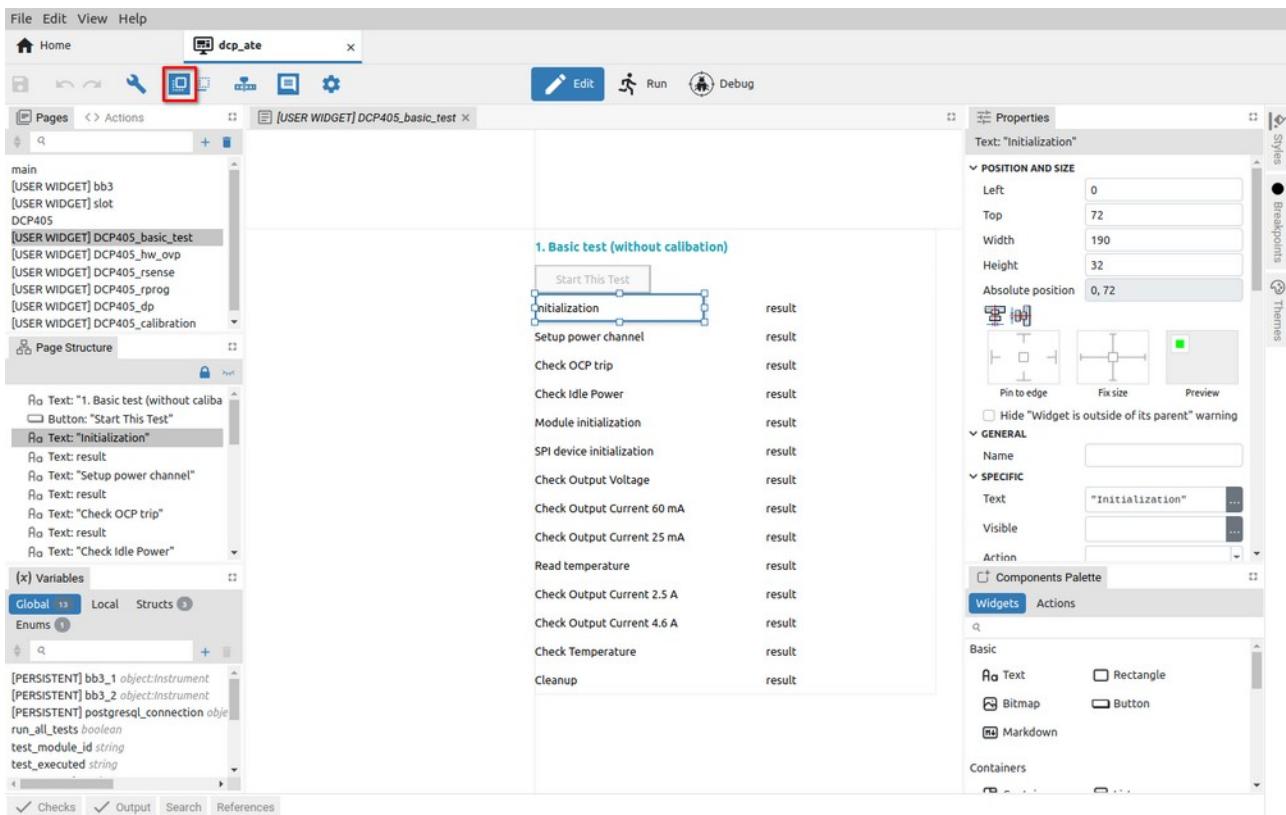


Fig. 33: Project page in front face view

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.34).

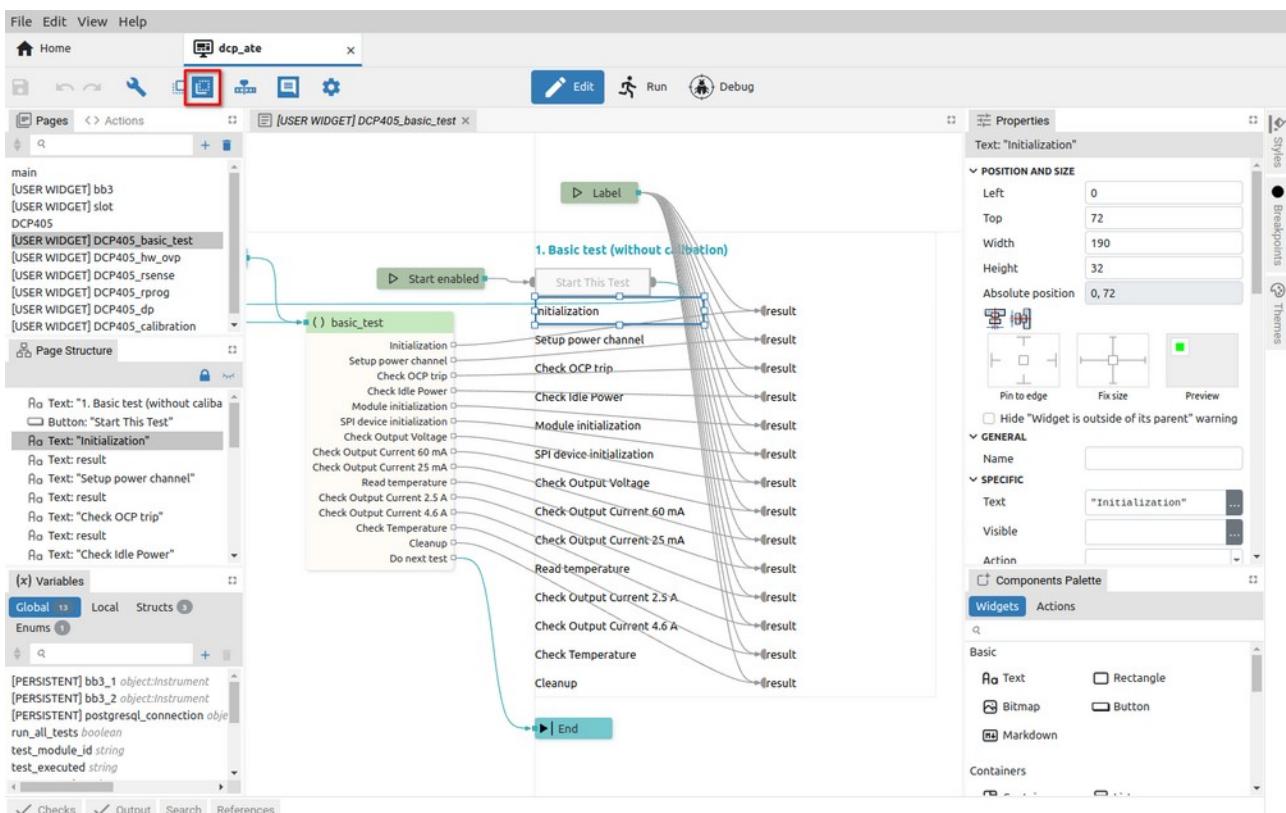


Fig. 34: Project page in back face view

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. 17).

The animation timeline editor is displayed in the space below the page editor (Fig. 35).

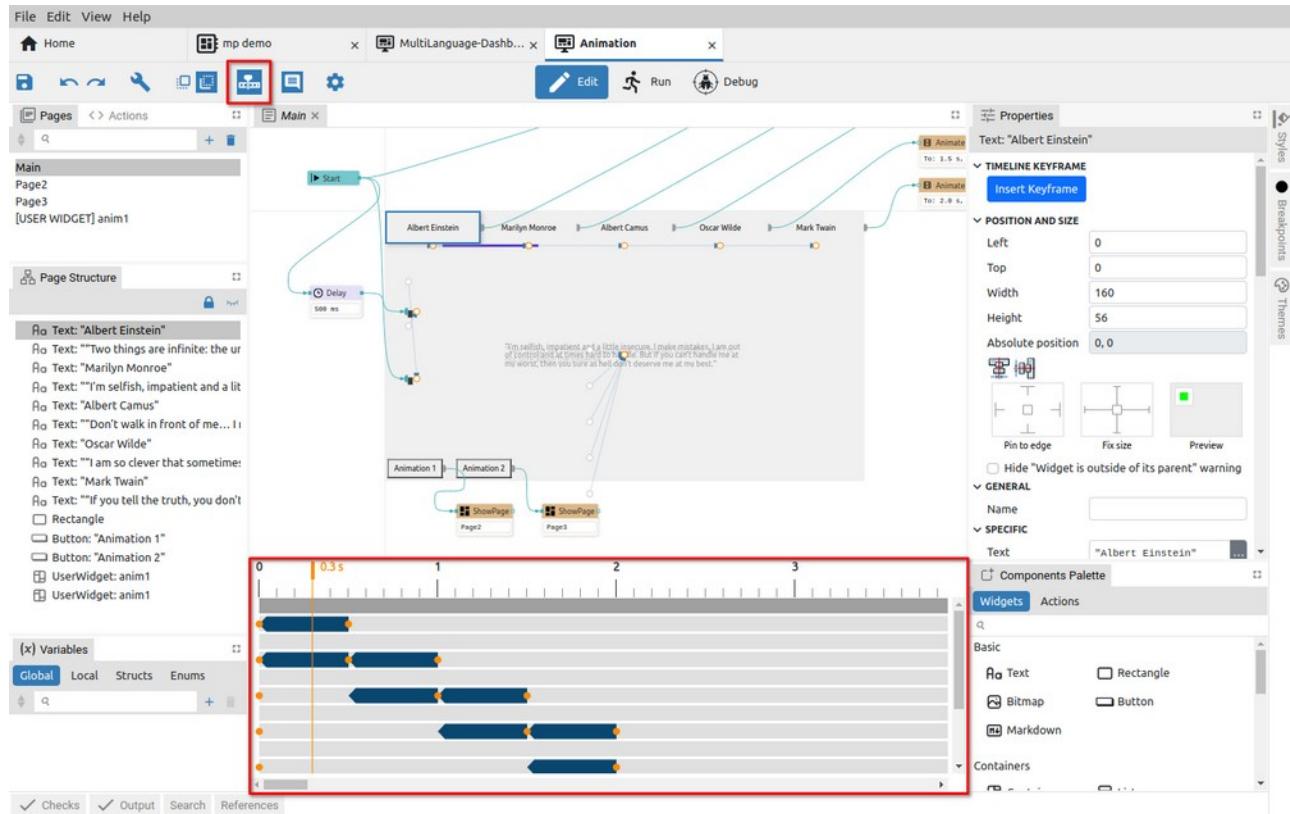


Fig. 35: Project animation timeline editor

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. 36).

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

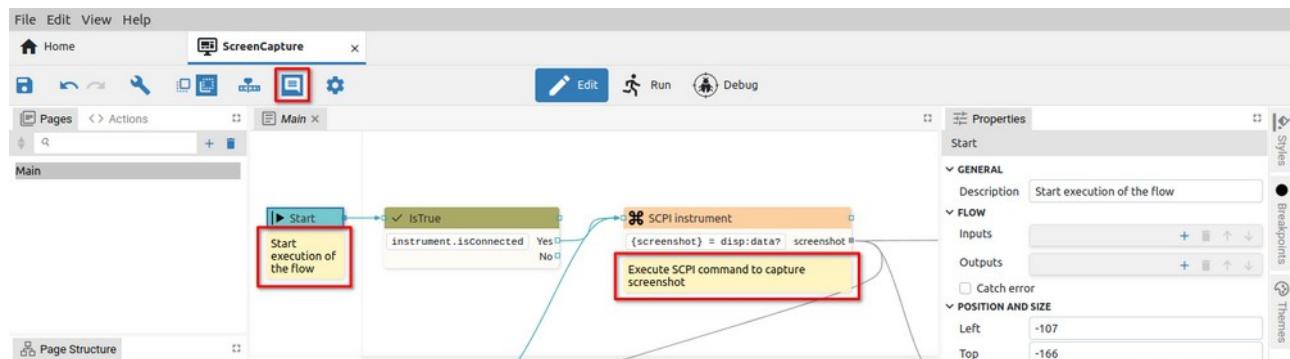


Fig. 36: Component descriptions are visible

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. 37 French (FR) is selected.

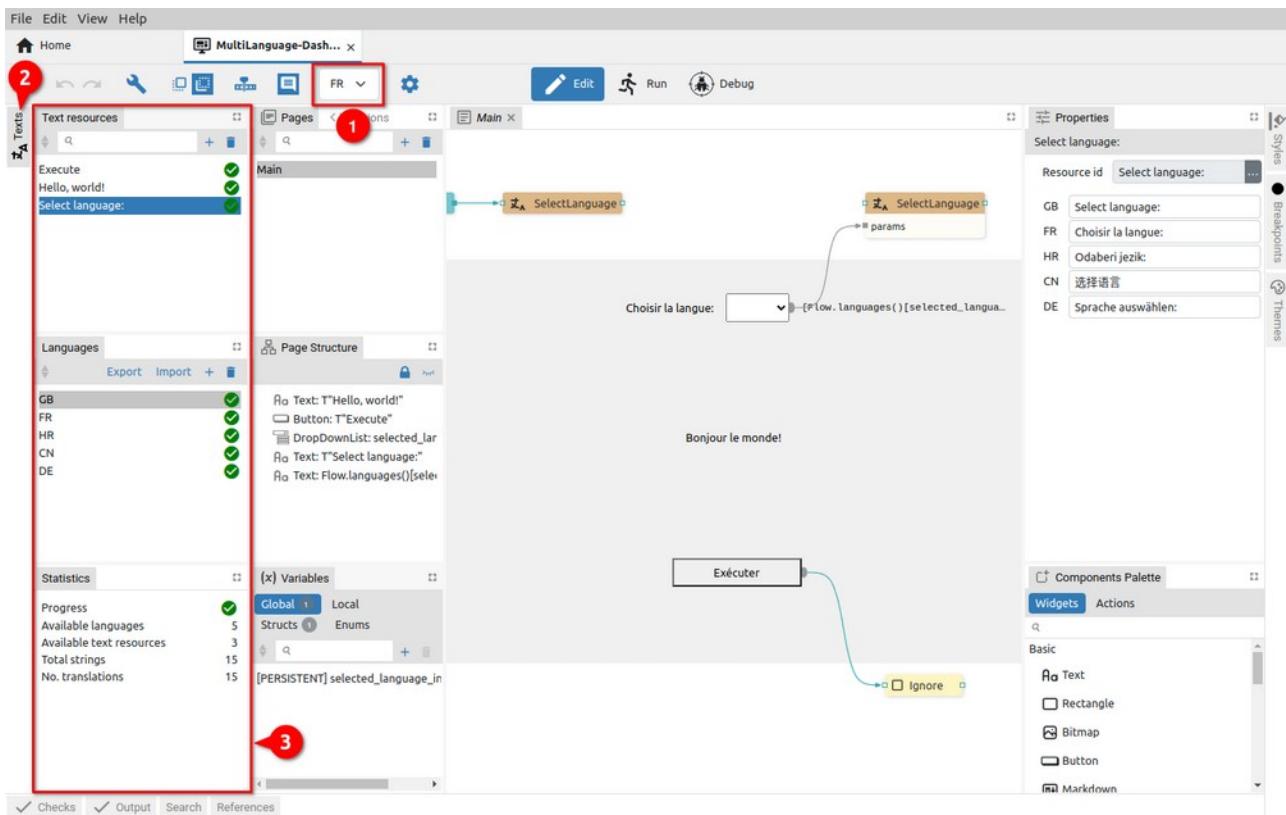


Fig. 37: Project language selector

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. 38 and described below. As you can see, different panels can have different number of items.

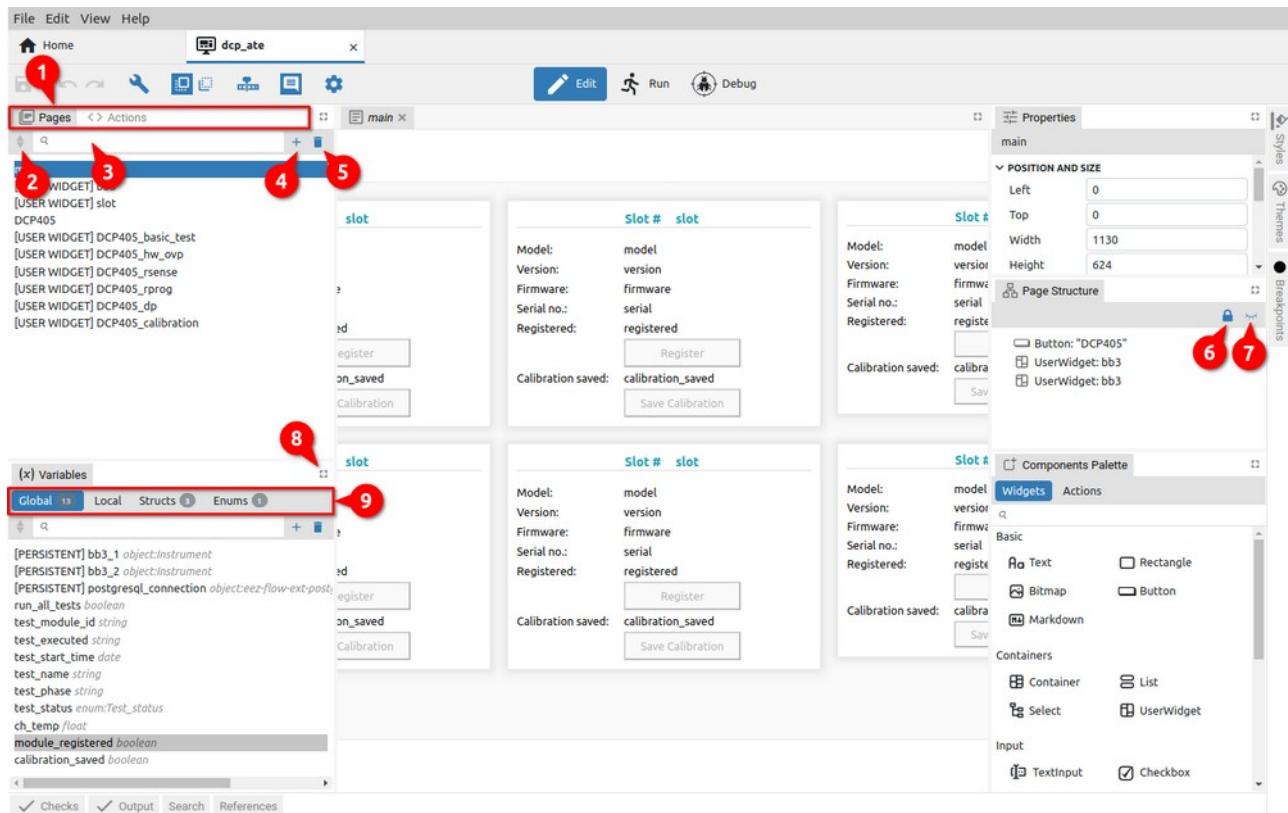


Fig. 38: Panel items

Item

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

Description

Selecting panels within a tabset.

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

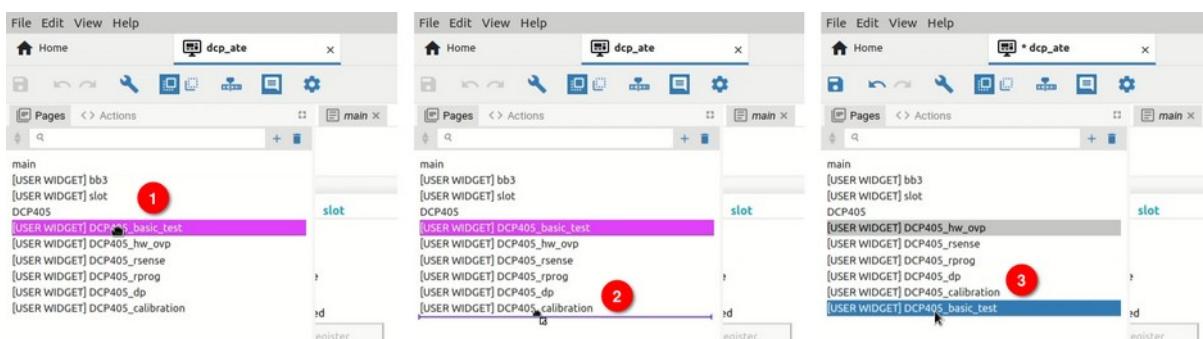


Fig. 39: Changing item position

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. 39).

- | | |
|----------------------|--|
| 3 <i>List filter</i> | Filtering items in the list according to the search term.
If something is entered in the list filter box, then drag & drop in the list of items is disabled when User sorting order is selected. |
| 4 <i>Add item</i> | Adding a new element to the panel. Opens a new dialog box with one or more parameters depending on the type of item. The name of the new item must be unique. Example in Fig. 40 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.

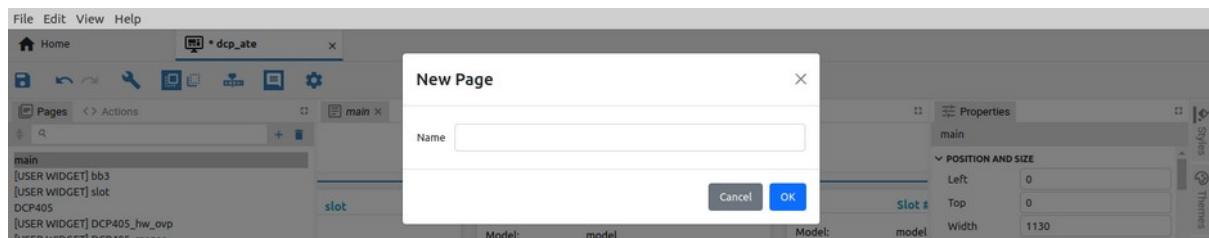


Fig. 40: Adding a new item (Page)

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

P4.2. Right-click menu

Right-click opens a context menu that generally contains options as in Fig. 41. The right-click menu for Widgets (Fig. 42) has a few more options and will be described in Chapter P7.2.2.

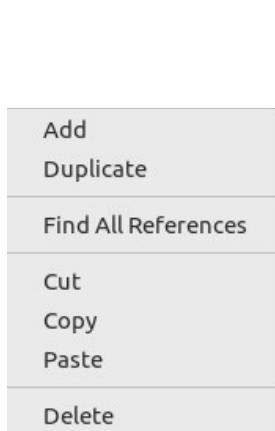


Fig. 41: Right-click menu (common)

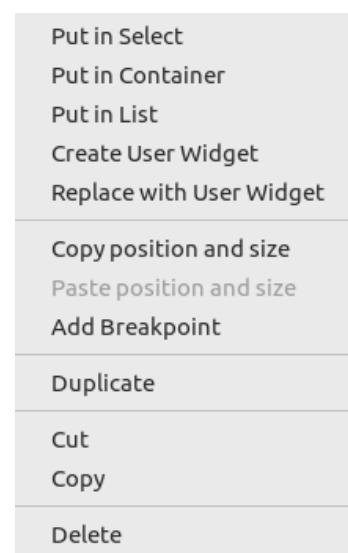


Fig. 42: Right-click menu (Widgets)

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

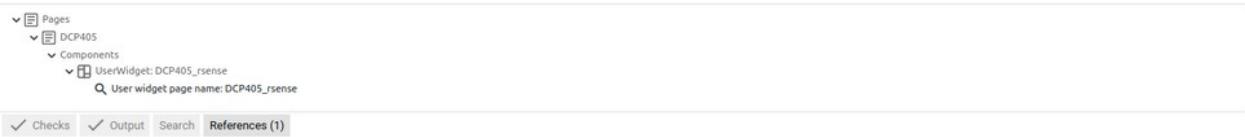


Fig. 43: Displaying the results of the Find all references operation

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

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 <i>Structs</i> and <i>Enums</i> types.
Properties	Display and edit properties of the selected item.
Breakpoints	List of all breakpoints where it is possible to enable / disable individual breakpoints.
Components Palette	List of all Widgets and Actions that can be added to a page or Action. The project type determines which Widgets and Actions will appear in the palette.
Styles	All styles for GUI elements.
Themes	Themes are used to easily switch styles and thus change the GUI appearance. When creating a new theme, all styles that currently exist will be added to the new theme.
Bitmaps	List of all imported bitmaps. It will be displayed if the <i>Bitmaps</i> feature is enabled in the project general settings. Bitmaps cannot be edited in the project editor.
Fonts	List of all imported fonts. It will be displayed if the <i>Fonts</i> feature is disabled in the project general settings. The project editor enables basic editing of fonts.
Texts	Localizing texts for multilingual GUI. It will be displayed if the <i>Texts</i> feature is disabled in the project general settings. The localization of the texts is described in Chapter P12.
IEXT (EEZ-GUI only)	Definition of IEXT extension. It will be displayed if the <i>IEXT defs</i> feature is disabled in the project general settings. One project can define multiple IEXT extensions. The IEXT creation procedure is described in Chapter XX.

<i>SCPI (EEZ-GUI only)</i>	List of SCPI commands that will be accessible in IEXT. It will be displayed if the <i>SCPI</i> feature is disabled in the project general settings.
<i>Shortcuts (EEZ-GUI only)</i>	List of Shortcuts that will be accessible in IEXT. It will be displayed if the <i>Shortcuts</i> feature is disabled in the project general settings.
<i>Changes</i>	List of all commits if the project is in a git repository. It will be accessible if the <i>Changes</i> feature is disabled in the project general settings.
<i>Checks</i>	The project editor is constantly looking for errors in the project (e.g. wrong expressions) in the background, and the errors found will be listed in this panel.
<i>Output</i>	It shows the report after the build is complete and the errors found.
<i>Search</i>	Content search and replace. See Section P4.3.1
<i>References</i>	The following items can be found where they are all used in the project: Variables, Struct, Enum, Page, Action, Style, Font, Bitmap. Right click on the object and "Find all references", the found references will be displayed in this panel (see Fig. 43).

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.

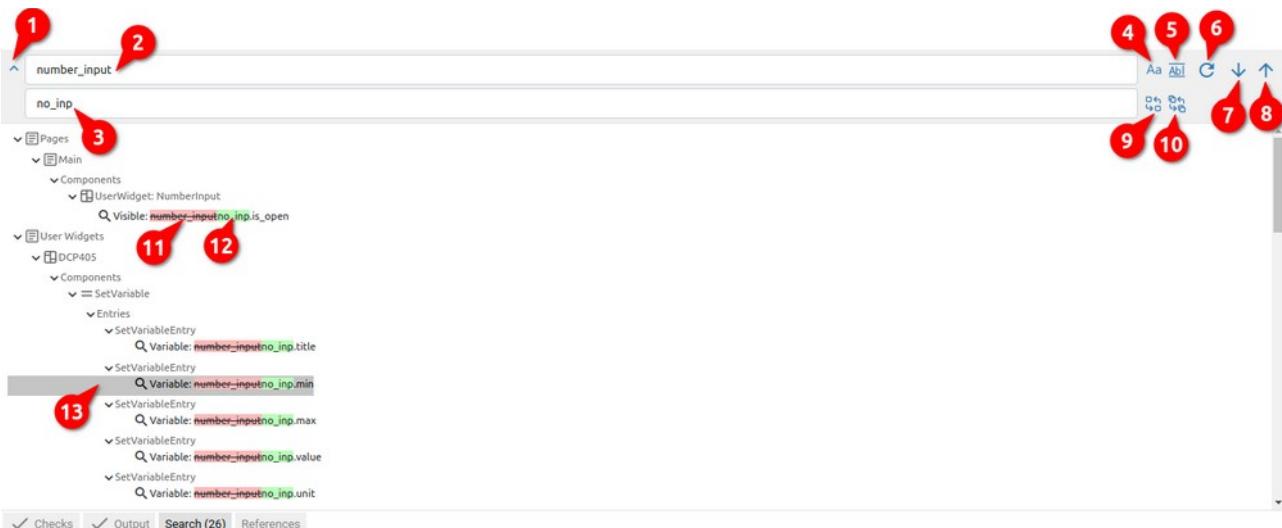


Fig. 44: Search and Replace panel

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

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

P4.4. Debug mode panels overview

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

P5. Project editors/viewers

P5.1. Editors

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

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. 46).

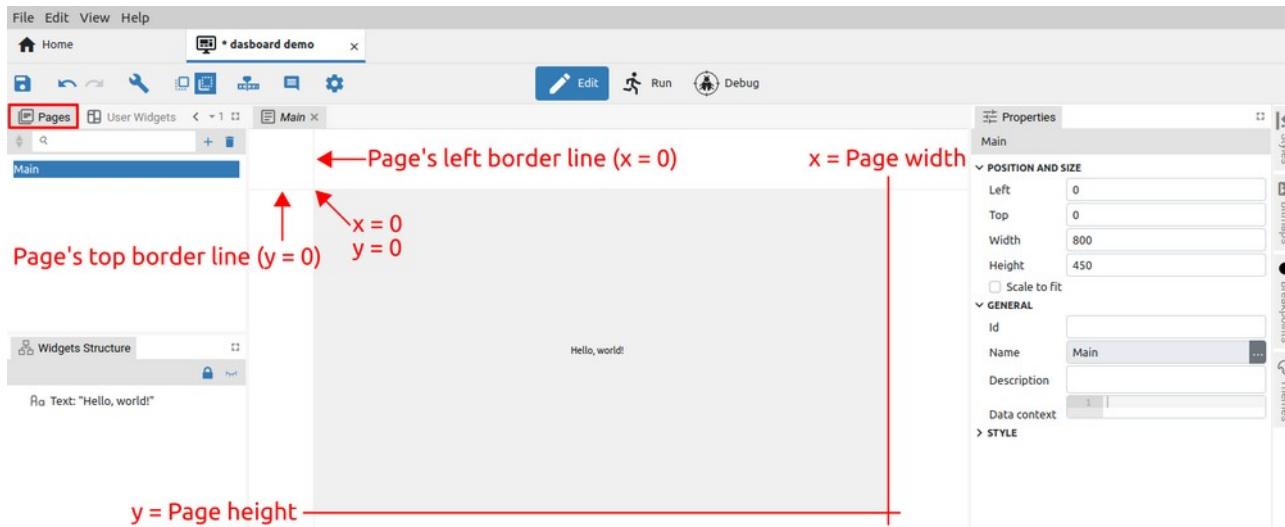


Fig. 45: Display of the page in the editor

P5.1.2. User Actions

The User Actions editor allows editing the selected Action from the User Actions panel (Fig. 46).

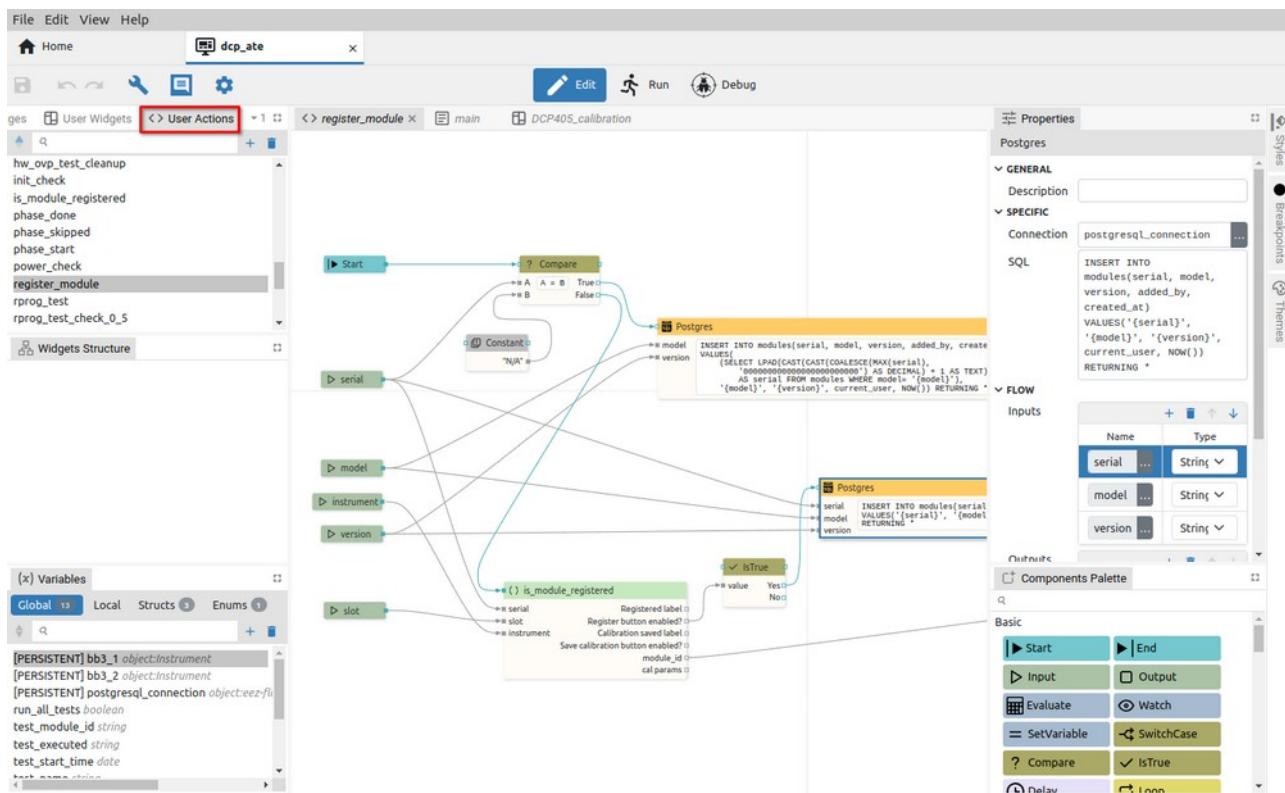


Fig. 46: User Actions editor page

P5.1.3. User Widgets

The User Widgets editor allows editing the selected Widget from the User Widgets panel (Fig. 47).

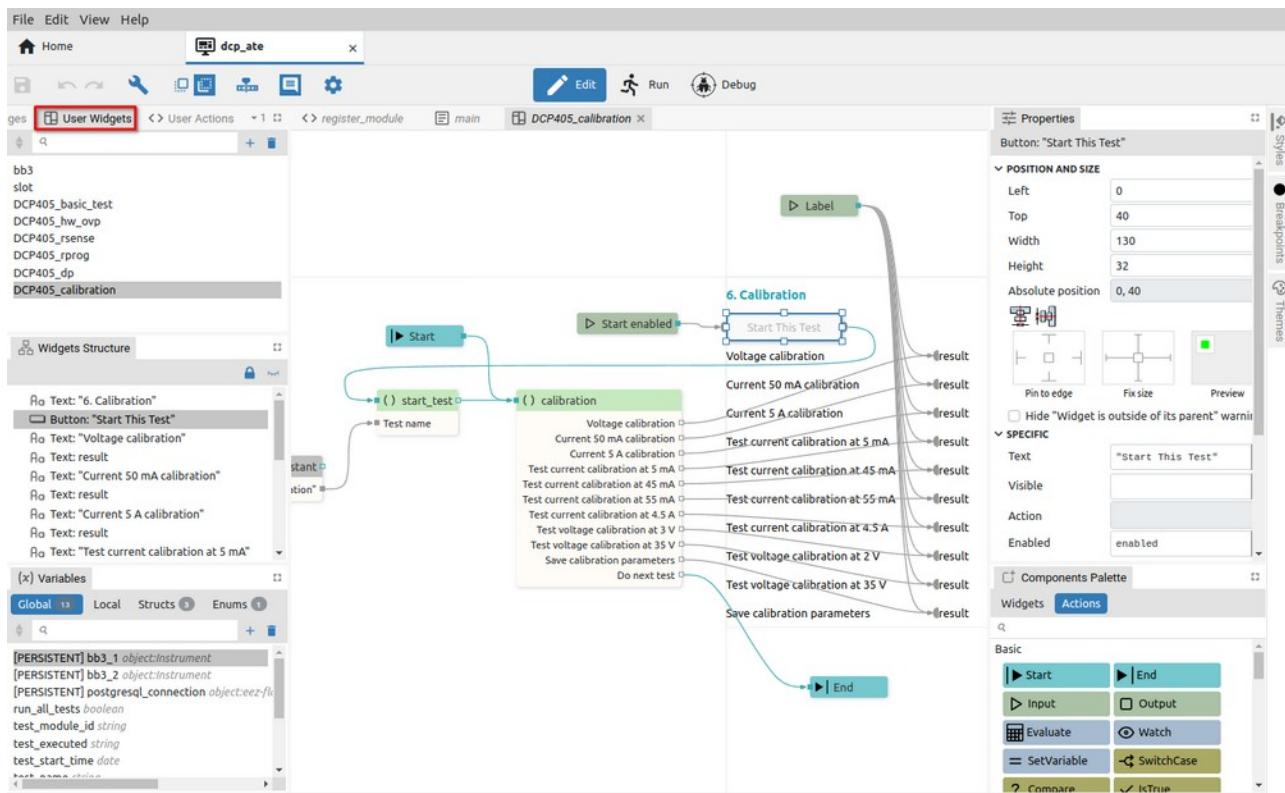


Fig. 47: User Widgets editor page

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.

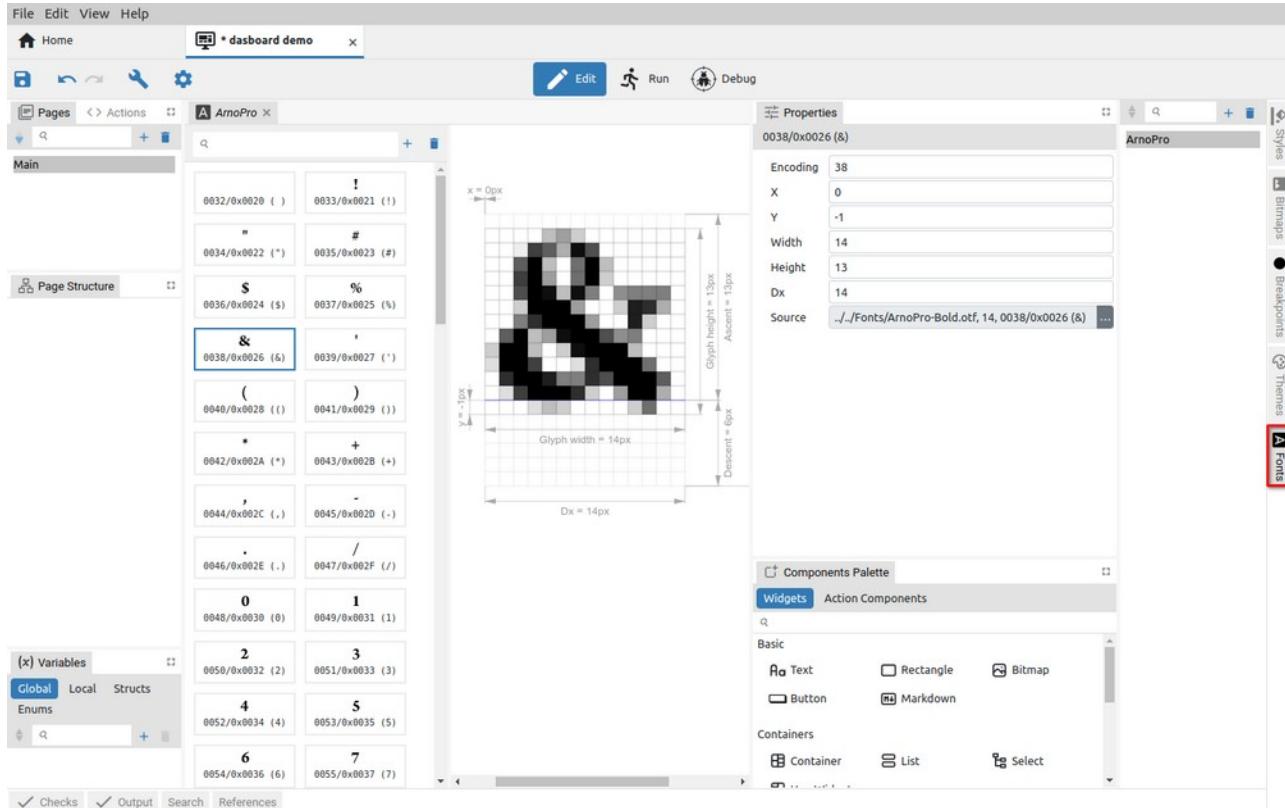


Fig. 48: Font editor page

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. 49).

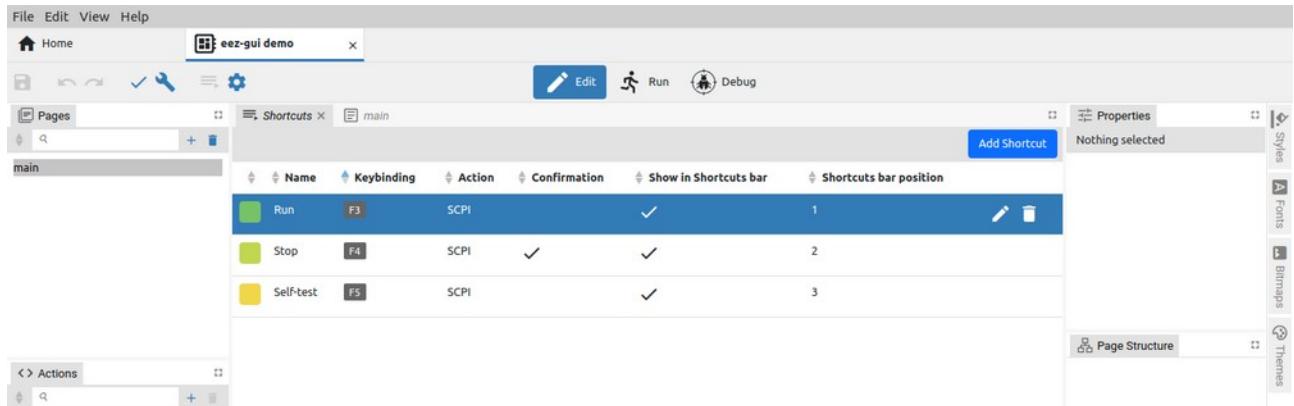


Fig. 49: Project shortcuts page

P5.1.6. MicroPython (EEZ BB3 only)

Opens the MicroPython text editor page (Fig. 50).

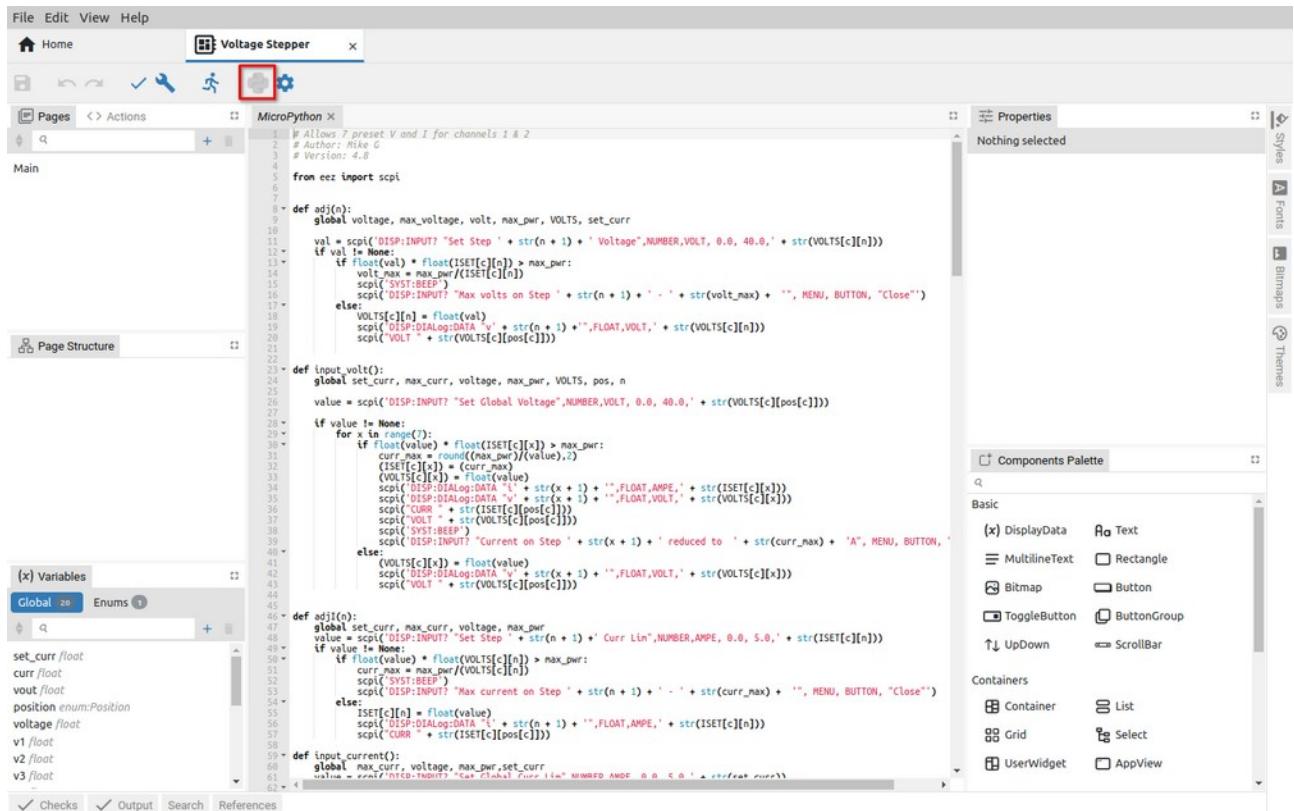


Fig. 50: MicroPython editor page

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.

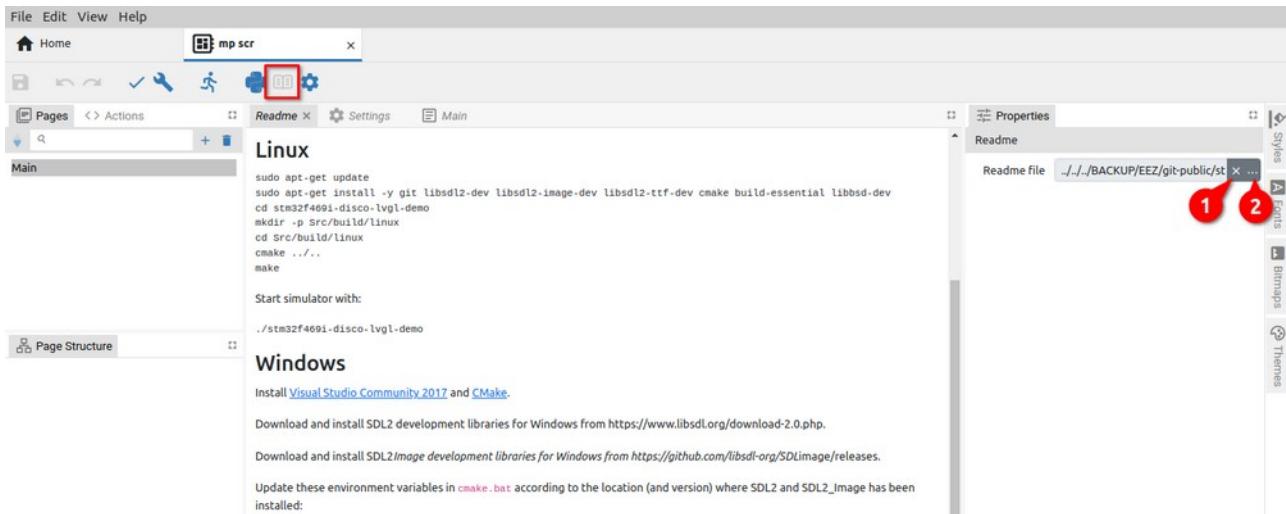


Fig. 51: Project Readme file

P5.1.8. Settings

The Settings page is used to edit the global parameters and features of the project (Fig. 52).

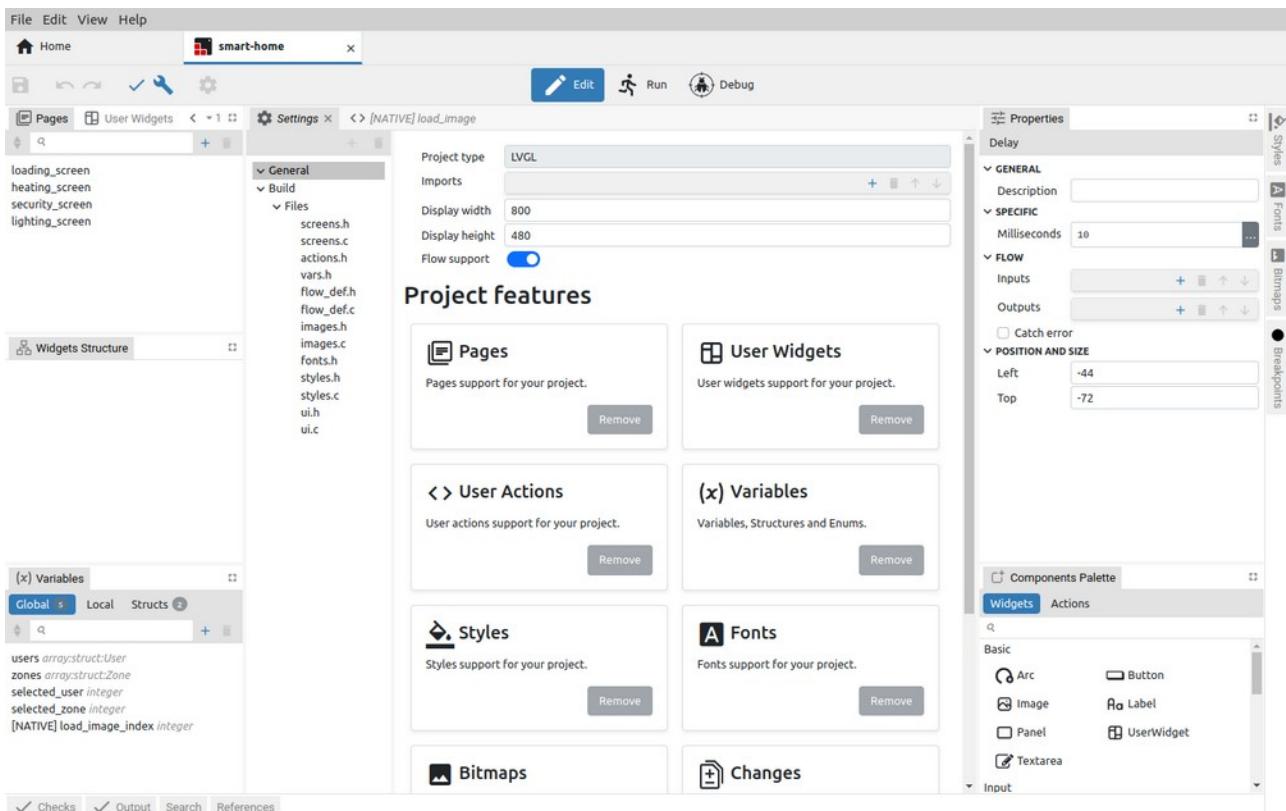


Fig. 52: Project settings page

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. 53). In *Debug* mode, pages and Actions cannot be edited, so editors effectively become viewers (Fig. 54).

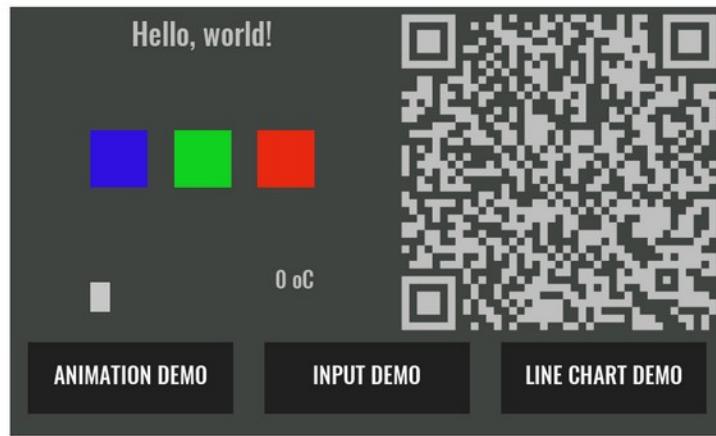


Fig. 53: Page view in Run mode

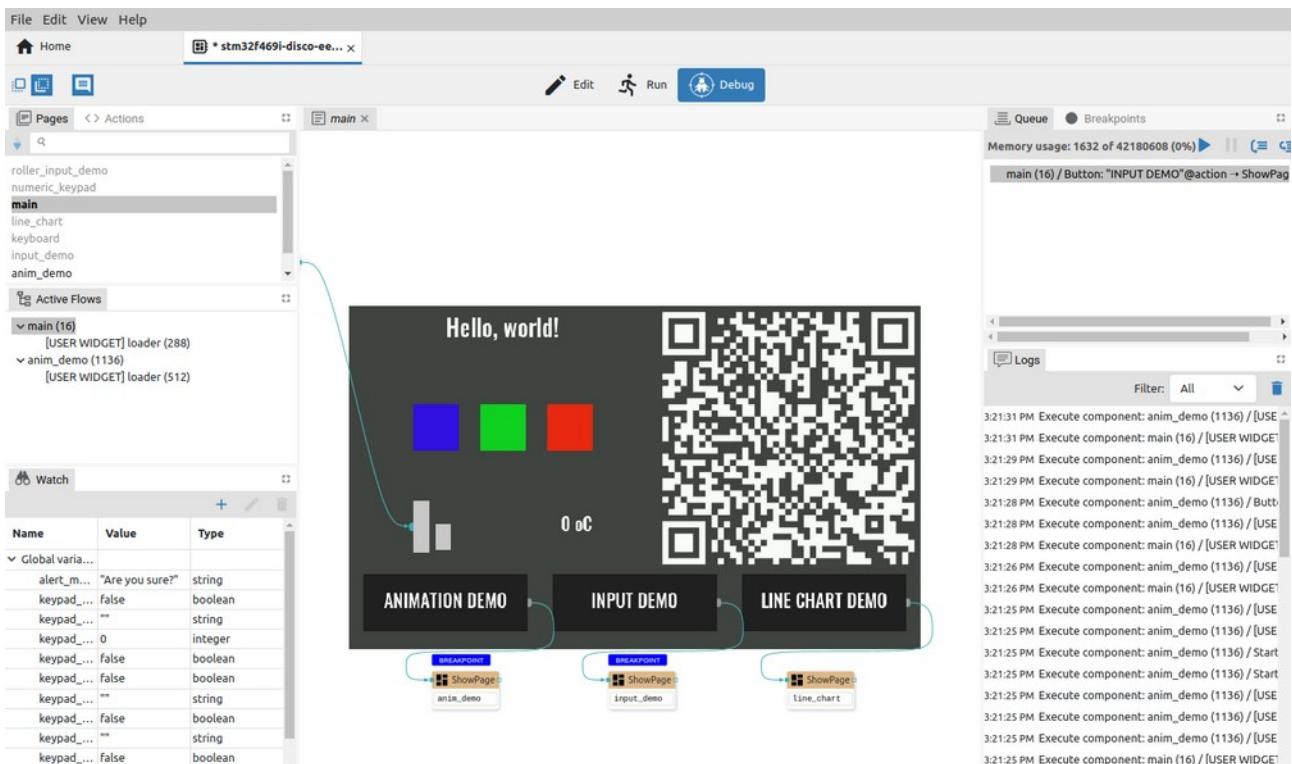


Fig. 54: Page view in Debug mode

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 breakpoints and see what values the component has on the inputs.

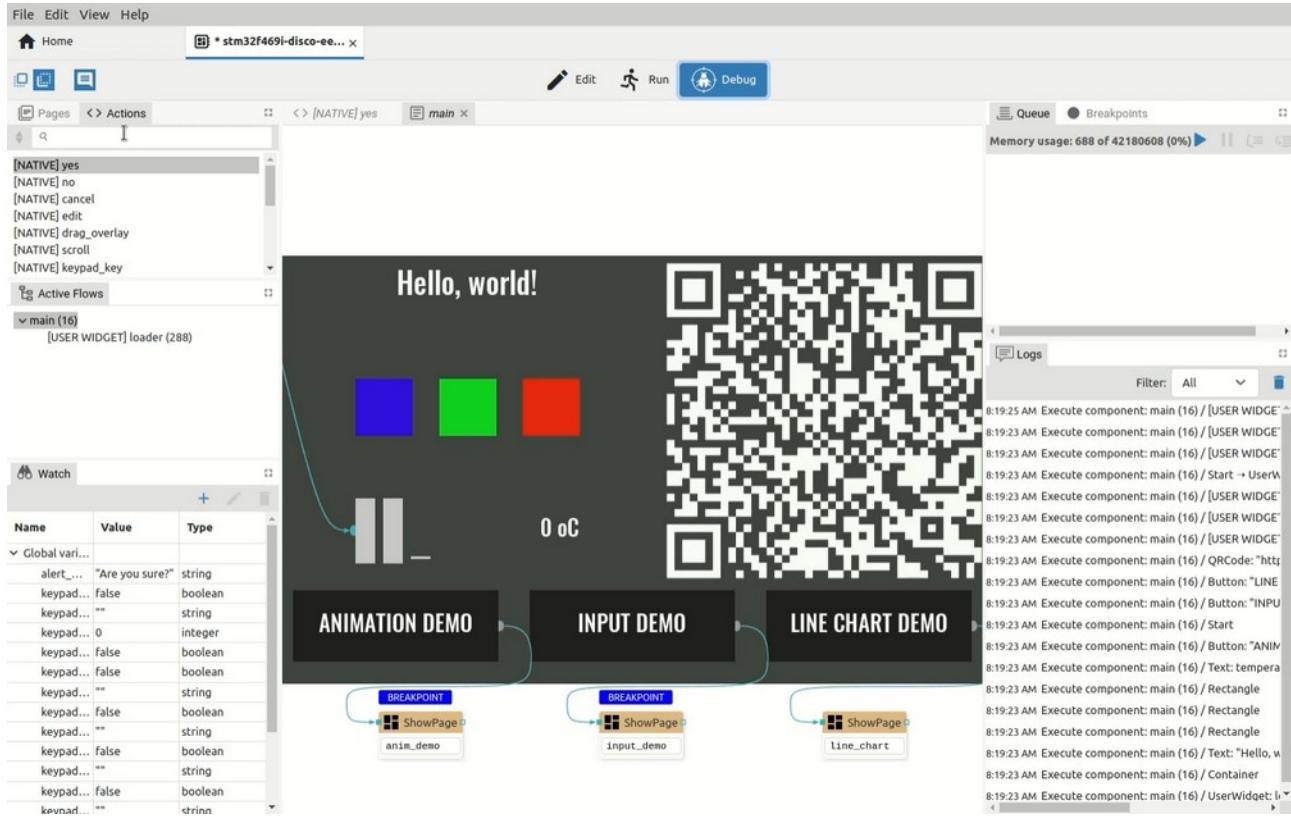


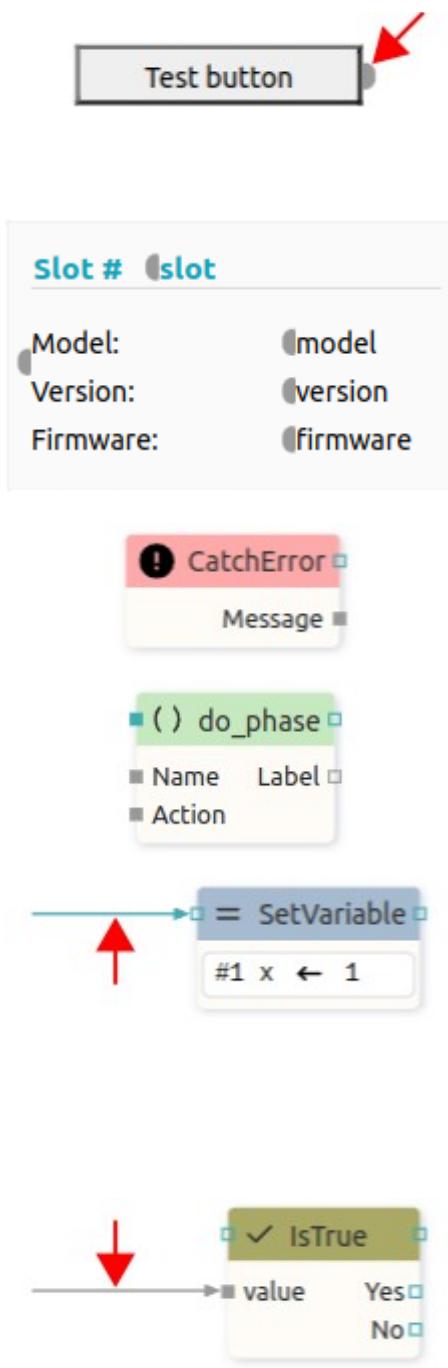
Fig. 55: Action viewer

P6. EEZ Flow

P6.1. EEZ Flow basic concepts

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

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



Widget

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

User Widget

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

Action

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

User Action

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

Sequence Flow line

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

Data Flow line

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

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. 56).

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.

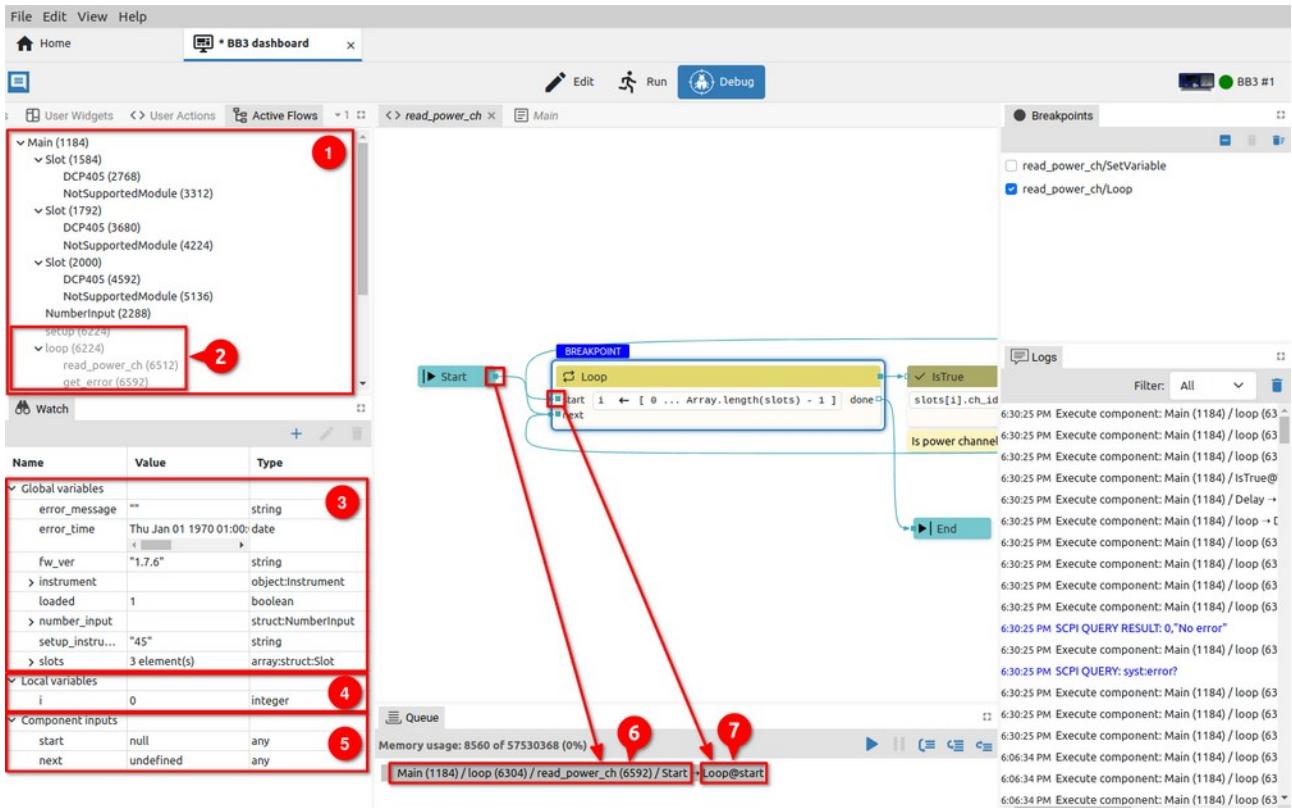


Fig. 56: Flow execution monitoring

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. 56 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.

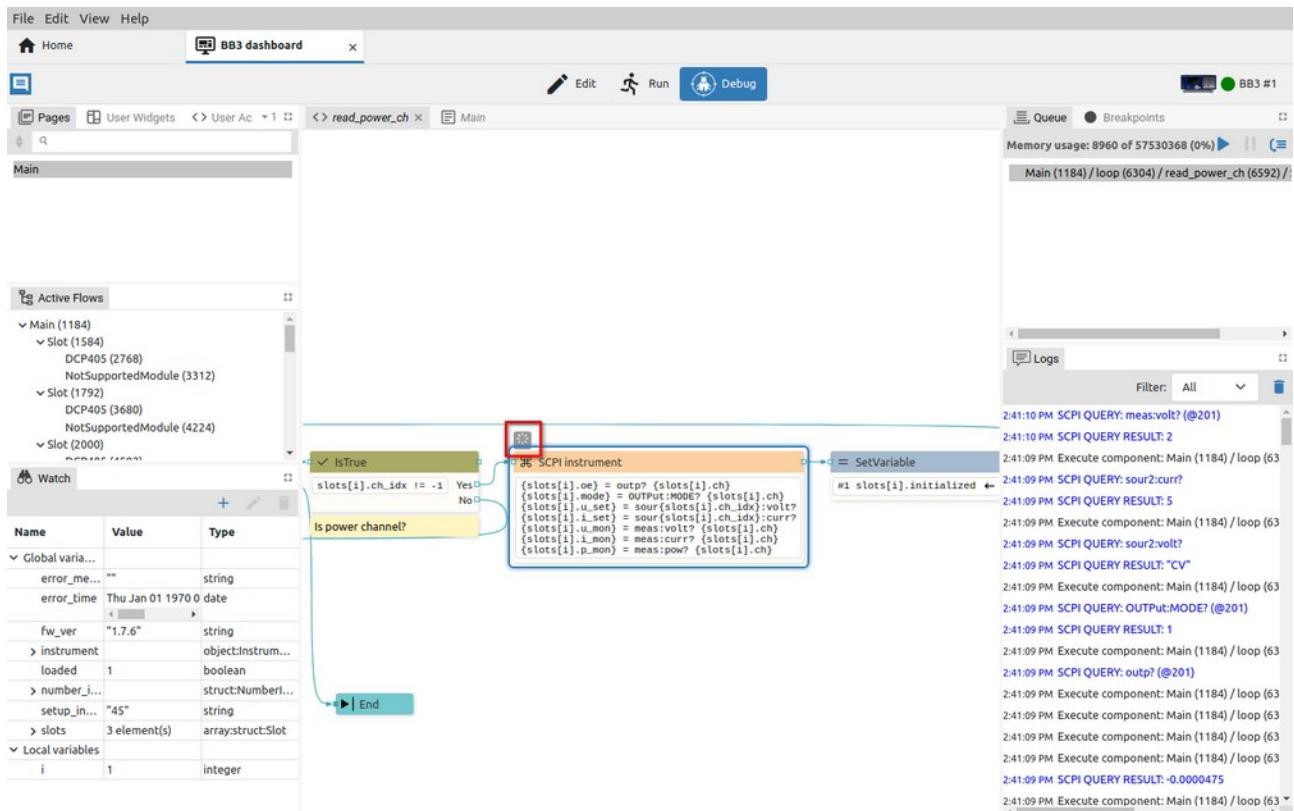


Fig. 57: Indication that the component preserves internal states

Components that preserve internal state, i.e. whose execution takes a long time, are also marked with a special icon in the debugger (Fig. 57). 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. 58 shows four simple Flows that contain User Actions whose inputs are triggered in different ways. Fig. 59 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).

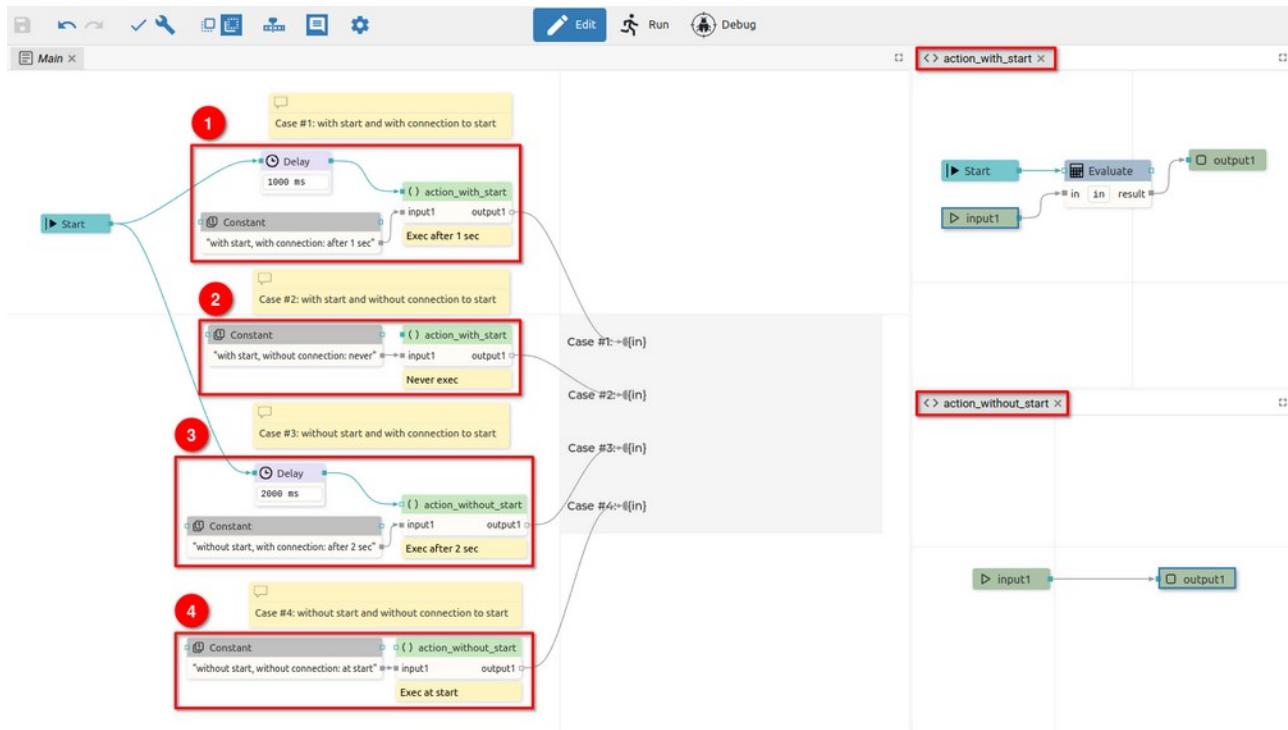


Fig. 58: Flow execution examples

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. 59: 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. 60)

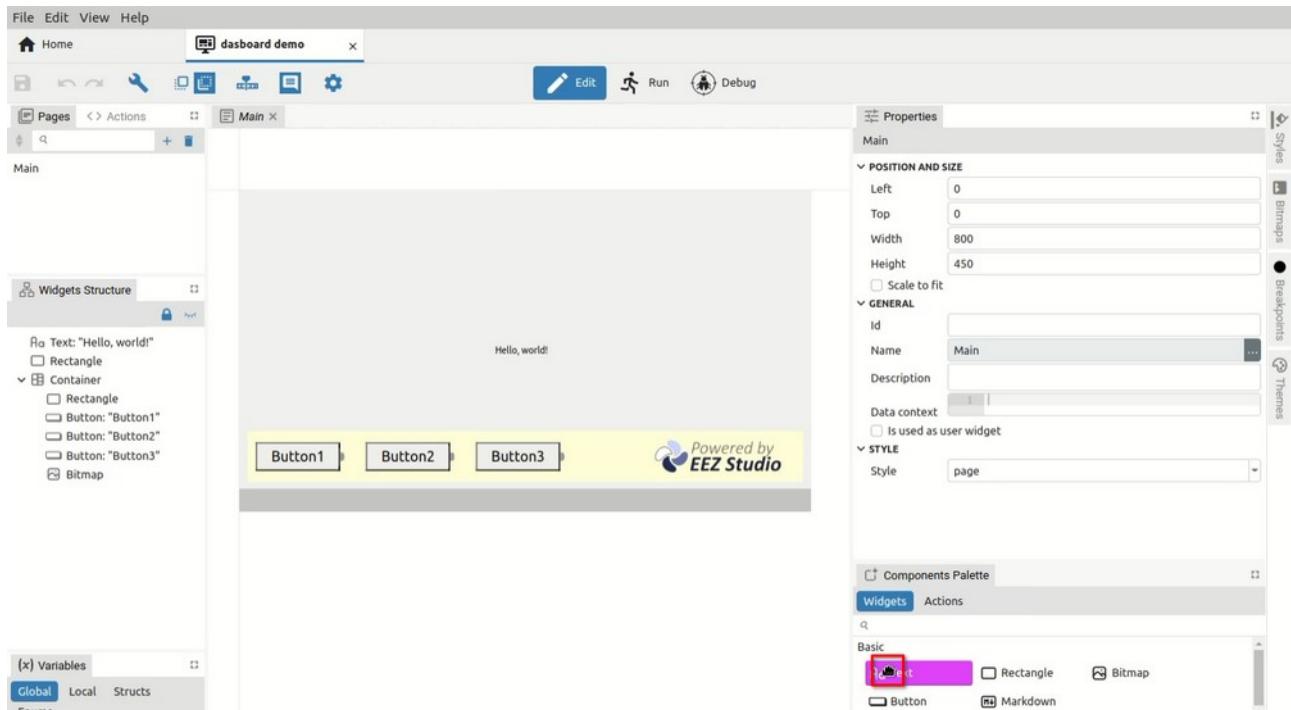


Fig. 60: Selection of Widget to add to the page

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

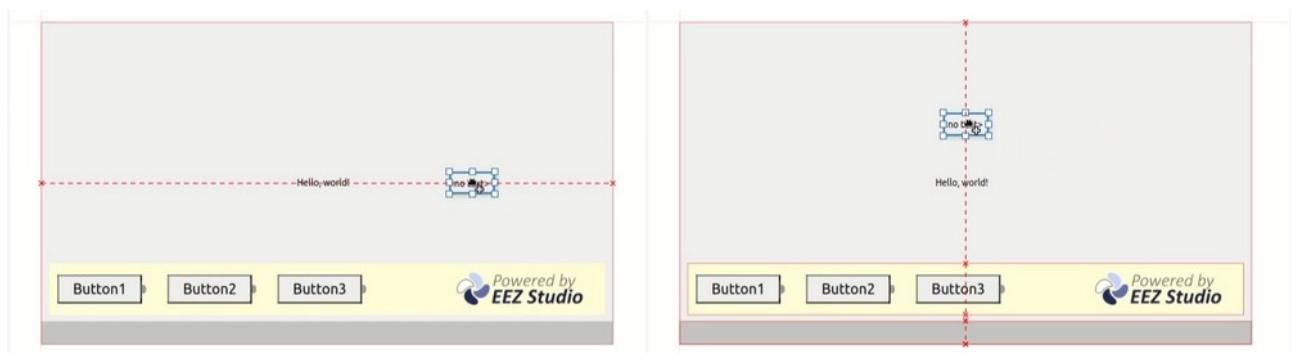


Fig. 61: Snap lines for centering in the page



Fig. 62: Snap lines for positioning versus other Widgets

Fig. 62 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.

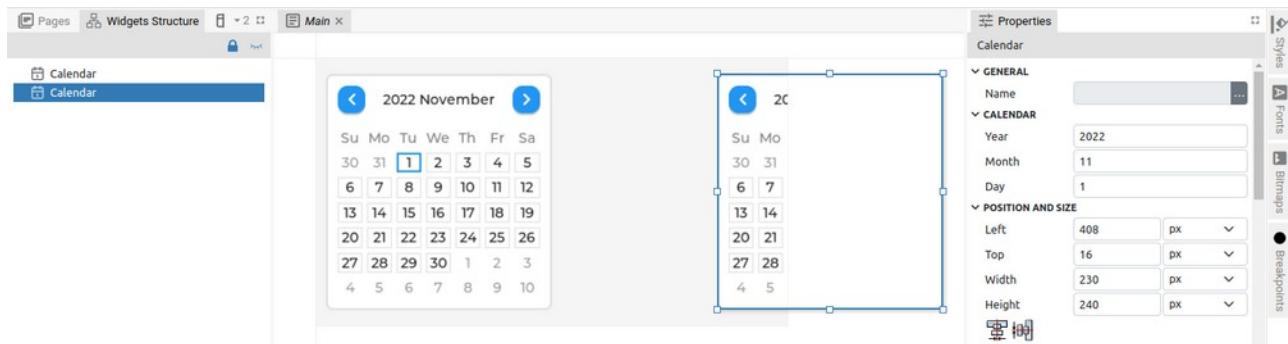


Fig. 63: Widget positioned outside the page

Please note that if the Widget in the *EEZ-GUI* and *LVGL* projects is set to protrude from the page (Fig. 63), 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. 64): 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. 65 when selecting the area that will include the Widgets we want to select.



Fig. 64: 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. 65: "Rubber band" selection

As shown in Fig. 64, 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. 66.

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

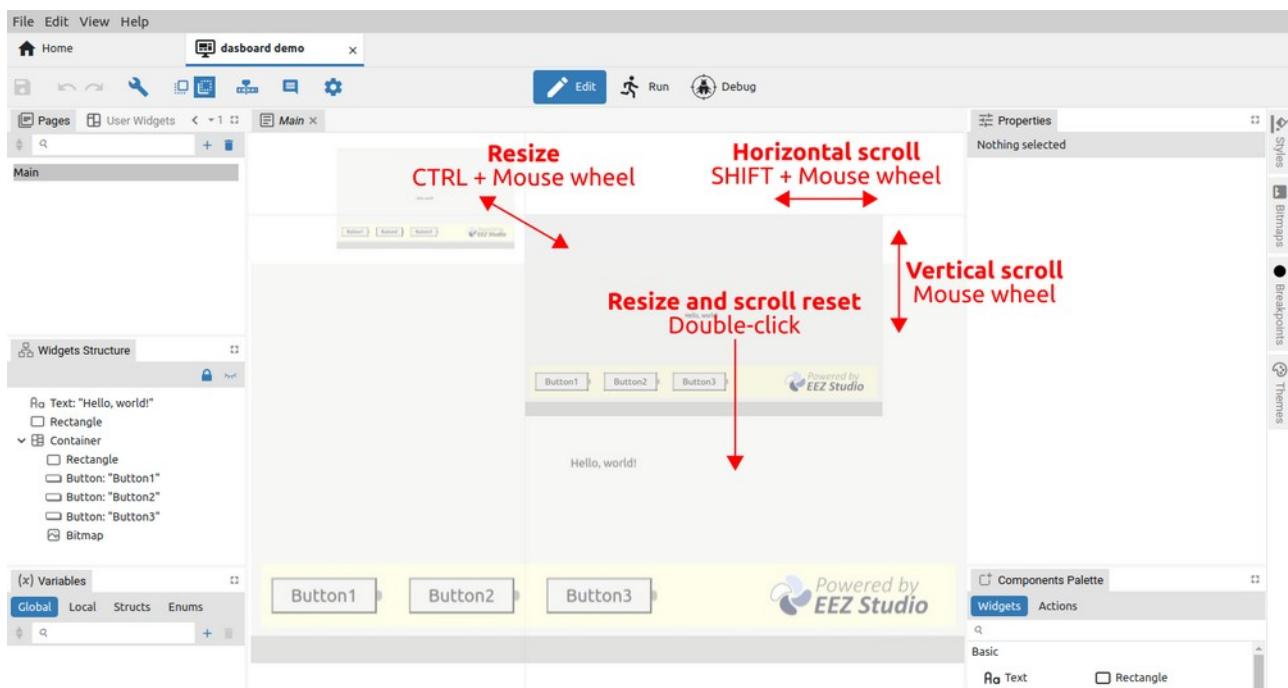


Fig. 66: Page resize and scroll

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. 61 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).

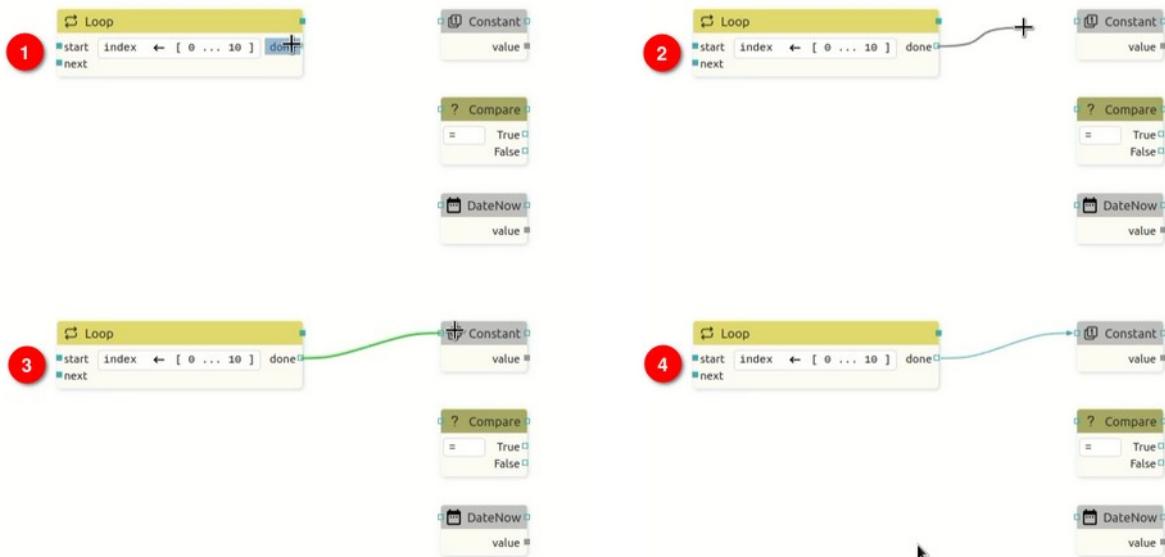


Fig. 67: Connecting the output to the input of the Widget

Adding a Flow line is also possible by starting from the input of one component to the output of another. In Fig. 68 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.

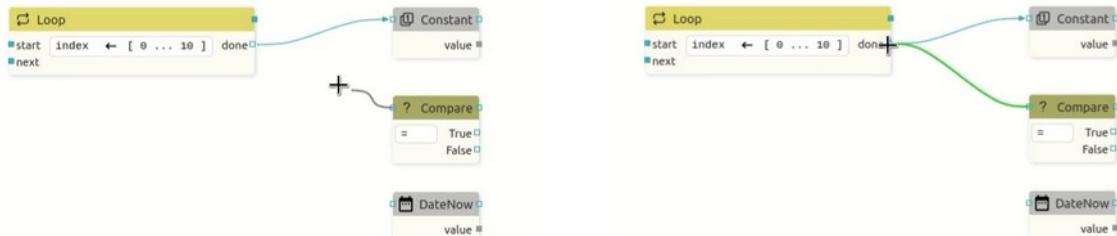


Fig. 68: Connecting the input to the output of the Widget

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. 69 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.

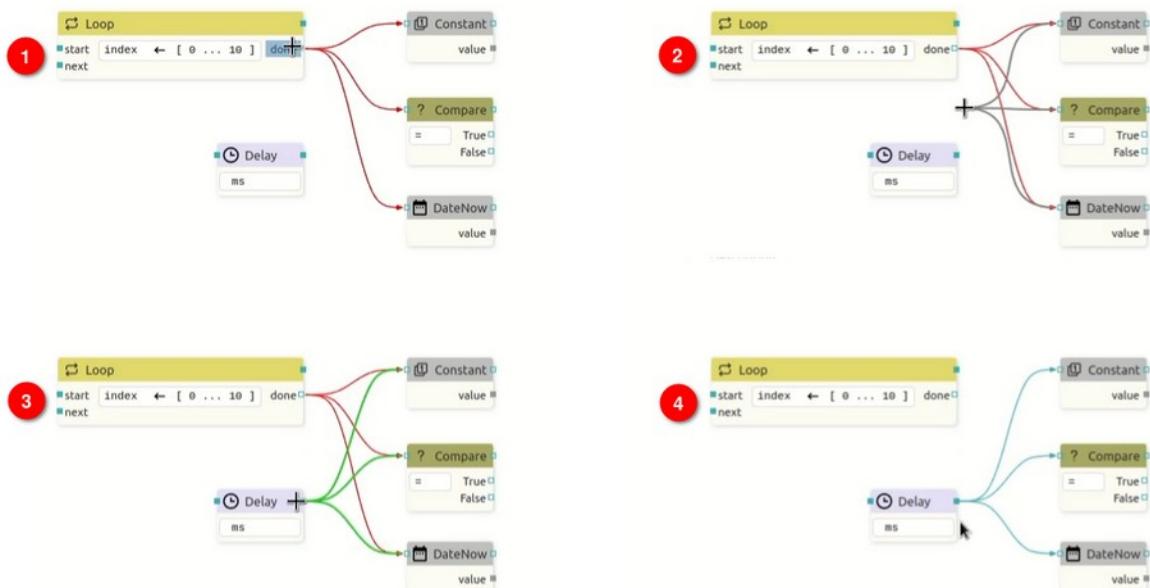


Fig. 69: Moving multiple connections

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. 70)
- *LVGL* – Widgets from the open-source embedded graphics library LVGL. They can only be used in a project of type LVGL. (Fig. 71)

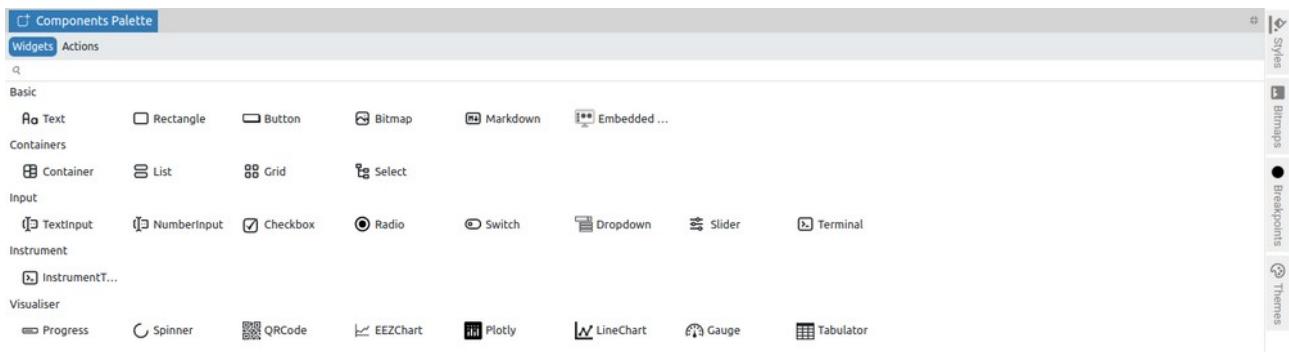


Fig. 70: Widgets palette for the EEZ-GUI project

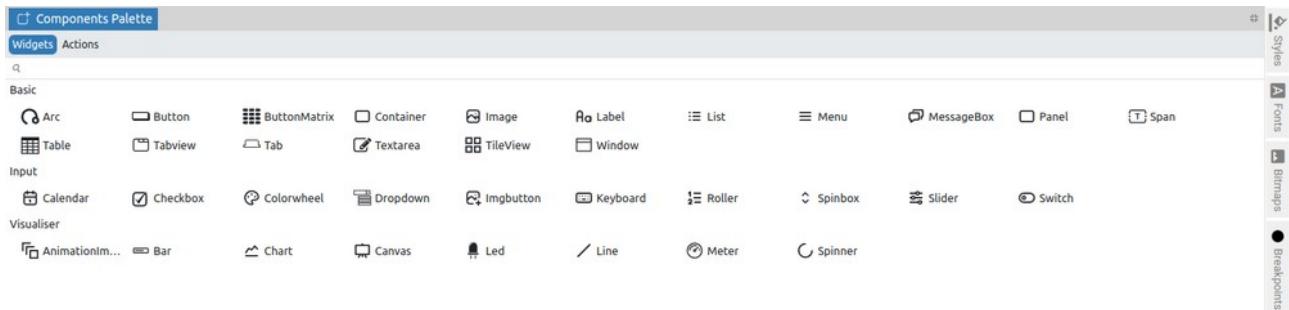


Fig. 71: Widgets palette for the LVGL project

P7.2.1. Widget component's items

Widget component items are shown in Fig. 72 and described below.

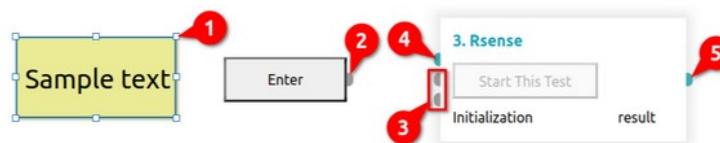


Fig. 72: Widget components items

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

rectly.

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.

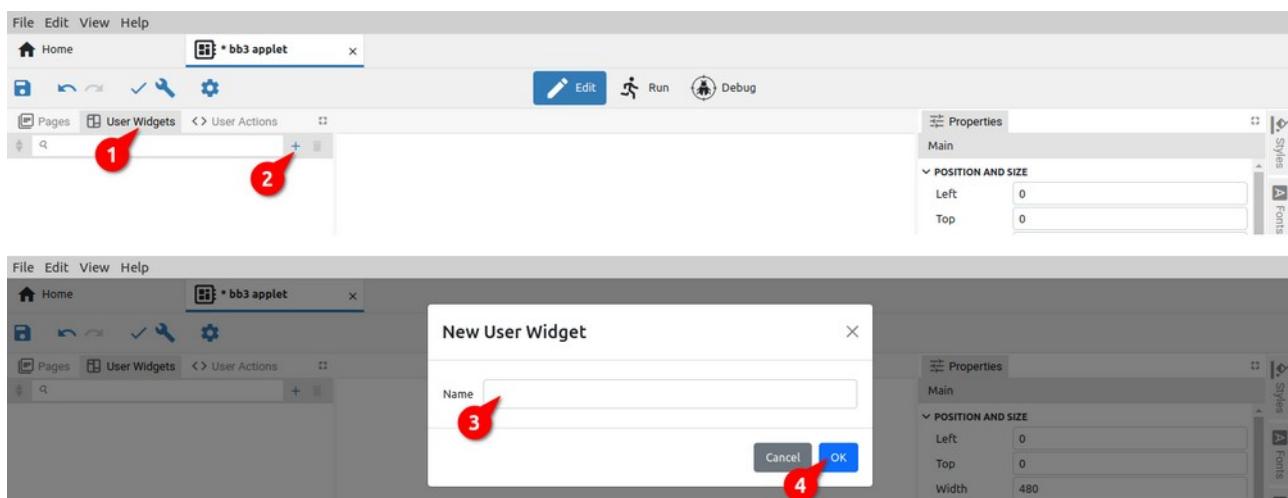


Fig. 73: Creating a new User Widget from the User Widgets panel

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. 73).

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 ($x = 0$, $y = 0$).

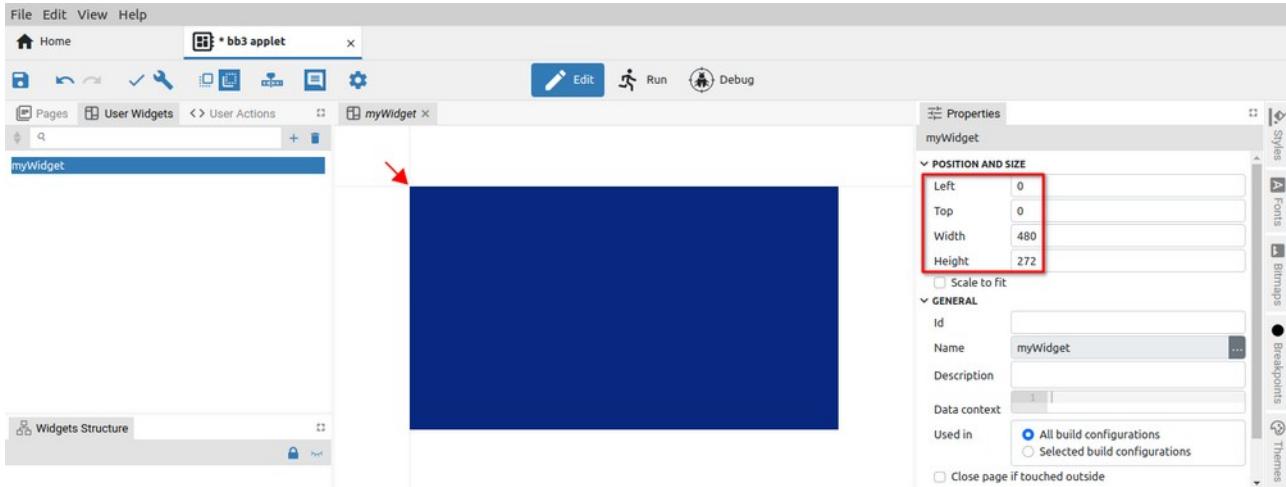


Fig. 74: Page editor of the newly created User Widget

User Widget can also be created by selecting one or more components on the currently displayed page (1) as shown on Fig. 75. By selecting the right-click menu option *Create User Widget*, a dialog box will appear as in the previous case (Fig. 73).

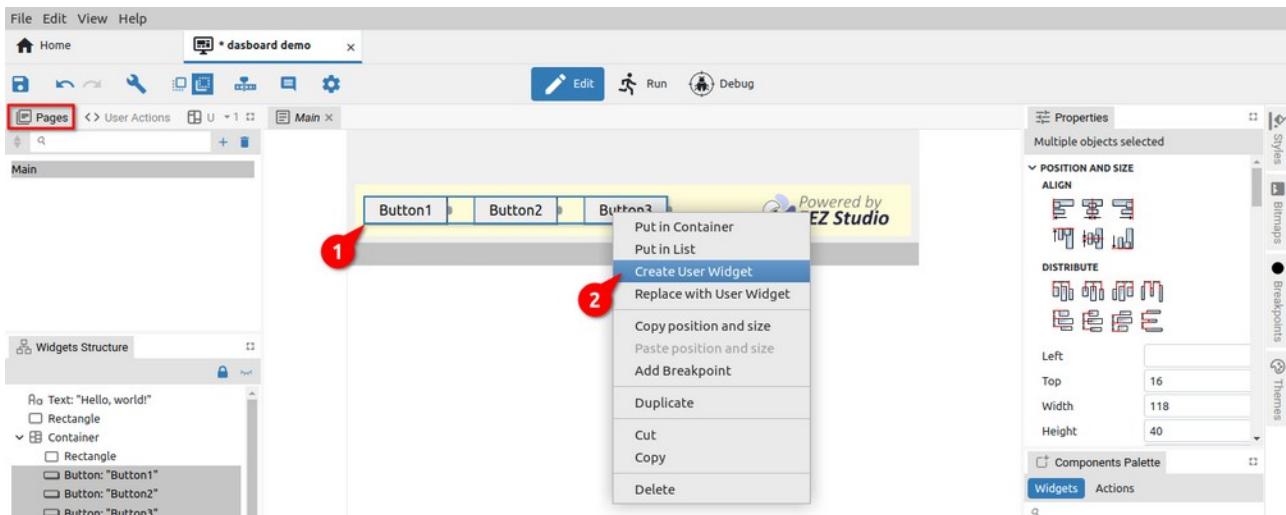


Fig. 75: Creating a new User Widget using the right-menu option

After successfully adding a new User Widget, it can be edited in the page editor (Fig. 76). 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.

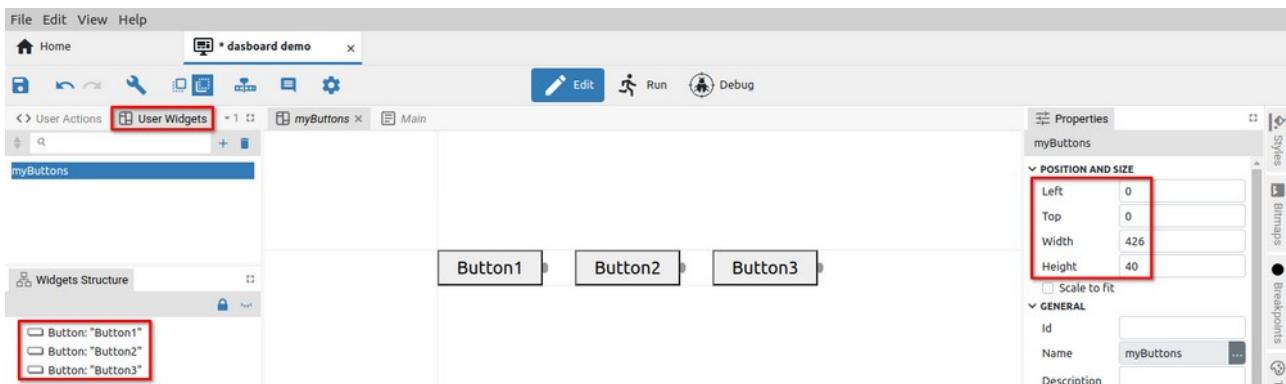


Fig. 76: The newly created User Widget in the editor

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. 77 shows the Actions for the Dashboard project, and Fig. 78 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. 77).

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. 78), from where they can be added to the project with drag and drop as any Action or Widget (1).

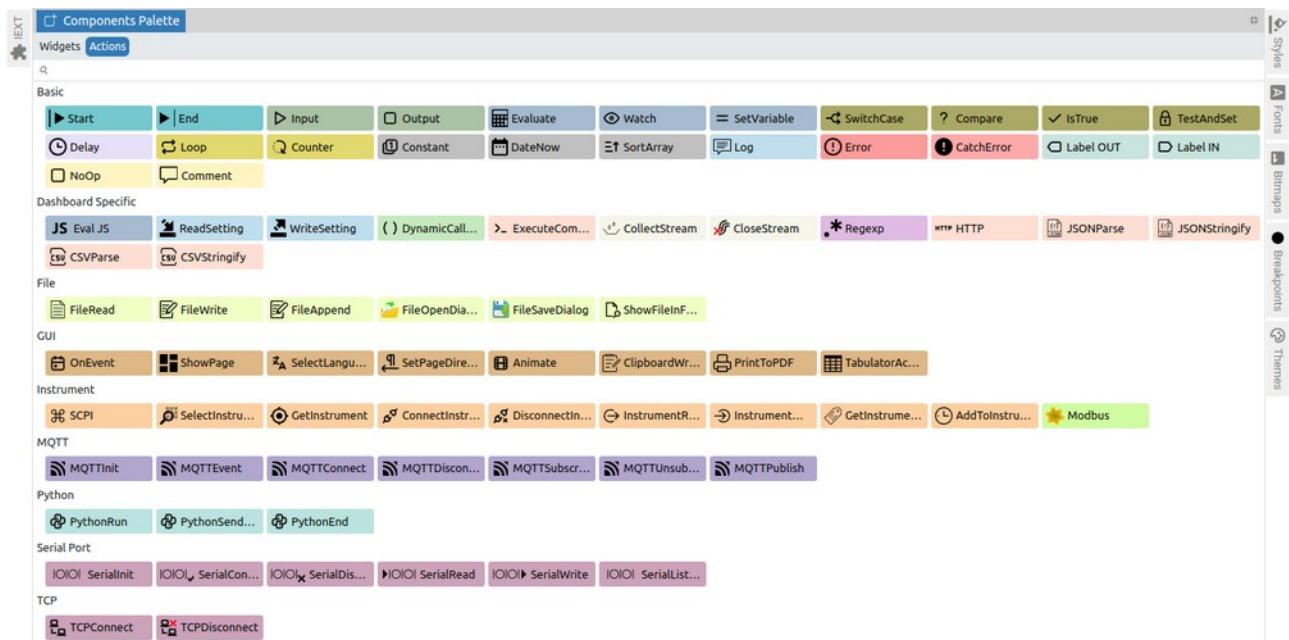


Fig. 77: Actions palette for the Dashboard project

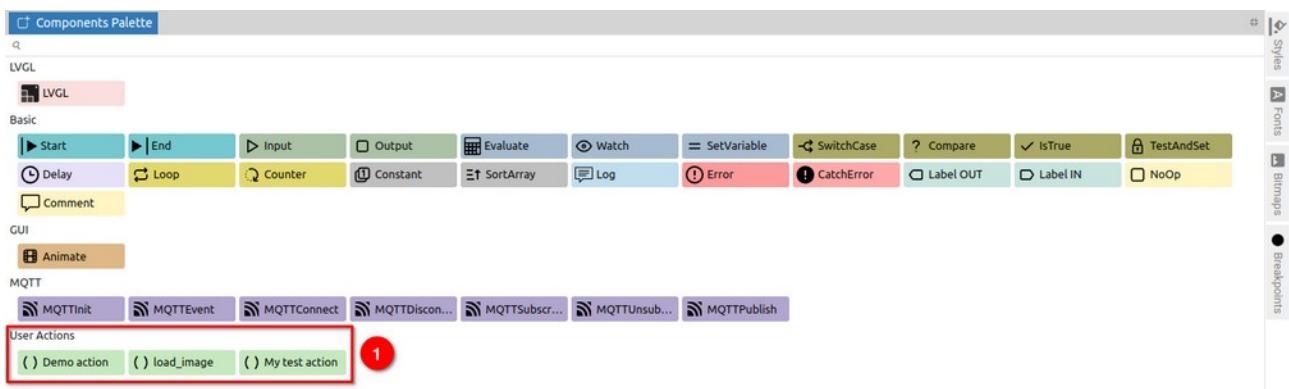


Fig. 78: Actions palette for the LVGL project

P7.3.1. Action component's items

Action component items are shown in Fig. 79 and described below.

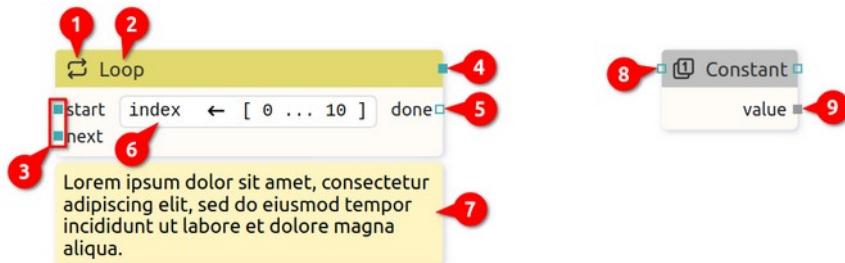


Fig. 79: Action components items

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

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

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

Table 3: Action's pin types

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.

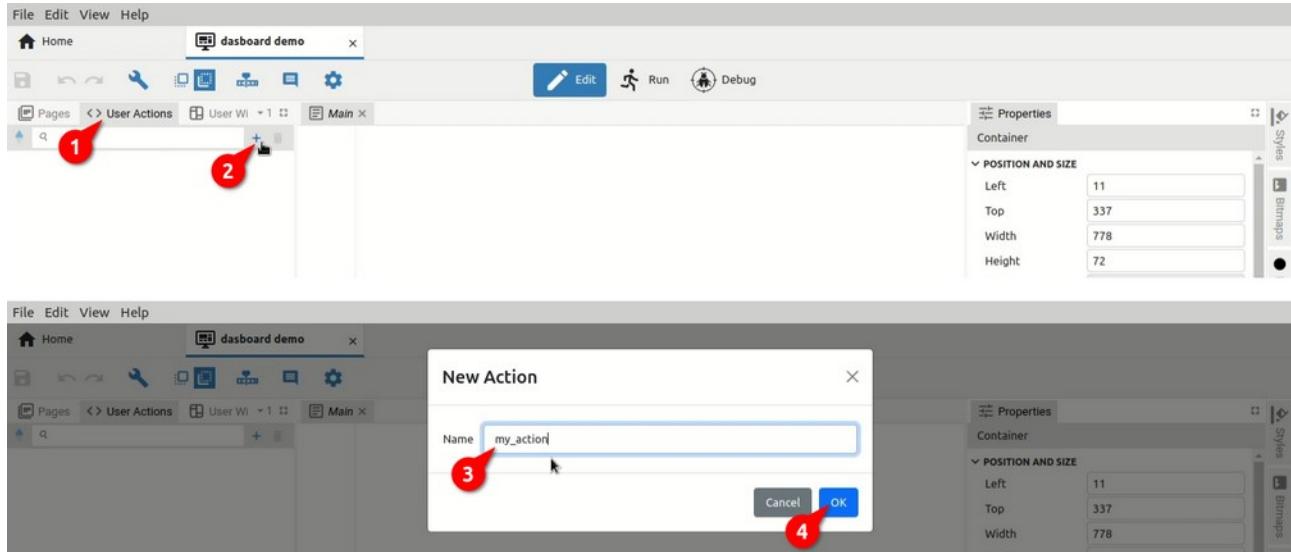


Fig. 80: Creating a new User Action

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.

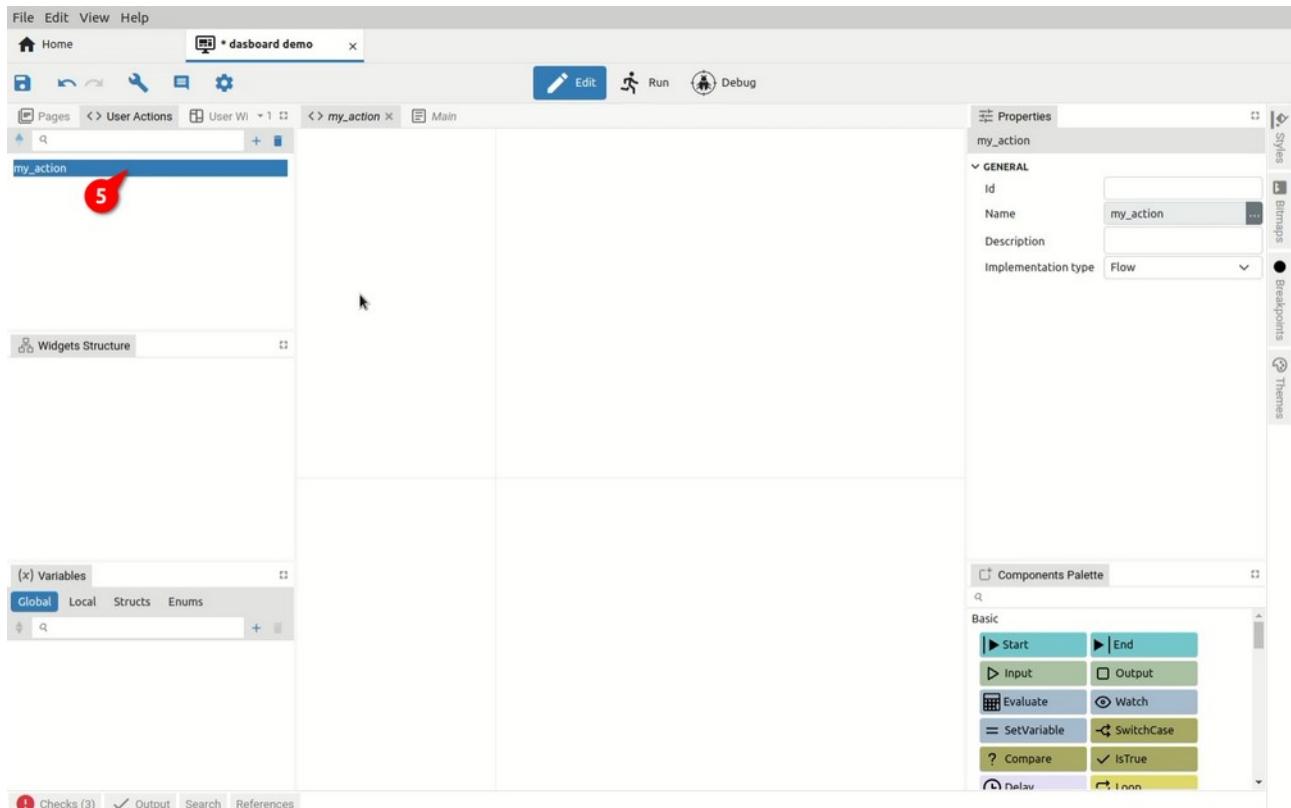


Fig. 81: Flow editor of the newly created User Action

Fig. 82 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.

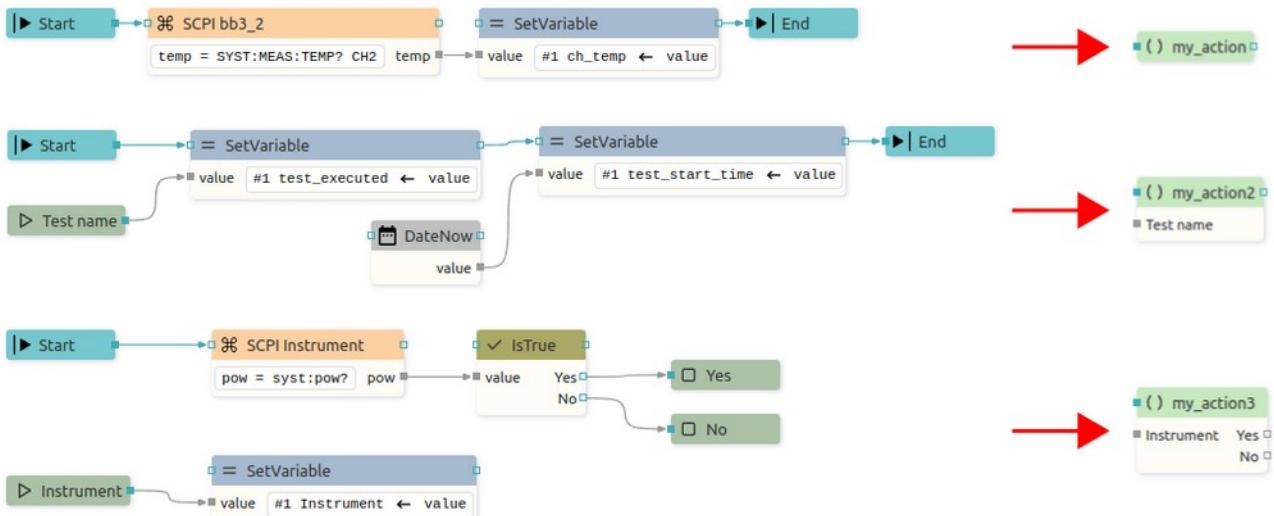


Fig. 82: User Action examples

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. 83), 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. 84, Fig. 85 and Fig. 86 shows *Properties* panels for different types of variables from different types of projects.

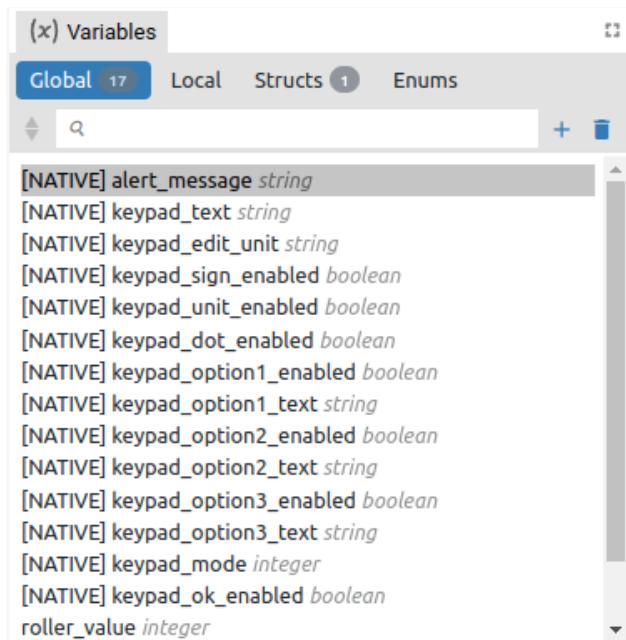


Fig. 83: Variables panel

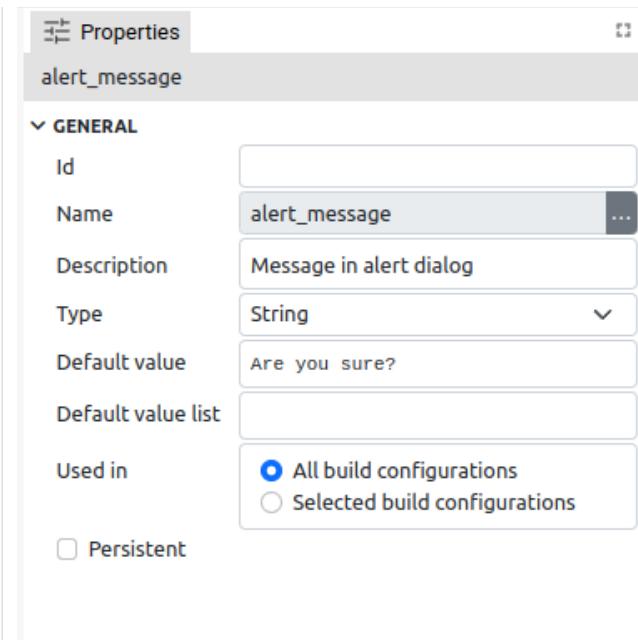


Fig. 84: Variable Properties panel (EEZ-GUI)

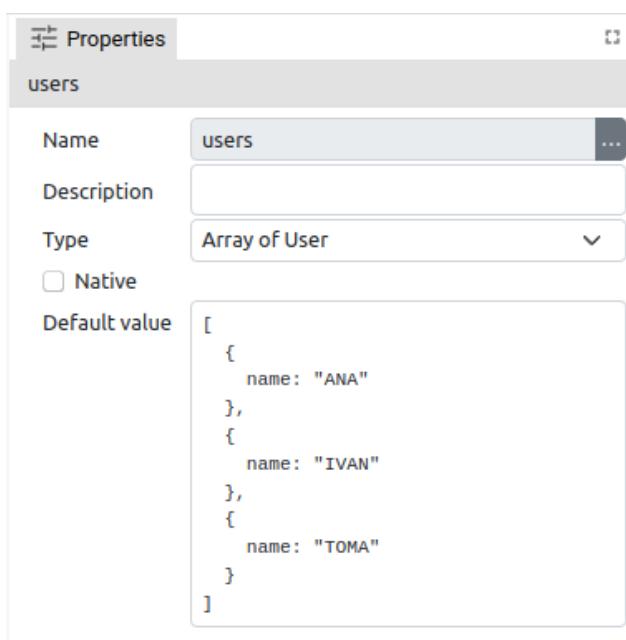


Fig. 85: Variable Properties panel (LVGL)

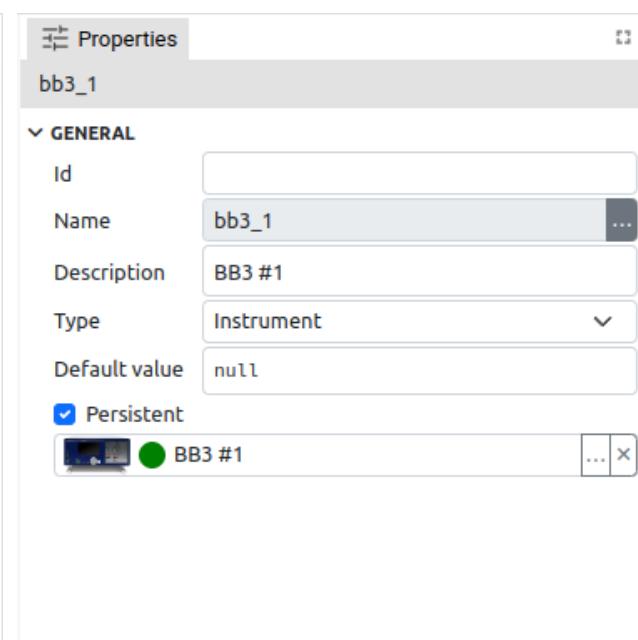


Fig. 86: Variable Properties panel (Dashboard)

Item	Description
<i>Id</i>	ID is <i>EEZ-GUI</i> project specific and is used in <i>Page</i> , <i>Action</i> , <i>Global Variable</i> , <i>Style</i> , <i>Font</i> , <i>Bitmap</i> and <i>Colors</i> . These are all resources that are referenced by name in the project editor. When that project is built, names are no longer used, but numerical IDs. This field is optional, i.e. an ID does not have to be specified, in which case an ID will be assigned during the build. However, if we want an object to always get the same ID, then it needs to be defined. Why would we want to always have the same ID? This is necessary when there is a master project such as <i>modular-psu-firmware.eez-project</i> from EEZ BB3 and that master project is used by BB3 Applets and BB3 MicroPython scripts and they can use resources from the master project that have that ID defined.
	<i>Important:</i> once the ID is set, it should not be changed, otherwise all BB3 scripts that depend on it should be rebuilt.
<i>Name</i>	Variable is referenced in other parts of project by its name. Rules for naming variables: Starts with a letter or an underscore (_), followed by zero or more letters, digits, or underscores. Spaces are not allowed.
<i>Description</i>	Optional field, contains a description of the variable.
<i>Type</i>	The type of data stored in variable. When adding a new variable, its suggested default value will depend on the selected type (0 for <i>Integer</i> , False for <i>Boolean</i> , etc.).
<i>Native</i>	The variable is managed by the native code (written in C++). A Dashboard project cannot have Native variables, and working with them is explained in Chapter XX.
<i>Default value</i>	Default value is the initial value of the variable when Flow starts. Given in JSON notation (https://www.json.org/json-en.html).
	For example: 123, "Hello", true If types is struct: { "member1": 42, "member2": "Hello" } If type is array: [1, 2, 3] or ["string1", "string2", "string3"]
<i>Default value list</i>	Only supported in <i>EEZ-GUI</i> project that does not have EEZ Flow enabled.
<i>Used in</i>	See <i>Configurations</i> (Chapter P13.2.1) in project <i>Settings</i> .
<i>Persistent (Global variables only)</i>	Stores the last value of the variable in the <i>.eez-runtime-settings</i> file, so that next time projects will have this value, instead of the default value.
	Supported only in Dashboard projects.

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

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. JSON objects

JSON is alternative to structures in Dashboard projects (for now, JSON is only supported in Dashboard projects!). Structure has fixed set of fields which must be specified during the development time. In the runtime you can't extend a structure value with a new filed name. Contrary to that, JSON values are completely open, you don't need to specify its structure during development and you can build arbitrary JSON values during the runtime. We can say that structures are statically typed and JSON is dynamically typed.

A JSON variable can be assigned to the struct or array variable and vice versa.

P8.2.7. Expressions

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

Expression element Description / Example

<i>Literal value</i>	Example: 42, "Hello", true
<i>Variable names</i>	Example: my_var
<i>Input names</i>	Retrieves the data stored in data input using the name of that input. Example: input_name
<i>Binary operator</i>	Example: my_integer_var + 1
<i>Logical operator</i>	Example: my_integer_var < 10
<i>Unary operator</i>	Example: -my_integer_var
<i>Ternary operator</i>	Example: my_integer_var == 1 ? true : false (evaluates to true if my_integer_var is 1, otherwise evaluates to false)
<i>Function calls</i>	Example: String.length(my_string_var)
<i>Parentheses "()"</i>	Specifying the order of the evaluation Example: "Counter: " + (a + 1)
<i>Accessor ". "</i>	Structure type member accessor by using ". " Example: my_struct_var.member1
<i>Accessor "[]"</i>	Array element accessor by using "[]" Example: my_array_var[3], my_array_var[index]
<i>Enum value</i>	Example: MyEnumTypeName.Member1

Expression examples:

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

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

P8.2.8. Literals

Type	Description / Example
<i>Integer</i>	42
<i>Float or double</i>	01.03.14
<i>String</i>	"Hello world!"
<i>Template literals (Template strings)</i>	Template literals are literals delimited with backtick (`) characters, allowing for multi-line strings and string interpolation with embedded expressions. For example: `Measured voltage is \${voltage_var} V` is the same as: "Measured voltage is " + voltage_var + " V" Inside \${...} can be complex expression, for example: `Progress: \${Math.round(factor, 2) * 100}%` Here is an example of multi-line strings:

```

`<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>Document</title>
    </head>
    <body>
        ${html_body}
    </body>
</html>`
```

Translated string T"text_resource_id" (prefix T is mandatory)

Boolean true or false

JSON json`{
 "a": 1,
 "b": 2,
 "c": {
 "arr": [1, 2, 3]
 }
}`.

See <https://www.json.org/json-en.html>

P8.2.9. 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
<i>integer</i>	<i>integer</i>	<i>float</i>	<i>double</i>	<i>integer</i>	<i>err</i>
<i>double</i>	<i>double</i>	<i>double</i>	<i>double</i>	<i>double</i>	<i>err</i>
<i>float</i>	<i>float</i>	<i>float</i>	<i>double</i>	<i>float</i>	<i>err</i>
<i>boolean</i>	<i>integer</i>	<i>float</i>	<i>double</i>	<i>integer</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>

Division /

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

Remainder %

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

Left shift <<

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

Right shift >>

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

Binary AND &

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

Binary OR |

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

Binary XOR ^

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

P8.2.10. Logical operators

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

Type Description

<code>==</code>	Equal to
<code>!=</code>	Not equal
<code><</code>	Greater than
<code>></code>	Less than
<code><=</code>	Less than or equal to
<code>>=</code>	Greater than or equal to
<code>&&</code>	And
<code> </code>	Or

P8.2.11. Unary operators**Type Description**

<code>-</code>	Negate the value
<code>~</code>	Binary invert
<code>!</code>	Logical invert

P8.2.12. 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
<code>index</code>	<code>Integer</code>	In case of nested List/Grid widgets use 0 for inner most List/Grid, 1 for List/Grid one up, etc.

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
<code>structName</code>	<code>String</code>	Structure name.
<code>value</code>	<code>JSON</code>	Structure name.

Return value

Created struct value. Return type is `Struct`.

Flow.makeArrayValue

Creates a new value of type array.

Parameters

Name	Type	Description
<code>value</code>	<code>JSON</code>	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
<code>textResourceID</code>	<code>String</code>	Text resource ID.

Return value

Translated string. Return type is `String`.

Flow.parseInteger

Parse integer value given as string.

Parameters

Name	Type	Description
<code>str</code>	<code>String</code>	Input string.

Return value

Parsed integer value. Return type is `Integer`.

Flow.parseFloat

Parse float value given as string.

Parameters

Name	Type	Description
<code>str</code>	<code>String</code>	Input string.

Return value

Parsed float value. Return type is `Float`.

Flow.parseDouble

Parse double value give as string.

Parameters

Name	Type	Description
<code>str</code>	<code>String</code>	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
<code>str date</code>	<code>Date</code>	Input date.

Return value

Date string. Return type is `String`.

Date.toLocaleString

Converts given date to locale string.

Parameters

Name	Type	Description
<code>str date</code>	<code>Date</code>	Input date.

Return value

Date string. Return type is `String`.

Date.fromString

Converts string to date.

Parameters

Name	Type	Description
<code>dateStr</code>	<code>String</code>	Input string.

Return value

Date. Return type is `Date`.

Date.getYear

Get year from date.

Parameters

Name	Type	Description
<code>date</code>	<code>Date</code>	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
<code>year</code>	<code>Integer</code>	Year
<code>month</code>	<code>Integer</code>	Month
<code>day</code>	<code>Integer</code>	Day
<code>hours</code>	<code>Integer</code>	Hours
<code>minutes</code>	<code>Integer</code>	Minutes
<code>seconds</code>	<code>Integer</code>	Seconds
<code>milliseconds</code>	<code>Integer</code>	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
<code>x</code>	<code>Integer Float Double</code>	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
<code>x</code>	<code>Integer Float Double</code>	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
<code>base</code>	<code>Integer Float Double</code>	Year
<code>exponent</code>	<code>Integer Float Double</code>	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 smallest 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
<code>string</code>	<code>String</code>	A string.
<code>substring</code>	<code>String</code>	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
<code>string</code>	<code>String</code>	A string.
<code>targetLength</code>	<code>Integer</code>	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.
<code>padString</code>	<code>String</code>	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
<code>string</code>	<code>String</code>	A string.
<code>separator</code>	<code>Integer</code>	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
<code>array</code>	<code>Array json</code>	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
<code>array</code>		<code>Array</code>	An array.
<code>start</code>		<code>Integer</code>	Zero-based index at which to start extraction.
<code>end</code>	Yes	<code>Integer</code>	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
<code>size</code>	<code>Array</code>	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
<code>array</code>	<code>Array json</code>	An array.
<code>value</code>	<code>Any</code>	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
<code>array</code>	<code>Array json</code>	An array.
<code>position</code>	<code>Integer</code>	Zero-based index at which new element will be inserted.
<code>value</code>	<code>Any</code>	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 json	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. JSON***JSON.get***

Get the property value of the JSON object.

Parameters

Name	Type	Description
json	json	A JSON object.
property	string	Path to the property. For example: "users.3.name"

Return value

The value of the property. Return type can be another JSON object, number, string, boolean or date.

JSON.clone

Deep clone of the JSON object.

Parameters

Name	Type	Description
json	json	A JSON object.

Return value

A new JSON object. Return type is `json`.

P8.3.8. LVGL***LVGL.MeterTickIndex***

See the LVGL Meter Widget description (Chapter W33) 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. 87). Expressions can be entered manually or using the *Expression Builder*.

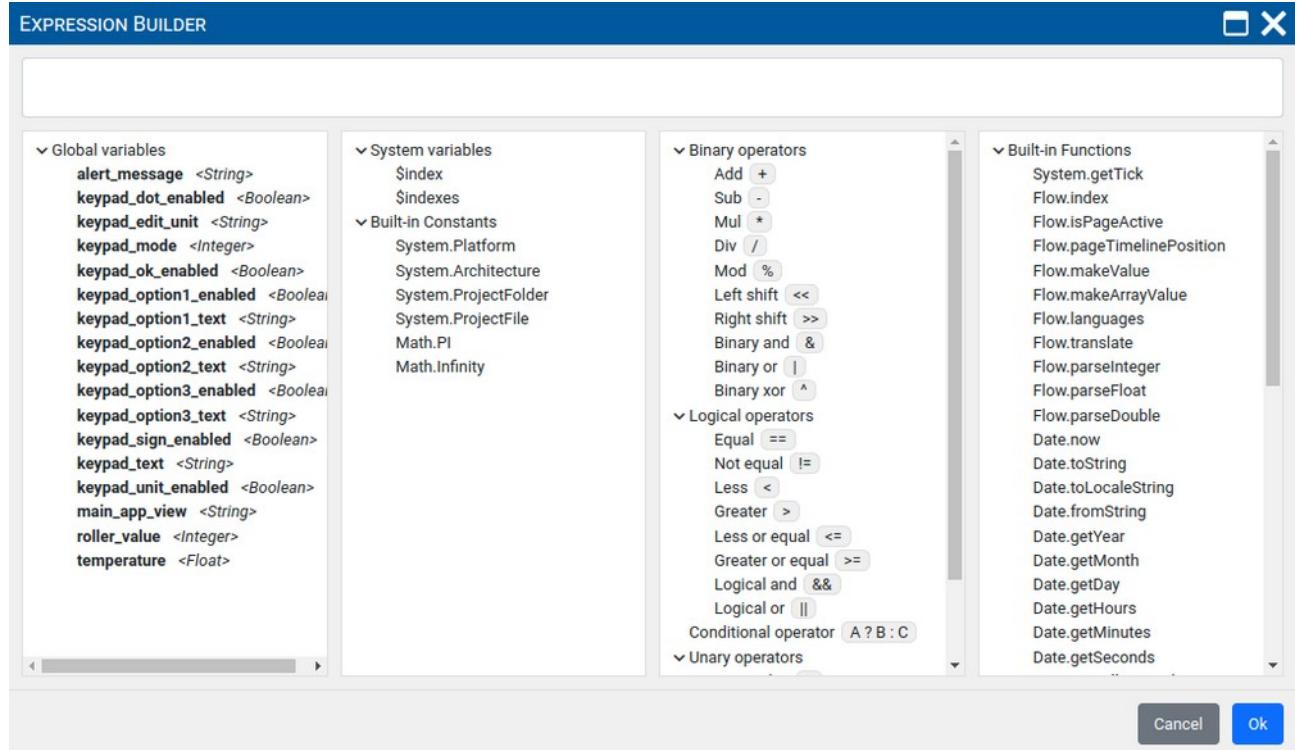


Fig. 87: Expression builder

P9. Styles and Color themes

P9.1. Overview

Styles make it easy to define a whole range of visual attributes and unify the appearance of widgets. Style attributes can be set at multiple levels, i.e. there are multiple scopes.

The base level/scope is Local: all style attributes will only apply to that Widget. When we want to use the same style on several Widgets, we can use the Project style defined in the *Styles* panel.

By using Project styles, instead of local style modifications, consistency is achieved (Widgets of similar purpose have the same visual appearance) and sustainability (a change to an attribute in a Style used by multiple Widgets is automatically propagated to all Widgets that use it).

Style attributes are inherited, which means that one attribute of a Style can inherit attribute from another Style. i.e. the Child style inherits all properties of the Parent style.

Additionally, all style attributes that contain a color definition can inherit the color from the *Color Theme* as explained in Section P9.4.2

P9.2. Style properties

Fig. 88 shows how for the selected widget (1) the locally defined style parameters can be found in the *Style* subsection (2) within the widget *Properties* panel.

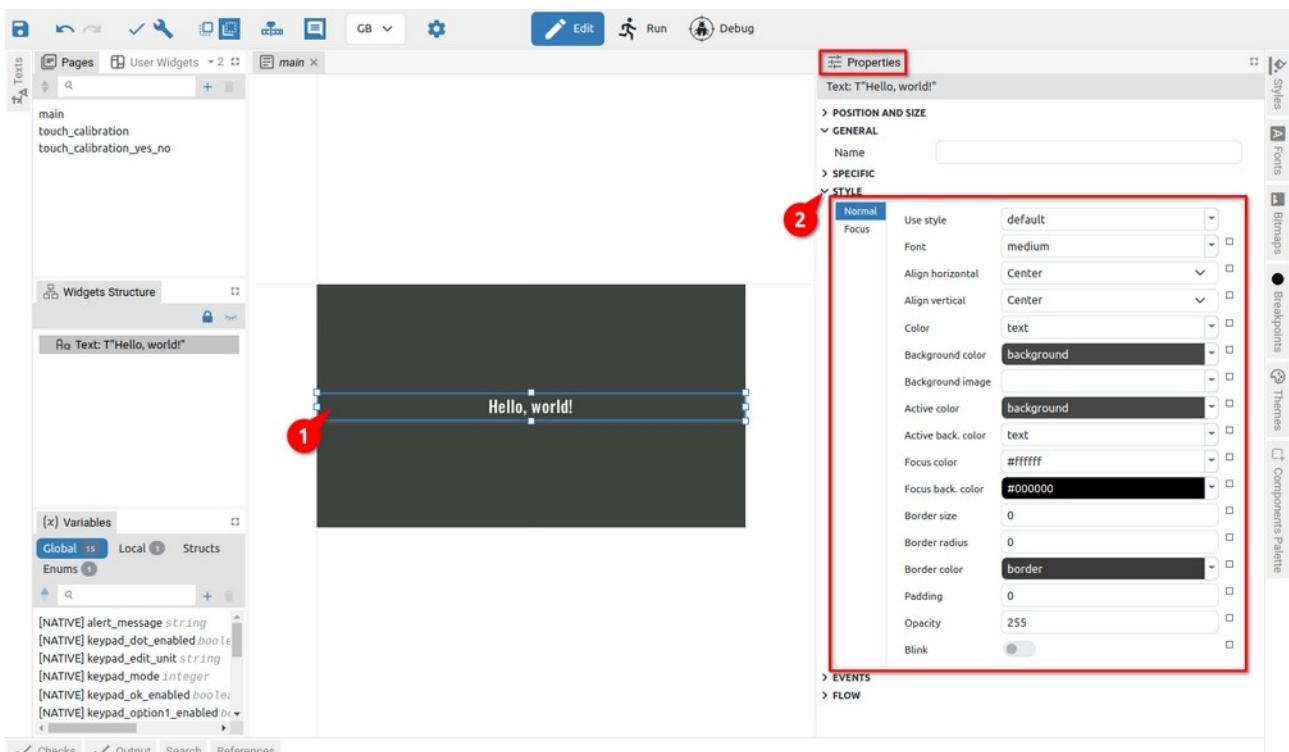


Fig. 88: Selected widget Style section for EEZ-GUI project type

The number of attributes and the appearance of the *Style* section depends on the project type in which the widget is used. In Fig. 88 the *Style* section for the *EEZ-GUI* project type is shown. Fig. 89 shows an example of the *Style* for the `Button` widget in the *Dashboard* project, and Fig. 90 example *Style* for `Label` widget in *LVGL* project.

The number of style attributes varies by Widget type and Project type. Widgets that can have multiple states (`Default`, `Focused`, `Disabled`, ...) will be able to define style attributes separately for all states.

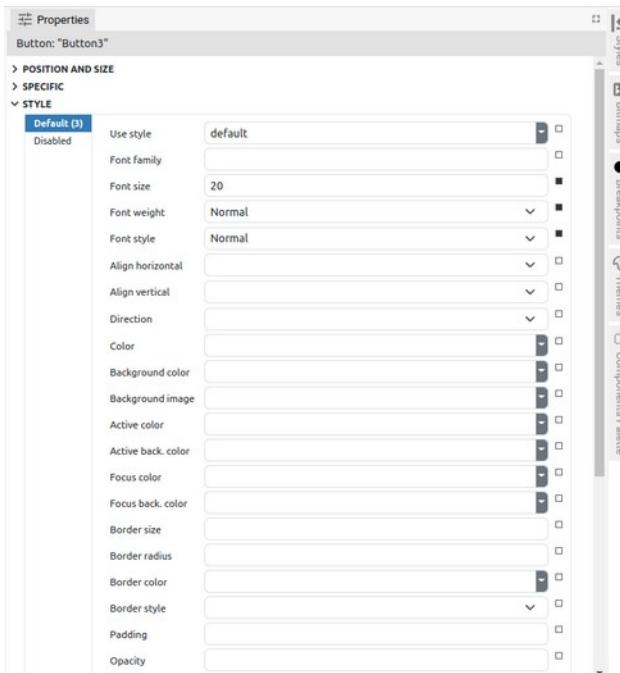


Fig. 89: Style properties (Dashboard project)

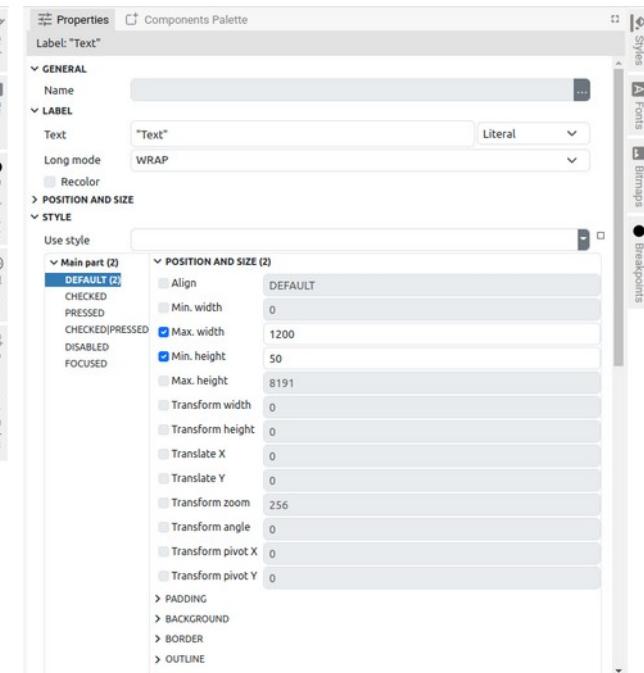


Fig. 90: Style properties (LVGL project)

P9.3. Project Styles

As shown in Fig. 88, Project styles have their own panel (1) where they can be searched, added and deleted. For the selected Project style, all properties will be displayed in the Properties panel, i.e. the same one used to display Widget and Action properties (5), and the name of the selected Style is displayed at the top (4).

Below the list of Project styles, a preview is shown. *EEZ-GUI* project styles have two previews: the first when **Color / Background Color** is used (2), and the second when **Active Color / Active back. color** is used (3). In Fig. 92 the preview section for the *LVGL* project is shown (6).

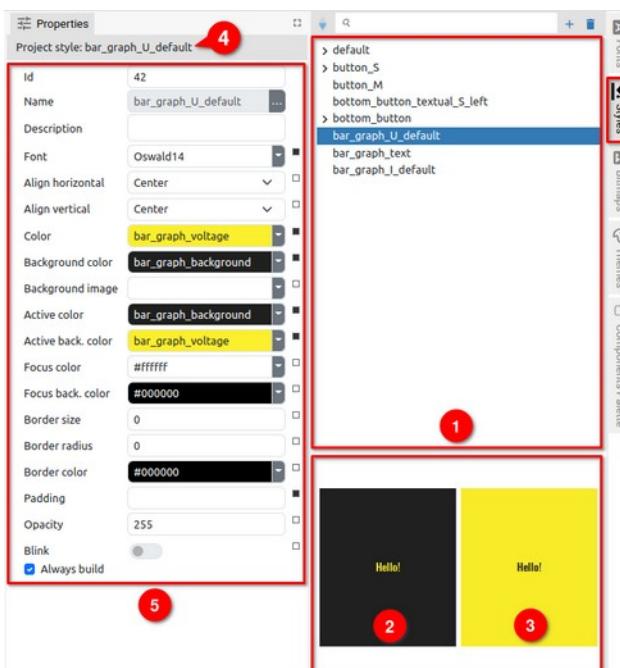


Fig. 91: Project Style Panel and Properties

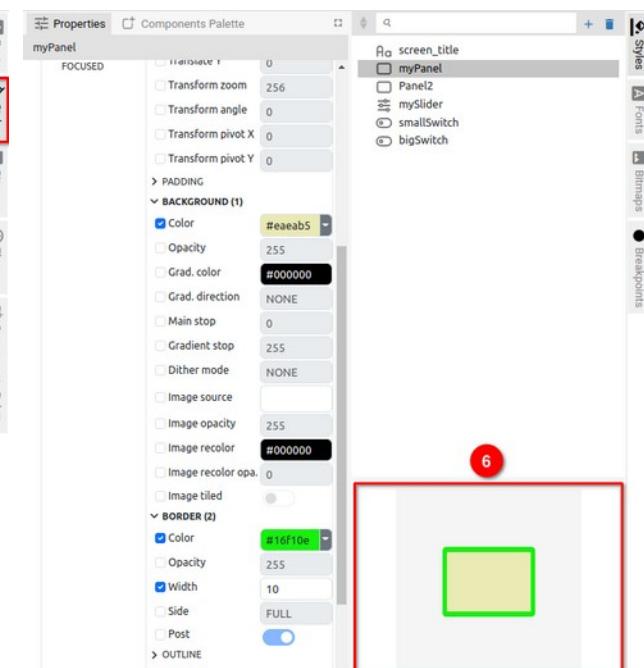


Fig. 92: LVGL Project style

P9.3.1. Creating a new Style

When creating a new Project style, it will be necessary to define the *Name* (Fig. 93) that must be unique. In the *LVGL* project, it will be necessary to choose for which Widget type it will be applied (Fig. 94).



Fig. 93: Adding a new Style



Fig. 94: Adding a new Style (LVGL)

A new Project style can be conveniently created directly from the local style of the selected Widget using the *Create New Style* option from the popup menu of the *Use Style* attribute (Fig. 95).

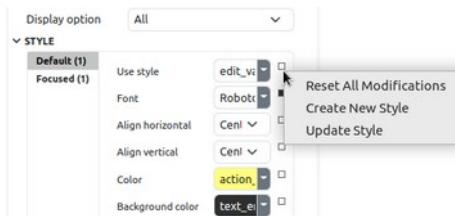


Fig. 95: Add new Style from currently selected Widget

Item	Description
<i>Reset All Modification</i>	Resetting (clearing) all local changes.
<i>Create New Style</i>	Creating a new Project style using the style settings of the currently selected Widget (opens a dialog box for creating a new Project style as in Fig. 93 or Fig. 94). After successful creation of the Style, it will be assigned to the selected Widget, too.
<i>Update Style</i>	The Project style used by the Widget will be updated with local modifications. Therefore the local modifications will be applied to all other Widgets that use the same Project style.

P9.4. Style hierarchy

Project styles can inherit properties, so a "child" style inherits all the properties of its "parent". The "Child-parent" relationship is shown in the Project style sheet, where changing the position sets or resets the "child" relationship. Inheritance can be multi-level, i.e. one "child" can become the parent of another "child". For example, in Fig. 96 Style *edit_value_active_M_center* has two child Styles (1): *edit_value_active_S_center* and *edit_value_active_M_left*. Child *edit_value_active_S_center* is the parent of two other child Styles (2): *icon_and_text_S* and *edit_value_active_S_center_icon*.

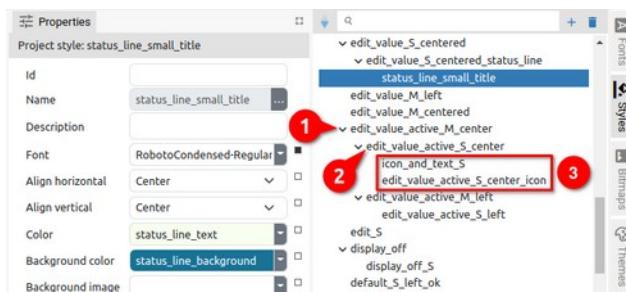


Fig. 96: Styles hierarchy

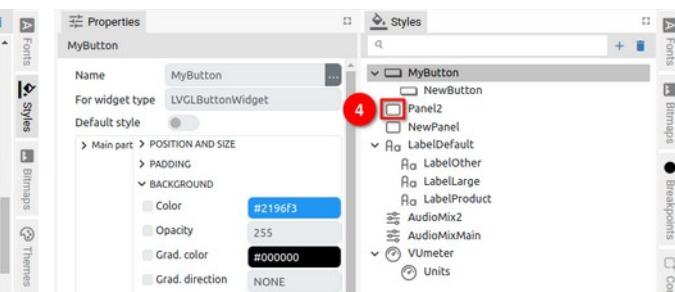


Fig. 97: Styles hierarchy in LVGL project

Since the Project styles in the *LVGL* project also have a Widget type defined, it is possible to establish

a "child-parent" relationship only between styles for the same Widget type. For this reason, in the Style Panel of the *LVGL* project (Fig. 97), an icon of the Widget type is displayed (4) in addition to the name of the Style, so that there is no need to guess whether the selected Style can become a child of a certain Parent or not.

Setting the Style as child is easily achieved with drag & drop as shown in Fig. 98: click and hold the Style you want to become a child (1); drag to the Style that will be the parent until the navigation line appears (2). Move the cursor to the right so that the beginning of the line is indented relative to the name of the parent Style. Finally, make drop and Style will appear indented and below its parent (3).



Fig. 98: Set child Style position

Resetting the child position is also carried out with drag & drop as shown in Fig. 99.

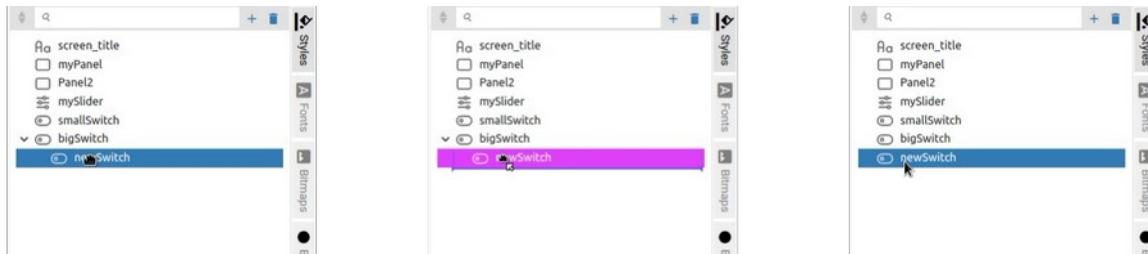


Fig. 99: Reset child Style position

P9.4.1. Setting the Style attribute color from the palette

A style can contain multiple attributes that define the color of a Widget part. Setting the color can be done in two ways and the first is by using the Color picker.

Figure 100 illustrates the selection of a style in the EEZ Studio Style panel. The 'Properties' tab is active, showing the configuration for the style 'Project style:bar_graph_I_default'. Step 1 points to the 'Color' dropdown menu for the attribute 'bar_graph_I_default', which is currently set to 'bar_graph_current'. Step 2 points to the 'Name' field for the style itself.

Figure 101 shows the color picker interface used to set the color for the style attribute. Step 3 points to the color picker itself, displaying a gradient from black to red. Step 4 points to the color preview window showing a red square. Step 5 points to the 'Hex' color code input field, which contains the value 'EE3157'.

Fig. 100: Selecting a Style in the Style panel

Fig. 101: Setting the Style attribute color

In Fig. 100 and Fig. 101 shows how it can be done for the Project style: the style should be selected in

the list (1) when its name will appear in the Properties panel (2). The attribute to which we want to set the color should be clicked on (3) to open the Color picker. By moving the cursor around the palette, we select a color whose hex and RGB value is simultaneously displayed in the lower section through which it is also possible to directly enter the desired hex or RGB value.

P9.4.2. Setting the Style attribute color using the Color theme

Another way to set the attribute color is by using the Color theme. The color theme must be selected from the Theme panel (1) and in the Styles panel select the Project style to which we want to set the color attribute.

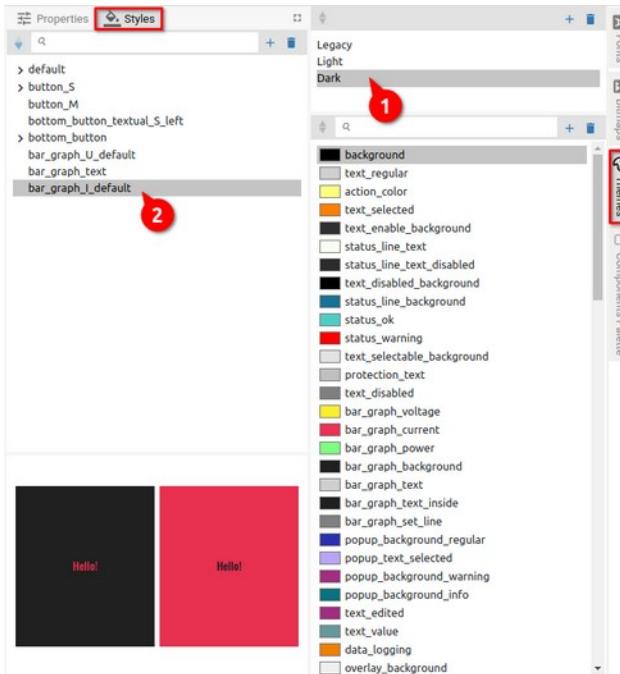


Fig. 102: Selecting a Style in the Style panel

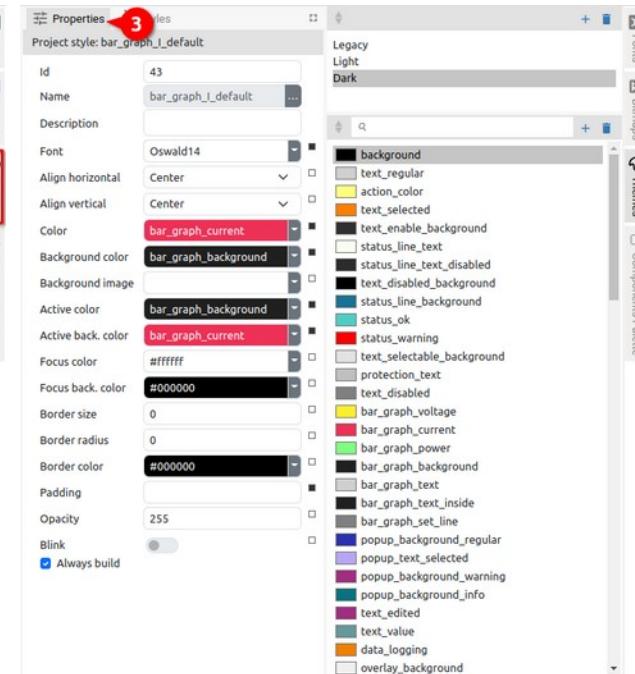


Fig. 103: Displaying the properties of the selected Style

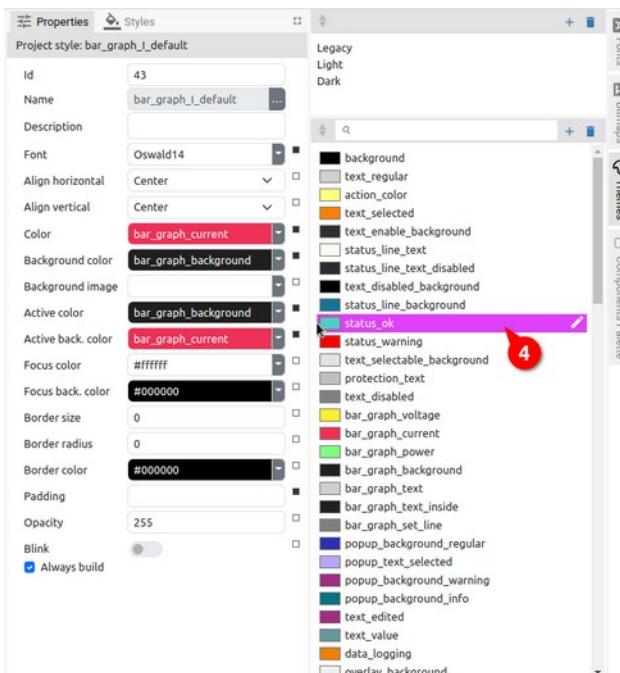


Fig. 104: Selecting a color from the Color theme

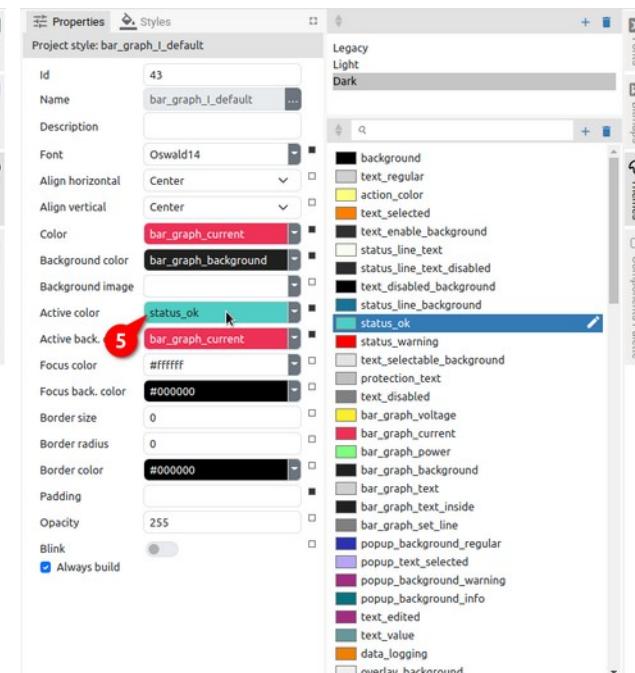


Fig. 105: Changing the color of the Style attribute

Now we need to select the Properties panel (3) in which all the attributes of the selected Style will be displayed. In the Theme panel, select the color (4) that we want to assign to the Style attribute and use drag and drop to place it in the name field of the attribute (5).

The attribute color can also be set by entering the name of the color from the Color theme, in our example it would be `status_ok`.

P9.5. Style attributes

P9.5.1. EEZ-GUI project

In Fig. 106 shows a LineChart widget that has several style definitions: Normal, Title, Legend, X axis, Y axis and Marker (1). The first attribute `Use style` (2) determines from which Project style the attributes will be inherited and it can be defined separately for each of the style definitions.

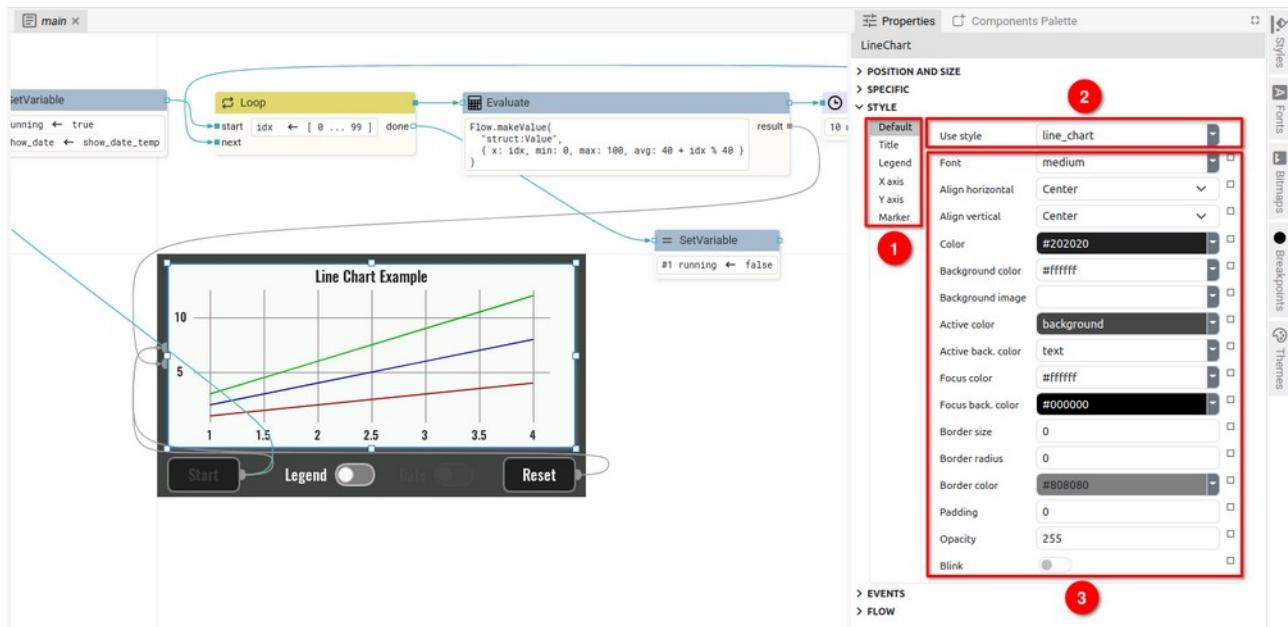


Fig. 106: EEZ-GUI project local Style attributes

Changing the `Use style` immediately propagates the attributes of the selected Project style to the corresponding attributes of the widget (3). All attributes listed below that contain a color value is either a hex color definition (e.g. `#4beef2`) or a color name as defined in the Color theme.

Item	Description
<code>Use style</code>	The name of the Style from which this widget inherits style attributes. If there is some locally modified attribute then it has precedence over this style definition. This attribute can be left empty, which means that this widget doesn't inherit any attribute from other Style and only local settings are used.
<code>Font</code>	The font for the texts displayed inside this Widget.
<code>Align horizontal</code>	Horizontal text alignment.
<code>Align vertical</code>	Vertical text alignment.
<code>Color</code>	The color of the text.
<code>Background color</code>	The background color of the Widget.
<code>Background image</code>	The background image of the Widget.
<code>Active color</code>	The color of the text when Widget is active. For example: <ul style="list-style-type: none"> • <code>Button</code> widget is active when it is clicked • when <code>Blink</code> property is enabled in <code>Text</code> widget it switches periodically between <code>Normal</code> and <code>Active</code> state.

Active back. color	The background color of the Widget when it is active.
Focus color	The color of the text when Widget is in focus.
Focus back. color	The background color of the Widget when it is in focus.
Border size	The line width used to draw the border.
Border radius	The radius of the border corner. It can be given as 1, 2 or 4 numbers separated by a space, with the following meaning:

- `radius` – sets the same radius value for all corners
- `radius1 radius2` – sets different radii for top-left / bottom-right corner (`radius1`) and for top-right / bottom-left corner (`radius2`)
- `radius1 radius2 radius3 radius4` – sets different values for each corner in this order: top-left, top-right, bottom-right, and bottom-left.

Border color	The color used to paint the border. Below are examples of different borders:
--------------	---



Padding	The offset of the text. It can be given as 1, 2 or 4 numbers separated by a space, with the following meaning:
---------	--

- `padding` – set the same number for all the sides
- `padding1 padding2` – `padding1` is for top / bottom and `padding2` is for left / right.
- `<padding1> <padding2> <padding3> <padding4>` – sets different values for each side in this order: top, right, bottom, and left.

Opacity	0 – fully transparent, 255 – fully opaque.
---------	--

Blink	If enabled, the Widget periodically switches between <i>Normal</i> and <i>Active</i> State. Use different normal and active colors to achieve blink effect.
-------	---

P9.5.2. Dashboard project

Dashboard project styles are based on CSS styles: <https://developer.mozilla.org/en-US/docs/Web/CSS>. The `use style` attribute has the same function as in the EEZ-GUI project.

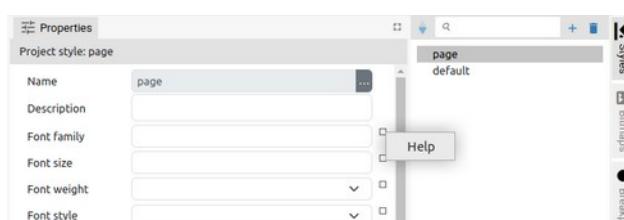


Fig. 107: Style attribute help

For each Style attribute, there is a *Help* link (Fig. 107) to open the corresponding help page for CSS properties on Mozilla's website. You can consult the CCS documentation for an explanation of the following attributes:

- Font family
- Font weight
- Font style
- Align horizontal
- Align vertical
- Direction
- Color
- Background color
- Background image
- Active color
- Active back. color
- Focus color
- Focus back. color
- Border size
- Border radius
- Border style
- Padding
- Opacity
- Box shadow

Use `Blink` attribute (1) to achieve Widget blinking (Fig. 108). The generated CSS can be checked in the `CSS preview` (2) when this attribute is enabled. Use `Additional CSS` section to enter any custom CSS properties (3).



Fig. 108: Dashboard style blink attribute

In `CSS preview`, which is a read-only section, there is a summary of the all CSS properties generated for the Style, including parent styles and local modifications (commented as `/* inline style */`).

P9.5.3. LVGL project

Style definition in *LVGL* project are grouped in *Parts*, *States* and *Categories*. Each Widget type can have different *Parts* which can be differently customized with the *Styles*.

In the example from Fig. 109 `Slider` widget shown has three different *Parts*: `Main`, `Indicator` and `Knob` (1). For each *Parts*, it is possible to separately define the attributes of six possible *States*: `Default`, `Checked`, `Pressed`, `Checked|Pressed`, `Disabled` and `Focused` (2). Finally, for each *State* it is possible to define 72 attributes that are grouped into 11 *Categories* i.e. `Position` and `Size`, `Padding`, `Background`, ... (3).

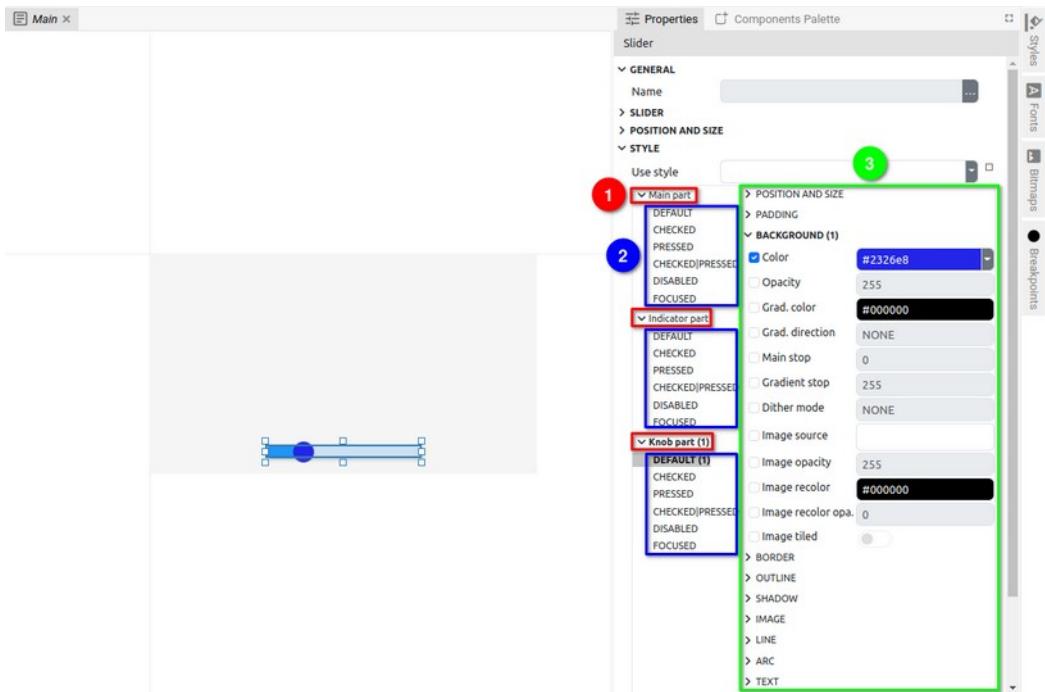


Fig. 109: LVGL project local Style attributes

Unlike the *EEZ-GUI* and *Dashboard* projects, LVGL Style attributes are changed by checking the checkbox to the left of the attribute name.

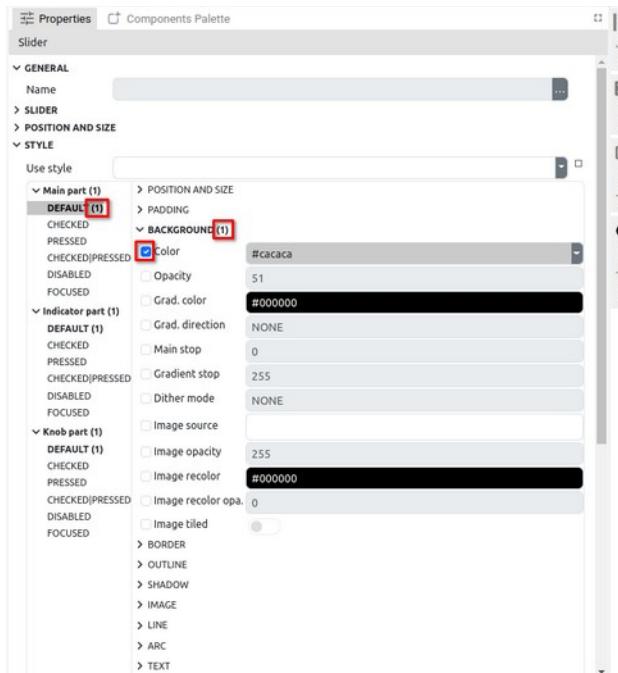


Fig. 110: Indication of modified attributes

In the category name, state name and part name there is an indication how many properties are changed (Fig. 110).

You can find out more about LVGL widget styles on the official pages of the *LVGL* project: <https://docs.lvgl.io/latest/en/html/overview/style.html>.

In the subsection <https://docs.lvgl.io/latest/en/html/overview/style.html#properties> there is a list with explanations of each attribute (i.e. properties).

P9.5.4. Inheriting local Style attributes

In Section P9.3.1 it was mentioned that local style can be used to create Project style (which can be assigned to other Widgets). Likewise, `Use style` is used to set a local style from the list of Project styles defined through the Style panel (Fig. 111). In the case of an LVGL project where the Project style refers to a specific Widget type, a list of only those Project styles for the corresponding Widget type will be displayed (Fig. 112).

In addition to the selection from the list, the style can also be set by directly entering a valid name (case sensitive).

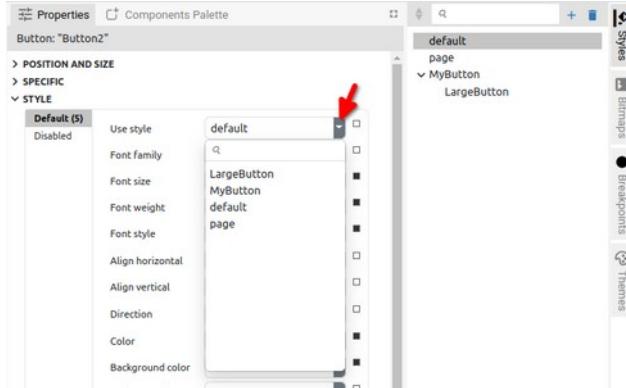


Fig. 111: Set a style from the Project style list

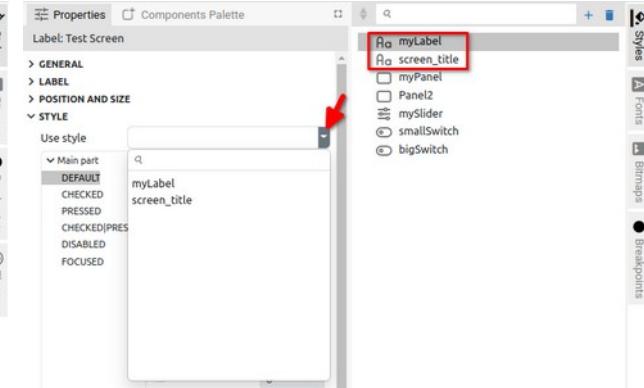


Fig. 112: Set a style from the Project style list (LVGL)

In the *EEZ-GUI* and *Dashboard* project type, on the right side of each attribute there is an indicator whether the attribute has been locally changed (filled square, see Fig. 113) or not (empty square). In the latter case, the name of the Project style whose attributes it inherited will be displayed at the mouse hover (Fig. 114).

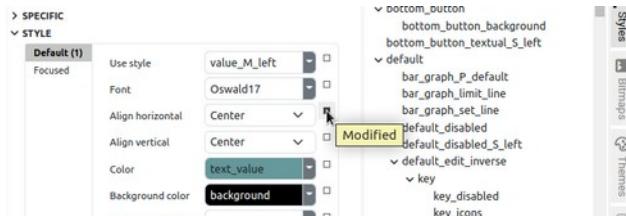


Fig. 113: Modified Style attribute indication

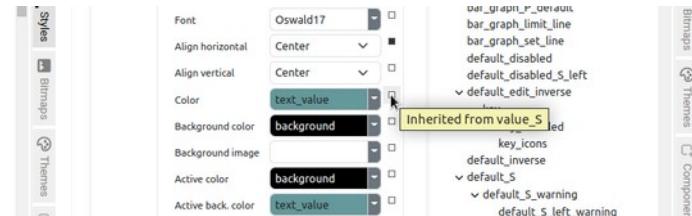


Fig. 114: Inherited Style attribute indication

An attribute that has been changed can be reverted to the original value it inherited from the set style. For this, it is necessary to select the *Reset* option from the popup menu on the indicator (Fig. 115). In case the attribute defines a background color, an option to set a transparent background will also appear in the menu as shown in Fig 116.

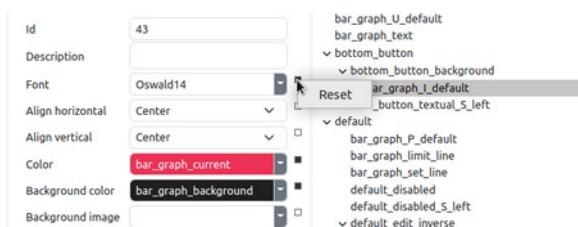


Fig. 115: Reset local style modification



Fig. 116: Reset local style modification (Background)

P10. Bitmaps

In Fig. 117 shows the *Bitmap* panel (1) in which there is a list of bitmaps that can be used in the project. The project can contain an unlimited number of bitmaps, and for the selected bitmap (2) you can see the size in pixels (3) and image preview (4).



Fig. 117: Bitmap panel

P10.1. Adding a bitmap

When adding a bitmap, a new dialog opens with a different number of parameters depending on the type of project. Fig. 118 shows the dialog for *EEZ-GUI*, Fig. 119 shows the *Dashboard* and Fig. 120 shows the *LVGL* project.



Fig. 118: Add new bitmap in EEZ-GUI project

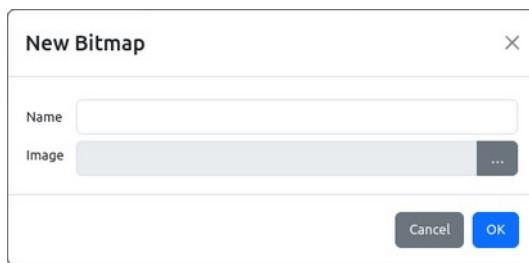


Fig. 119: Add new bitmap in Dashboard project

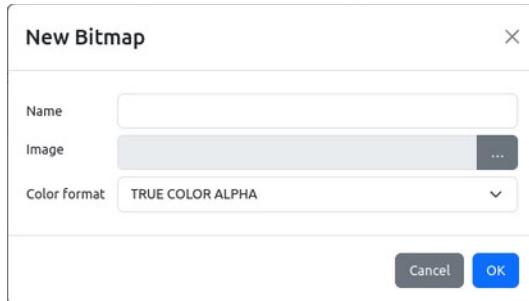


Fig. 120: Add new bitmap in LVGL project

Item	Description
Name	Bitmap is referenced in other parts of project by its name.
Image	Selection of bitmap file from local storage.
Bits per pixel (EEZ-GUI only)	Color depth: 16 (RGB565) or 32 (RGBA, i.e. 24-bit color + Alpha channel).
Color format (LVGL only)	Described in https://docs.lvgl.io/8.3/overview/image.html#color-formats

P10.2. Bitmap properties

Fig. 121 shows the properties for the bitmap from the *Bitmap* Panel for the EEZ-GUI project. The parameters are described in the following table.

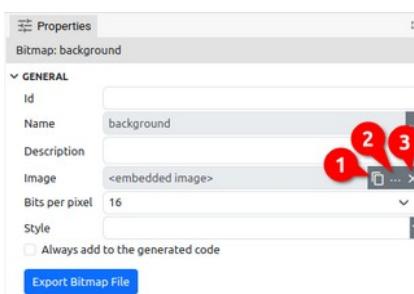


Fig. 121: Bitmap properties (EEZ-GUI)

Item	Description
<i>Id</i> (EEZ-GUI only)	<i>Bitmap</i> is one of resources that is referenced by name in the project editor. When that project is built, names are no longer used, but numerical ID. This field is optional, i.e. an ID does not have to be specified, in which case an ID will be assigned during the build. However, if we want an object to always get the same ID, then it needs to be defined. Why would we want to always have the same ID? This is necessary when there is a master project such as <i>modular-psu-firmware.eez-project</i> from EEZ BB3 and that master project is used by BB3 Applets and BB3 MicroPython scripts and they can use resources from the master project that have that ID defined. <i>Important:</i> once the ID is set, it should not be changed, otherwise all BB3 scripts that depend on it should be rebuilt.
<i>Name</i>	Bitmap is referenced in other parts of project by its name. Use the ... button to change the existing name.
<i>Description</i>	Optional field, contains a description of the bitmap.
<i>Image</i>	This is the image file itself that is saved within the project file (embedded within project file). Options for Copy to clipboard (1), Paste from clipboard (2) and loading from local storage (3) are also available.

<i>Bits per pixel (EEZ-GUI only)</i>	16 – RGB565 32 – RGBA																																
<i>Style (EEZ-GUI only)</i>	The option is only enabled if <i>Bits per pixel</i> is set to 16. Only the background color from the entire style is used. If there is a transparent pixel in the default bitmap, then the background color will be displayed.																																
<i>Always add to the generated code (EEZ-GUI only)</i>	During the project build, i.e. when the source code is generated, only those Bitmaps that are used in the project will be inserted into the source code. However, if a bitmap is used within the native code but not in the EEZ Studio project, then using this option you can force the bitmap to be added to the source code even though it is not used in the project.																																
<i>Color format (LVGL only)</i>	Described in https://docs.lvgl.io/8.3/overview/image.html#color-formats Below are the LVGL constant names and their counterparts used in EEZ Studio.																																
<table border="0"> <thead> <tr> <th>LVGL constant name</th> <th>EEZ Studio value</th> </tr> </thead> <tbody> <tr> <td>LV_IMG_CF_ALPHA_1_BIT</td> <td>ALPHA 1 BIT</td></tr> <tr> <td>LV_IMG_CF_ALPHA_2_BIT</td> <td>ALPHA 2 BIT</td></tr> <tr> <td>LV_IMG_CF_ALPHA_4_BIT</td> <td>ALPHA 4 BIT</td></tr> <tr> <td>LV_IMG_CF_ALPHA_8_BIT</td> <td>ALPHA 8 BIT</td></tr> <tr> <td>LV_IMG_CF_INDEXED_1_BIT</td> <td>INDEXED 1 BIT</td></tr> <tr> <td>LV_IMG_CF_INDEXED_2_BIT</td> <td>INDEXED 2 BIT</td></tr> <tr> <td>LV_IMG_CF_INDEXED_4_BIT</td> <td>INDEXED 4 BIT</td></tr> <tr> <td>LV_IMG_CF_INDEXED_8_BIT</td> <td>INDEXED 8 BIT</td></tr> <tr> <td>LV_IMG_CF_RAW</td> <td>RAW</td></tr> <tr> <td>LV_IMG_CF_RAW_CHROMA</td> <td>RAW CHROMA</td></tr> <tr> <td>LV_IMG_CF_RAW_ALPHA</td> <td>RAW ALPHA</td></tr> <tr> <td>LV_IMG_CF_TRUE_COLOR</td> <td>TRUE COLOR</td></tr> <tr> <td>LV_IMG_CF_TRUE_COLOR_ALPHA</td> <td>TRUE COLOR ALPHA</td></tr> <tr> <td>LV_IMG_CF_TRUE_COLOR_CHROMA_KEYED</td> <td>TRUE COLOR CHROMA</td></tr> <tr> <td>LV_IMG_CF_RGB565A8</td> <td>RGB565A8</td></tr> </tbody> </table>		LVGL constant name	EEZ Studio value	LV_IMG_CF_ALPHA_1_BIT	ALPHA 1 BIT	LV_IMG_CF_ALPHA_2_BIT	ALPHA 2 BIT	LV_IMG_CF_ALPHA_4_BIT	ALPHA 4 BIT	LV_IMG_CF_ALPHA_8_BIT	ALPHA 8 BIT	LV_IMG_CF_INDEXED_1_BIT	INDEXED 1 BIT	LV_IMG_CF_INDEXED_2_BIT	INDEXED 2 BIT	LV_IMG_CF_INDEXED_4_BIT	INDEXED 4 BIT	LV_IMG_CF_INDEXED_8_BIT	INDEXED 8 BIT	LV_IMG_CF_RAW	RAW	LV_IMG_CF_RAW_CHROMA	RAW CHROMA	LV_IMG_CF_RAW_ALPHA	RAW ALPHA	LV_IMG_CF_TRUE_COLOR	TRUE COLOR	LV_IMG_CF_TRUE_COLOR_ALPHA	TRUE COLOR ALPHA	LV_IMG_CF_TRUE_COLOR_CHROMA_KEYED	TRUE COLOR CHROMA	LV_IMG_CF_RGB565A8	RGB565A8
LVGL constant name	EEZ Studio value																																
LV_IMG_CF_ALPHA_1_BIT	ALPHA 1 BIT																																
LV_IMG_CF_ALPHA_2_BIT	ALPHA 2 BIT																																
LV_IMG_CF_ALPHA_4_BIT	ALPHA 4 BIT																																
LV_IMG_CF_ALPHA_8_BIT	ALPHA 8 BIT																																
LV_IMG_CF_INDEXED_1_BIT	INDEXED 1 BIT																																
LV_IMG_CF_INDEXED_2_BIT	INDEXED 2 BIT																																
LV_IMG_CF_INDEXED_4_BIT	INDEXED 4 BIT																																
LV_IMG_CF_INDEXED_8_BIT	INDEXED 8 BIT																																
LV_IMG_CF_RAW	RAW																																
LV_IMG_CF_RAW_CHROMA	RAW CHROMA																																
LV_IMG_CF_RAW_ALPHA	RAW ALPHA																																
LV_IMG_CF_TRUE_COLOR	TRUE COLOR																																
LV_IMG_CF_TRUE_COLOR_ALPHA	TRUE COLOR ALPHA																																
LV_IMG_CF_TRUE_COLOR_CHROMA_KEYED	TRUE COLOR CHROMA																																
LV_IMG_CF_RGB565A8	RGB565A8																																
<i>Export bitmap file</i>	Use to export embedded image.																																

P10.3. Using a bitmap

Bitmap can be used in *Bitmap* widget (EEZ-GUI and Dashboard project), i.e. *Image* widget (LVGL project). It can also be used in *Style*.

Below is an example of using a bitmap in the *Bitmap* widget in the *Dashboard* project. Fig. 122 shows the added *Bitmap* widget (1) to the page. In the *Specific* section, we select which bitmap from the list of bitmaps we want to use (2), which in our example is the bitmap called *background* (3).

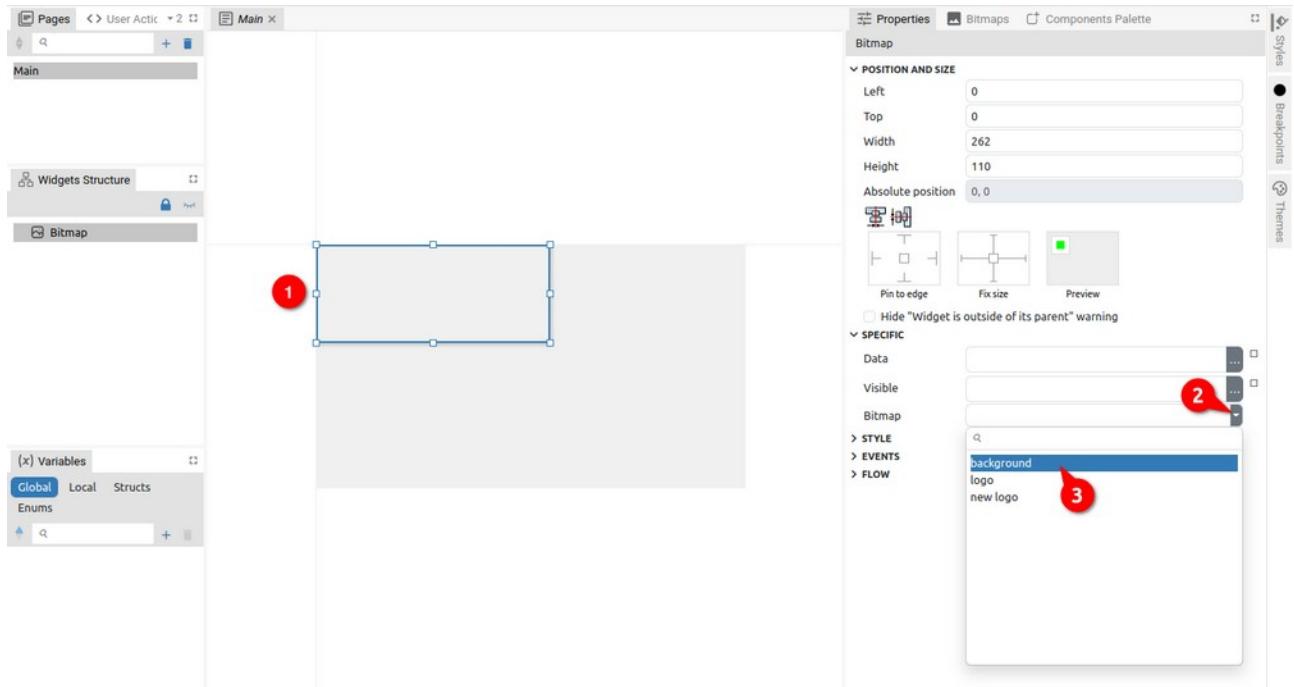


Fig. 122: Adding a bitmap to a Widget

Since the Widget's dimensions are smaller than the selected bitmap, the bitmap will exceed the Widget's borders as shown in Fig. 123. Here we can use the option *Resize to Fit Bitmap* (4) when the dimensions of the Widget will be adjusted to the size of the bitmap (Fig. 124).

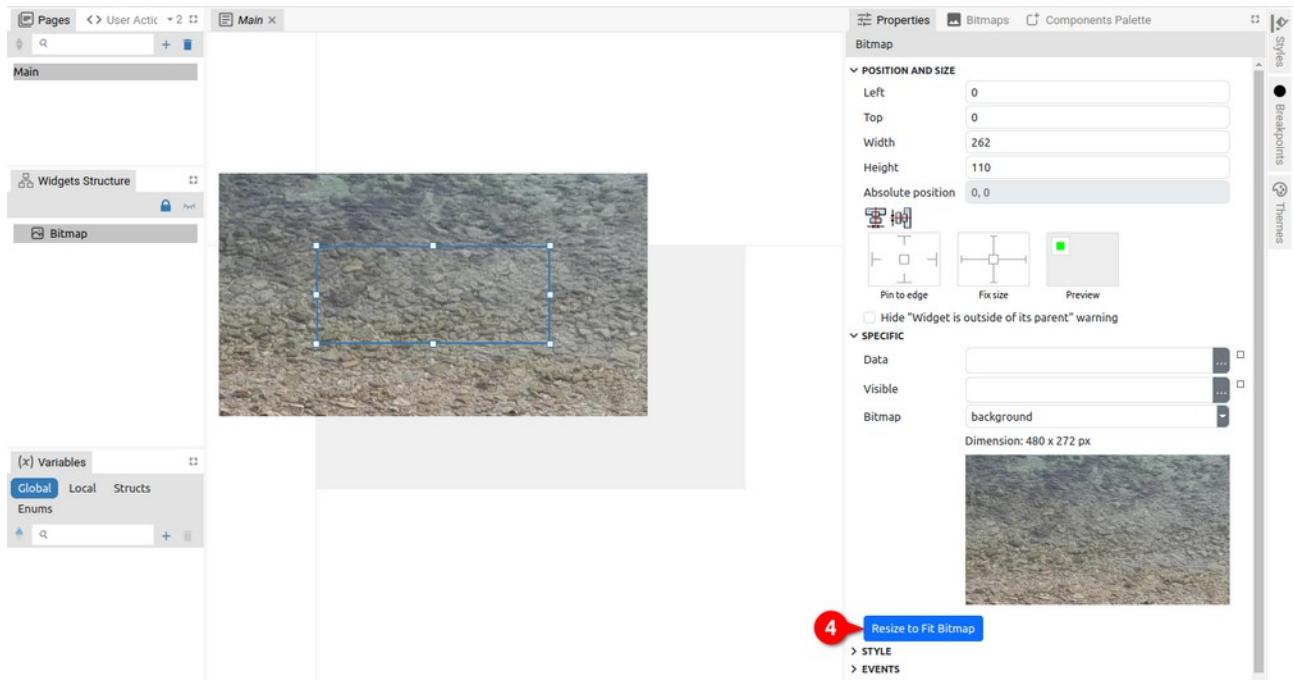


Fig. 123: Resizing the widget to fit the bitmap

Please note that this option is visible only if the current dimensions of the Widget do not match the dimensions of the bitmap (Fig. 124).

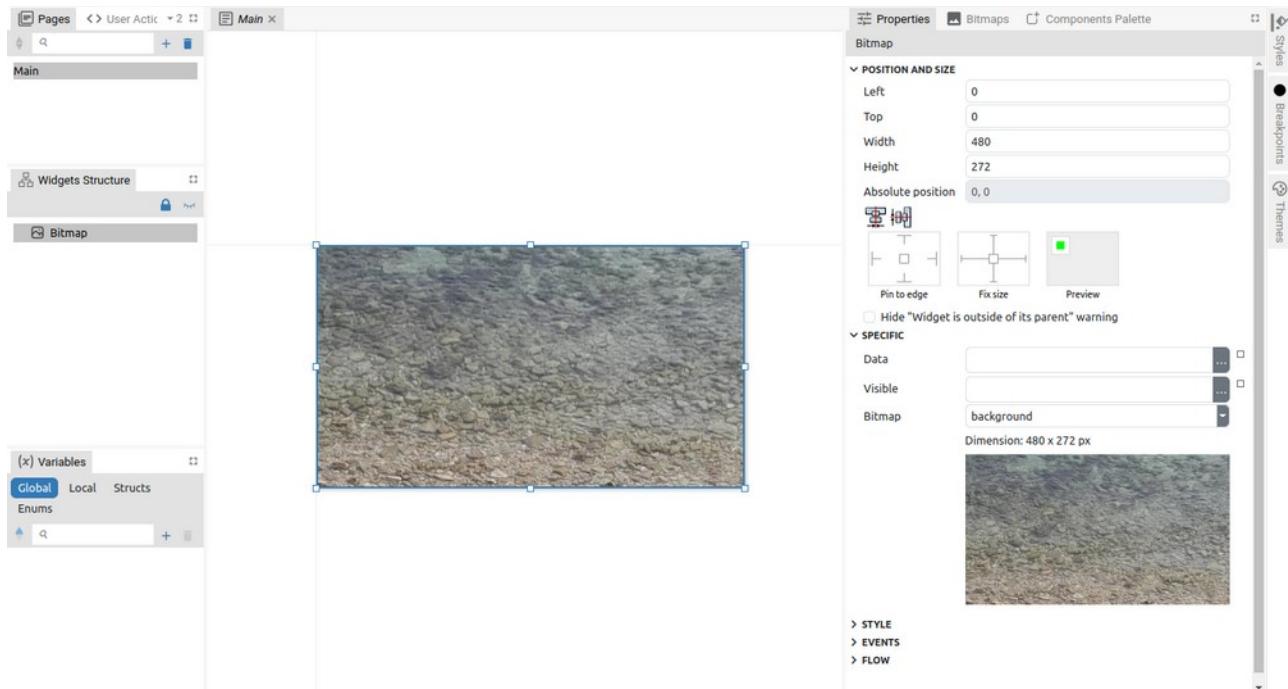


Fig. 124: Widget resized to bitmap size

P11. Fonts

The EEZ Studio project supports working with fonts. To work with fonts, it will be necessary in the project Settings (Fig. 125) under the *General* section (1) to enable the *Fonts* option in the project Settings (2).

The *Font* used consists of one or more characters taken from a TTF or OTF file and converted to anti-aliased bitmaps.

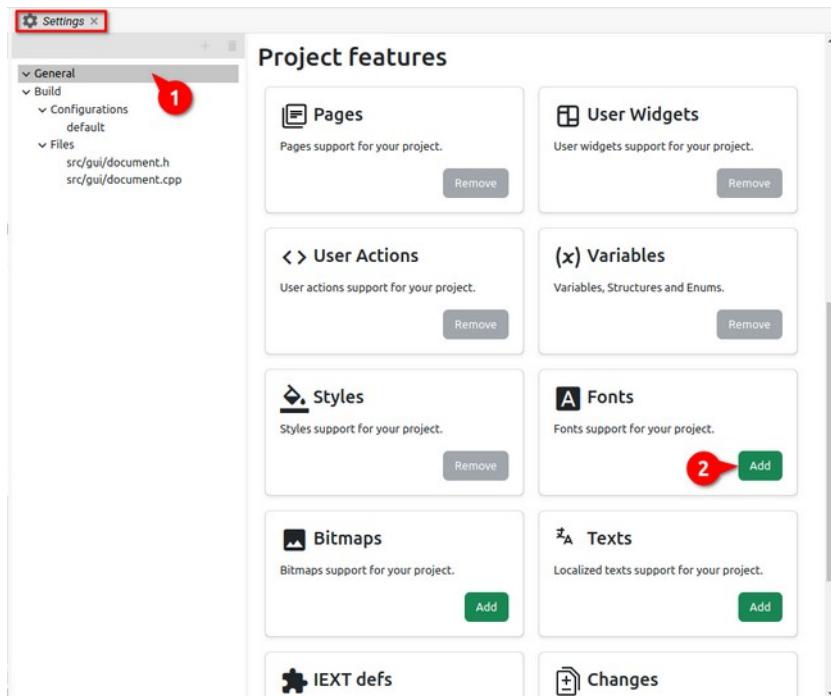


Fig. 125: Enable Fonts in project Settings

Fonts are defined only for *EEZ-GUI* and *LVGL* projects, and *Dashboard* projects do not use fonts. In the *Dashboard* project, vector fonts are used and the font is selected according to the name (*Font Family* attribute in *Style*).

In the *EEZ-GUI* project we have more options for editing fonts than in the *LVGL* project. Therefore, we will describe the work with fonts in those two types of projects in separate subsections.

P11.1. EEZ-GUI project fonts

P11.1.1. Add new font

To add a new font, it is necessary to select the *Add item* option in the *Fonts* panel, when the dialog shown in Fig. 126.

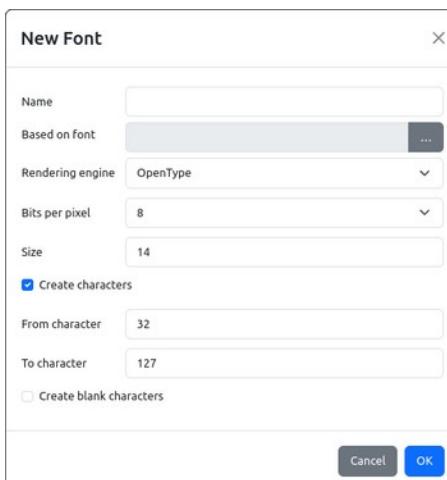


Fig. 126: Adding a new font to the EEZ-GUI project

Item	Description
<i>Name</i>	The name of the font to be used in the project.
<i>Font file</i>	Selection of font file from local storage.
<i>Rendering engine</i>	The rendering engine, which can be FreeType (https://freetype.org/) or OpenType (https://opentype.js.org/), converts from vector to bitmap format.
<i>Font size (points)</i>	Size is in points (pt). Use this formula to convert points to pixels: 1 pt = 1.333 px.
<i>Create characters</i>	If unchecked, not a single character will be created when adding a font, i.e. characters can be added later. If it is checked, then the range of characters to be created will need to be selected, and then <i>Create blank characters</i> can be used if we want all characters to be empty. These options are rarely used, and can be used to create icon fonts.
<i>From character</i>	Decimal number of the initial character we want to create (e.g. 32 = 0x20 = blank space).
<i>To character</i>	Decimal number of the final character we want to create.
<i>Create blank characters</i>	If it is enabled, all added characters will be empty.

After the font has been successfully added and the desired characters have been created, it is possible to view them in the table as shown in Fig. 127. For the selected character, its enlarged preview will be displayed on the right.

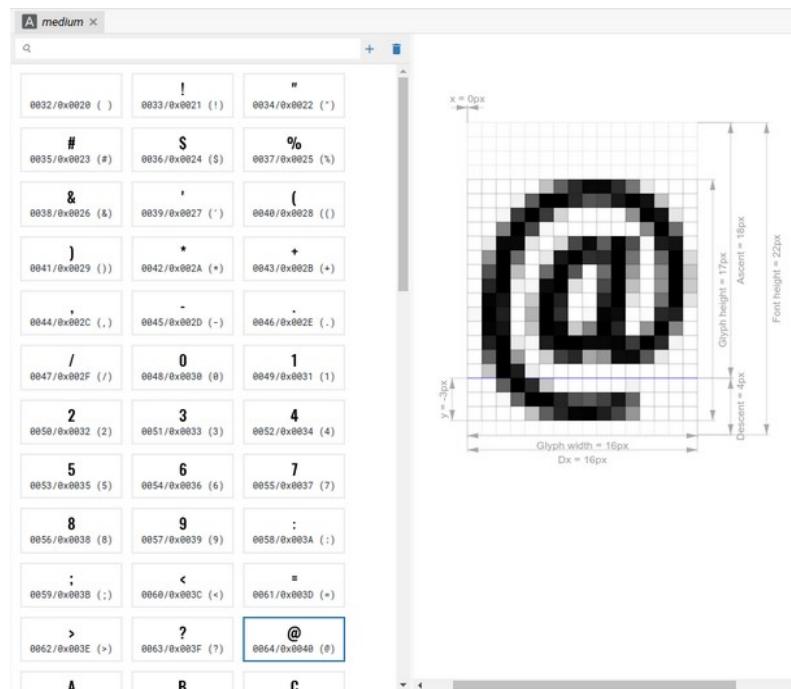


Fig. 127: Font character table (EEZ-GUI project)

P11.1.2. Add character

Once we have added the font to the project, it is possible to add new characters or delete existing ones. For this, we use the options shown in Fig. 128.



Fig. 128: Add/delete font character

When adding a new character, a dialog opens as shown in Fig. 129.

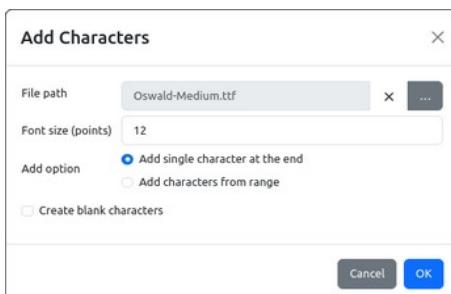


Fig. 129: Add new font character

Item	Description
<i>File path</i>	File path to the font file on local storage. An existing one can be deleted or a new one can be added.
<i>Font size (points)</i>	Size is in points (pt). Use this formula to convert points to pixels: 1 pt = 1.333 px.
<i>Add option</i>	
<i>Add single character at the end</i>	Adding only one character to the end of the table.
<i>Add characters from range</i>	Adding two or more characters from a defined range.
<i>Add missing characters</i>	The option is available only if multi-language is used (<i>Texts</i> panel, see Chapter P12) and there is a character in one of the strings that is not present in the font.
<i>Create blank characters</i>	If it is enabled, all added characters will be empty.

P11.2. LVGL project fonts

P11.2.1. Add new font

To work with fonts in the *LVGL* project, the library https://github.com/lvgl/lv_font_conv is used. To add a new font, it is necessary to select the *Add item* option in the *Fonts* panel, when the dialog shown in Fig. 130.

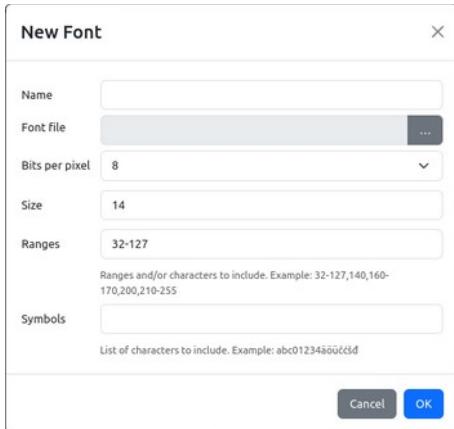


Fig. 130: Adding a new font to the LVGL project

Item	Description
Name	The name of the font to be used in the project.
Font file	Selection of font file from local storage.
Bits per pixel	1, 2, 4, or 8-bits. Defines the number of shades to be used for anti-aliasing. The higher the number, the softer the characters will look, but the font will also use more storage memory.
Font size (pixels)	Size in pixels (px).
Ranges	Defines ranges and/or characters to include.
Symbols	List of characters to include.

In Fig. 131 shows the properties of the selected font, and Fig. 9 properties of the selected character from the font table. All properties are informative in nature, i.e. cannot be changed, except that the Description for the Font can be edited.



Fig. 131: Font properties

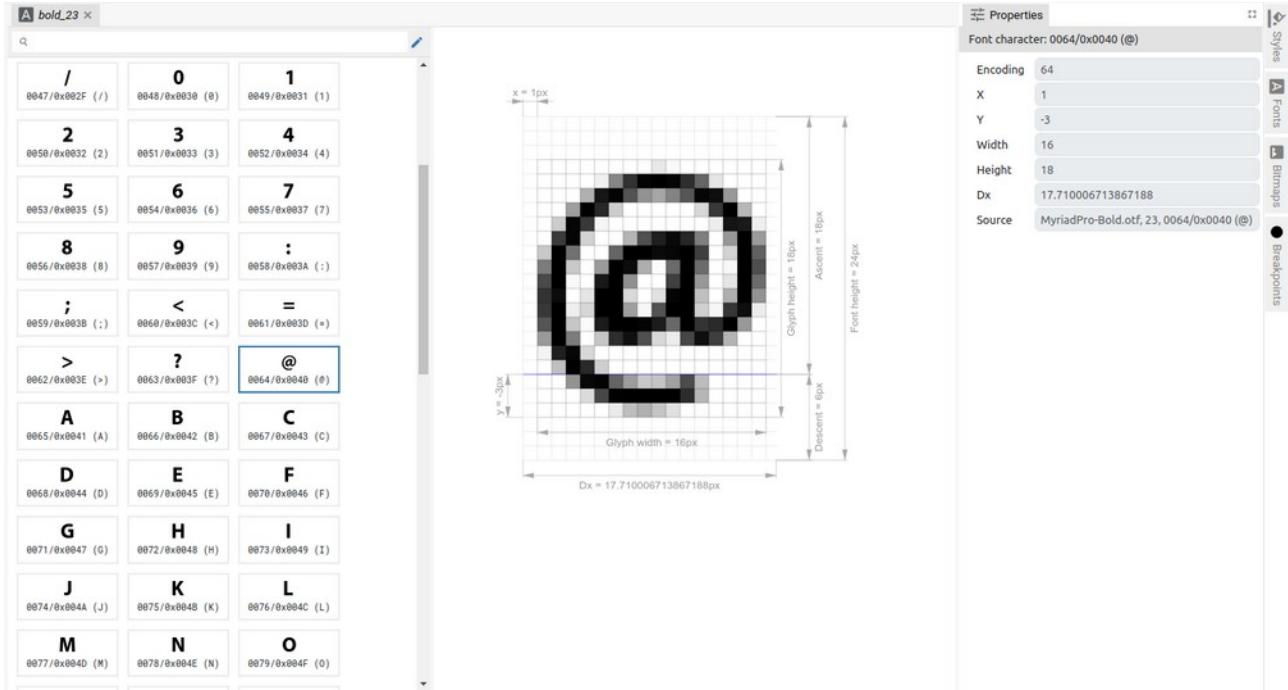


Fig. 132: Font character table (LVGL project)

P11.2.2. Edit characters

Once the font is created, the only thing we can do with the font in terms of editing is to add or delete characters. For this, it is necessary to select the *Add or Remove Characters* option shown in Fig. 133.



Fig. 133: Add or Remove characters option (LVGL project)

The dialog shown in Fig. 134 through which it is possible to define the Ranges and/or Symbols of the character we want to have in the font table.

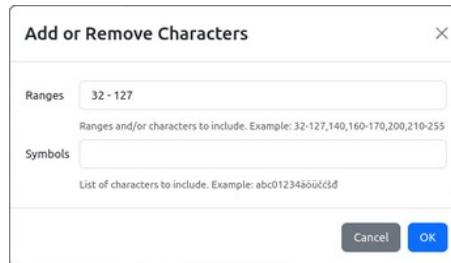


Fig. 134: Font characters editing (LVGL project)

P12. Texts

Projects that support multi-language texts can be found in *New Project Examples* (Fig. 135) and can serve as a starting point for a project that requires the use of multiple languages.

Multi-language is currently supported only in EEZ-GUI and Dashboard projects.

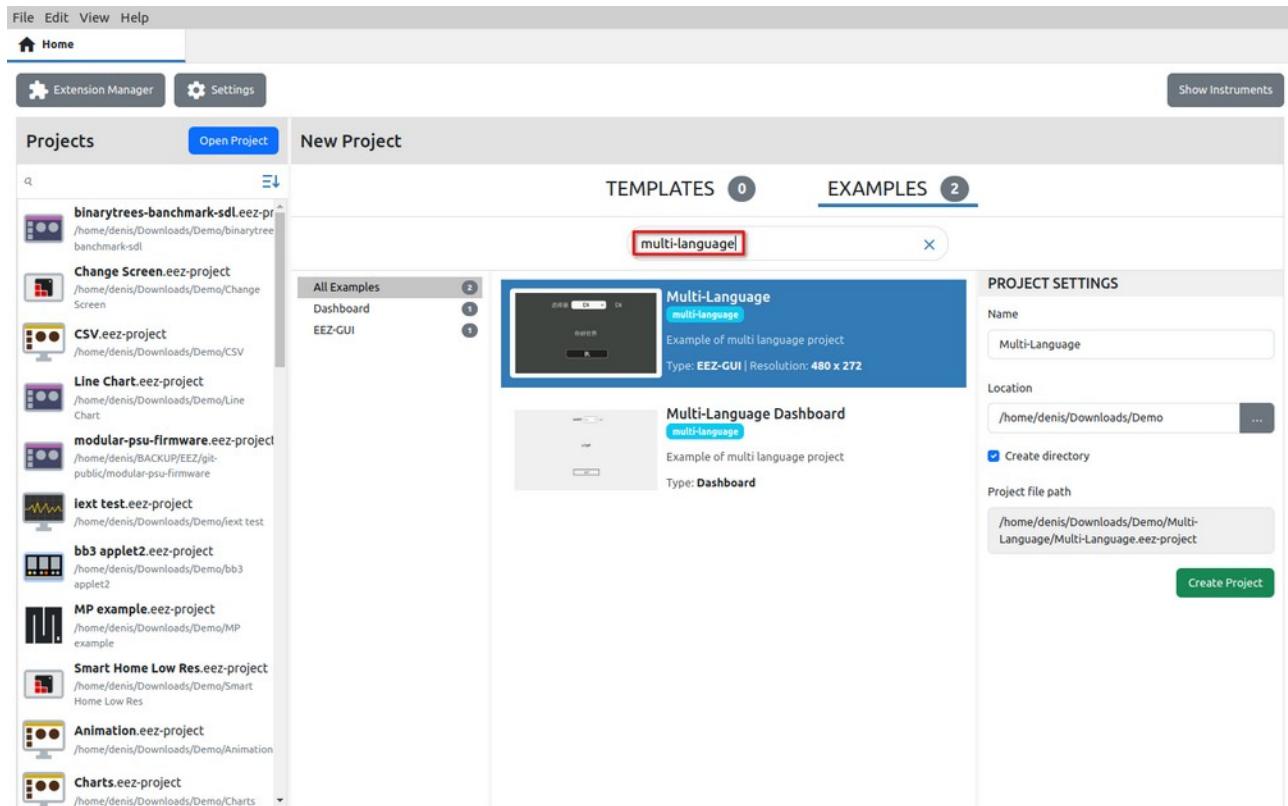


Fig. 135: Multi-language project examples

P12.1. Texts panel

Fig. 136 shows the *Texts* panel for multilingual text editing, which consists of the following three tabs: *Text resources* (1), *Languages* (2) and *Statistics* (3).

Fig. 136: Editing Texts resources

Fig. 137: Completed multi-language translation

Text resources contains a list of IDs of all texts that are multilingual. For each *Resource ID* there should be a translation for all defined *Languages*.

The content of the texts for all defined languages can be seen in *Properties* (4).

There is no limit to the number of languages and text resources in the project. When adding a new *Language*, a dialog will open as shown in Fig. 138, and for adding a new multilingual *Text resource*, a dialog opens as shown in Fig. 139.



Fig. 138: Add new Language



Fig. 139: Add new Text resource

The completeness of the translation can be easily checked thanks to the progress bars for each *Resource ID* and *Language*. The overall translation statistics are displayed in the *Statistics* tab (3). Fig. 137 shows an example when all texts are translated.

The *SelectLanguage* action is used at runtime to select the active language.

There are two methods for using localized text in expressions:

- Using the special literal `T"<text resource ID>"`. For example `T"Hello, world!"` where "Hello, world!" one of the IDs in the *Text Resources* tab.
- Using the function `Flow.translate("<text resource ID>")`. For example, `Flow.translate("Hello, world!")`

Since it is simpler, it is recommended to use the first method.

If there is currently no translation for a language, then the text resource ID itself will be used, so it is convenient for that ID to be the same as the translation for one of the languages, for example in English.

P12.2. XLIFF Import/export

XLIFF (XML Localization Interchange File Format) is an XML-based format created to standardize the way localizable data are passed between and among tools during a localization process and a common format for CAT (Computer-Aided Translation) tool exchange.

By using this format, it is possible for a professional translator to prepare translations in the tool with which he/she is familiar and deliver the translations to the developer, who will insert them into the EEZ Studio project.

Options for Import and Export text resources in XLIFF format can be found in the *Language* panel.

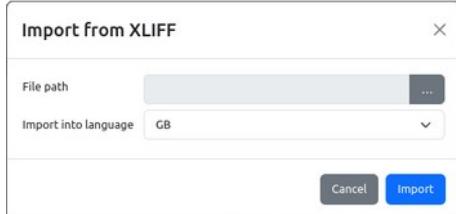


Fig. 140: Text import from XLIFF file



Fig. 141: Text export to XLIFF file

During import, a dialog opens as shown in Fig. 140 where you have to select the *File path* to XLIFF file

and the *Language* into which the text strings will be imported (combo box with a list of all defined languages).

When exporting (Fig. 141), the *Source language* and *Target language* should be defined, as well as the *XLIFF file format version* (1.2 or 2.0 depending on what the translation tool supports). In Fig. 142 shows the exported XLIFF file in the *Poedit* application (*Source language* is GB, *Target language* is FR).

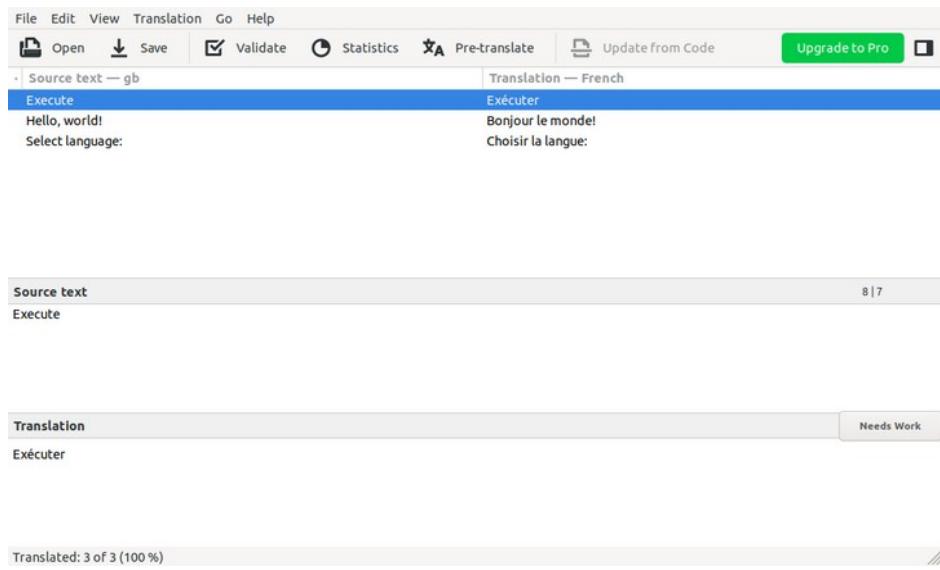


Fig. 142: Exported XLIFF file opened in Poedit

P13. Settings

Project Settings is used to configure the project and the number of parameters and features depends on the project type. Examples of Settings page for *EEZ-GUI* project is shown in Fig. 143.

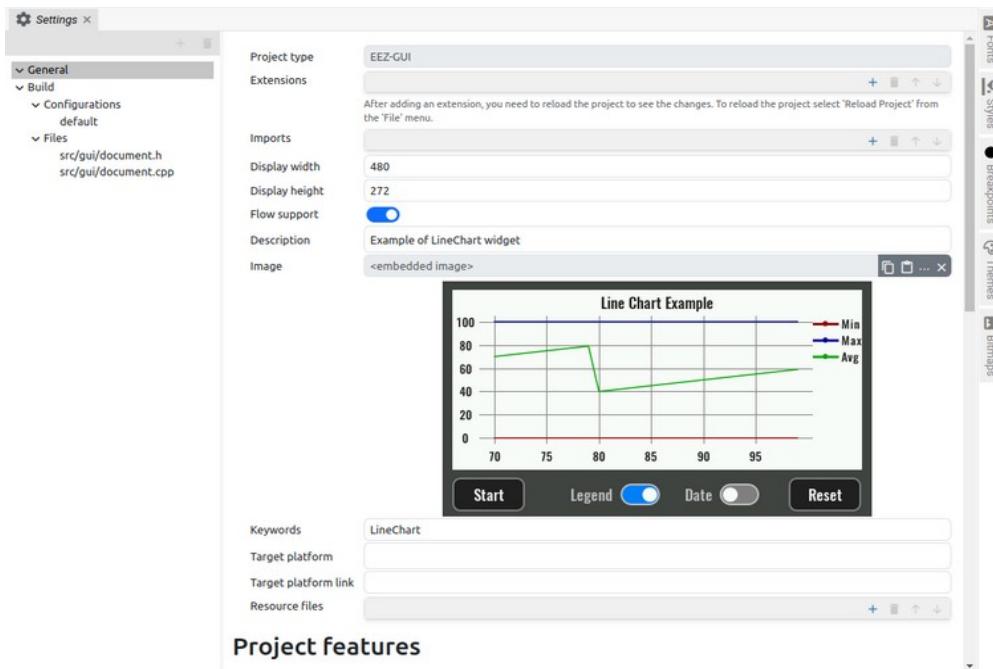


Fig. 143: General Project settings (*EEZ-GUI*)

P13.1. General

Item	Description
<i>Project type</i>	Information about the project type. It is generated when creating a new project and cannot be changed later. Supported project types: <i>Dashboard</i> , <i>EEZ-GUI</i> , <i>LVGL</i> , <i>BB3 MicroPython Script</i> and <i>BB3 Applet</i> (for descriptions see Section P1.1).
<i>Target BB3 firmware (BB3 MicroPython Script only)</i>	Supported versions: <i>1.7.X or older</i> or <i>1.8 or newer</i>
<i>Master project (BB3 Applet and BB3 MicroPython Script only)</i>	This is populated when creating a project with the BB3 firmware project name i.e. <i>modular-psu-firmware.eez-project</i> . It can be replaced with a different name and location if needed.
<i>Extensions</i>	List of extensions used by the project. Extensions can be added, deleted and moved in the order in which they will be loaded (Note that the order of loading is not crucial for code execution).
<i>Import</i>	List of external projects used by the project. More info is needed XX.
<i>Title (Dashboard only)</i>	The name of the standalone application or instrument dashboard.
<i>Icon (Dashboard only)</i>	Icon for standalone applications or instrument dashboard.
<i>Display width (EEZ-GUI and LVGL only)</i>	Page width in pixels.
<i>Display height (EEZ-GUI and LVGL only)</i>	Page height in pixels.
<i>Flow support (EEZ-GUI and LVGL only)</i>	Enable the use of EEZ Flow in the project.
<i>Description</i>	Project description shown in the <i>Examples</i> section.
<i>Image</i>	Project screenshot shown in the <i>Examples</i> section.
<i>Keywords</i>	Project keywords shown in the <i>Examples</i> section.

<i>Target platform</i>	Project target platform shown in the <i>Examples</i> section.
<i>Target platform link</i>	Link to the website of the target platform shown in the <i>Examples</i> section.
<i>Author</i>	Project author shown in the <i>Examples</i> section.
<i>Author link</i>	Link to the website of the project author shown in the <i>Examples</i> section.
<i>Min. studio version</i>	Minimum version of EEZ Studio required to run example project. This version will be compared with the version of running Studio and if it is greater then Example will not be shown.
<i>Resource files</i>	List of external files used by Examples. It can be e.g., a <code>.py</code> file used by the Python example (<i>Charts</i> project), or a <code>.csv</code> file used by the CSV example (<i>CSVProject</i>), etc.
<i>Project features</i>	The number of Features depends on the project type. In Fig. 144 are shown for the EEZ-GUI project.

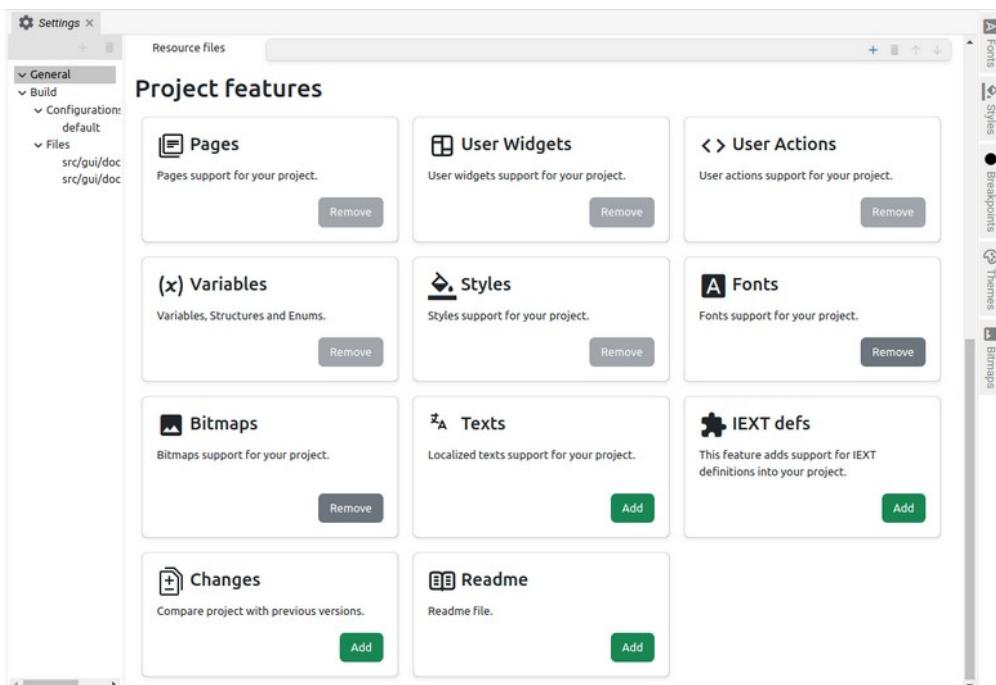


Fig. 144: Project features (EEZ-GUI)

P13.2. Build

The *Build* subsection is only available for *EEZ-GUI* and *LVGL* projects.

Item	Description
<i>Destination folder</i>	The folder in which the build files will be inserted.
<i>LVGL include (LVGL only)</i>	Path to the <code>lvgl.h</code> header file. Normally it is <code>lvgl/lvgl.h</code> , but if it is located somewhere else then it can be specified there.
<i>Generate source code for EEZ Flow engine (eez-framework)</i>	When EEZ Flow is used in LVGL project (i.e. "Flow support" option in General Settings is checked) and this option is checked then Studio will generate in destination folder all the files required to build project on the target platform (i.e. no additional library from EEZ is required). Following files will be additionally generated along with other files defined in Files subsection: <ul style="list-style-type: none"> • <code>eez-flow.h</code> • <code>eez-flow.cpp</code>

These files will be generated only if "Compress flow definition" is checked (see below):

- `eez-flow-lz4.h`
- `eez-flow-lz4.c`

These files will be generated only if `Crypto.sha256` expression function is used in your project:

- `eez-flow-sha256.h`
- `eez-flow-sha256.c`

Since `eez-flow.cpp` is C++ source file you need to enable C++ compilation.

Otherwise, if this option is not checked then user needs to include [eez-framework](#) library manually. This option is checked by default as this is most simple way to enable EEZ Flow in your project.

Following options affects FLASH/SRAM memory usage.

Compress flow definition

When this option is checked then LZ4 library (`eez-flow-lz4.h` and `eez-flow-lz4.c`) will be also included in generated source files. You would want to compress flow definitions if you care about FLASH memory usage, but if you care about SRAM memory usage then you should leave it unchecked. This option will be unchecked by default for the new projects.

Execution queue size

Required execution queue size depends on how complex are flows in your project. This is an advanced option and normally should be left unchanged. Default value is enough for most cases and you can try to decrease this value if you want to spare some SRAM memory. If errors start appearing in the execution of the flow, it means that you have reduced the queue size too much.

Expression evaluator stack size

Required expression evaluator stack size depends on how complex are expressions in your flows. This is an advanced option and normally should be left unchanged. Default value is enough for most cases and you can try to decrease this value if you want to spare some SRAM memory. If errors start appearing in the execution of the flow, it means that you have reduced the queue size too much.

P13.2.1. Configurations

The *Configurations* subsection is only available for *EEZ-GUI* projects.

A project can define multiple build configurations. For example, if we use the same project to build native firmware for the hardware board and for the simulator and we do not want to include in the build files for the hardware board resources that are used only for the simulator and vice versa, we will define two configurations.

For *Page*, *Action*, *SCPI command*, *Shortcut* and *Variable*, we can indicate in which configuration they are used.

The *Used in* property (Fig. 145) is used to define in which configuration the item will be used.

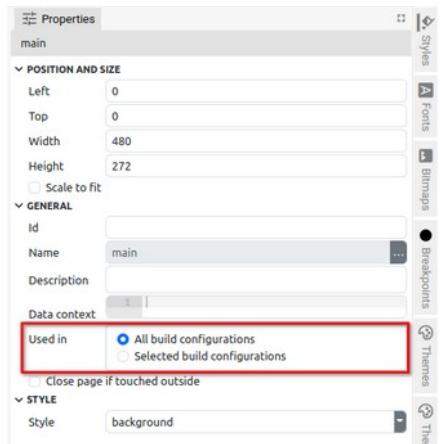


Fig. 145: EEZ-GUI project Used in parameter

Build Configuration parameters are shown in Fig. 146.



Fig. 146: EEZ-GUI build configuration settings

Item	Description
<i>Name</i>	The name of the build configuration.
<i>Description</i>	Description of the build configuration.
<i>Properties</i>	They are used for IEXT to specify additional IEXT options and are defined in JSON format. In the IEXT definition, it is indicated which configuration is used, which is described in Chapter XX.

P13.2.2. Files

The *Files* subsection is only available for *EEZ-GUI* and *LVGL* projects. List of template source files from which source files will be generated. This is all already prepared during the creation of the project from the wizard.

EEZ Studio
Actions

Table of Contents

<i>EEZ Studio Actions</i>	A.1
A1. AddToInstrumentHistory.....	A.7
A2. Animate.....	A.13
A3. CatchError.....	A.15
A4. ClipboardWrite.....	A.17
A5. CloseStream.....	A.19
A6. CollectStream.....	A.21
A7. Comment.....	A.23
A8. Compare.....	A.25
A9. ConnectInstrument.....	A.27
A10. Constant.....	A.29
A11. Counter.....	A.31
A12. CSVParse.....	A.33
A13. CSVStringify.....	A.37
A14. DateNow.....	A.39
A15. Delay.....	A.41
A16. DisconnectInstrument.....	A.43
A17. DynamicCallAction.....	A.45
A18. End.....	A.47
A19. Error.....	A.49
A20. Eval JS.....	A.51
A21. Evaluate.....	A.53
A22. ExecuteCommand.....	A.55
A23. FileAppend.....	A.57
A24. FileOpenDialog.....	A.59
A25. FileRead.....	A.61
A26. FileSaveDialog.....	A.63
A27. FileWrite.....	A.65
A28. FocusWidget.....	A.67
A29. GetInstrument.....	A.69
A30. GetInstrumentProperties.....	A.73
A31. HTTP.....	A.77
A32. Input.....	A.79
A33. InstrumentRead.....	A.81
A34. InstrumentTerminal.....	A.83
A35. InstrumentWrite.....	A.87
A36. IsTrue.....	A.89
A37. JSONParse.....	A.91
A38. JSONStringify.....	A.95
A39. Label IN.....	A.97
A40. Label OUT.....	A.99
A41. Log.....	A.101
A42. Loop.....	A.103

A43. LVGL.....	A.107
A44. Modbus.....	A.115
A45. MQTTConnect.....	A.119
A46. MQTTDisconnect.....	A.121
A47. MQTTEvent.....	A.123
A48. MQTTInit.....	A.125
A49. MQTTPublish.....	A.127
A50. MQTTSubscribe.....	A.129
A51. MQTTUnsubscribe.....	A.131
A52. NoOp.....	A.133
A53. OnEvent.....	A.135
A54. Output.....	A.137
A55. OverrideStyle.....	A.139
A56. PlayAudio.....	A.141
A57. PrintToPDF.....	A.143
A58. PythonEnd.....	A.145
A59. PythonRun.....	A.147
A60. PythonSendMessage.....	A.151
A61. ReadSetting.....	A.153
A62. Regexp.....	A.155
A63. SCPI.....	A.159
A64. SelectInstrument.....	A.163
A65. SelectLanguage.....	A.165
A66. SerialConnect.....	A.167
A67. SerialDisconnect.....	A.169
A68. SerialInit.....	A.171
A69. SerialListPorts.....	A.173
A70. SerialRead.....	A.175
A71. SerialWrite.....	A.177
A72. SetPageDirection.....	A.179
A73. SetVariable.....	A.181
A74. ShowFileInFolder.....	A.183
A75. ShowKeyboard.....	A.185
A76. ShowKeypad.....	A.189
A77. ShowMessageBox.....	A.193
A78. ShowPage.....	A.197
A79. SortArray.....	A.199
A80. Start.....	A.201
A81. SwitchCase.....	A.203
A82. TabulatorAction.....	A.205
A83. TCPConnect.....	A.207
A84. TCPDisconnect.....	A.209
A85. TCPEvent.....	A.211
A86. TCPListen.....	A.213

A87. TCPWrite.....	A.215
A88. TestAndSet.....	A.217
A89. UDP In.....	A.219
A90. UDP Out.....	A.221
A91. Watch.....	A.223
A92. WriteSetting.....	A.225

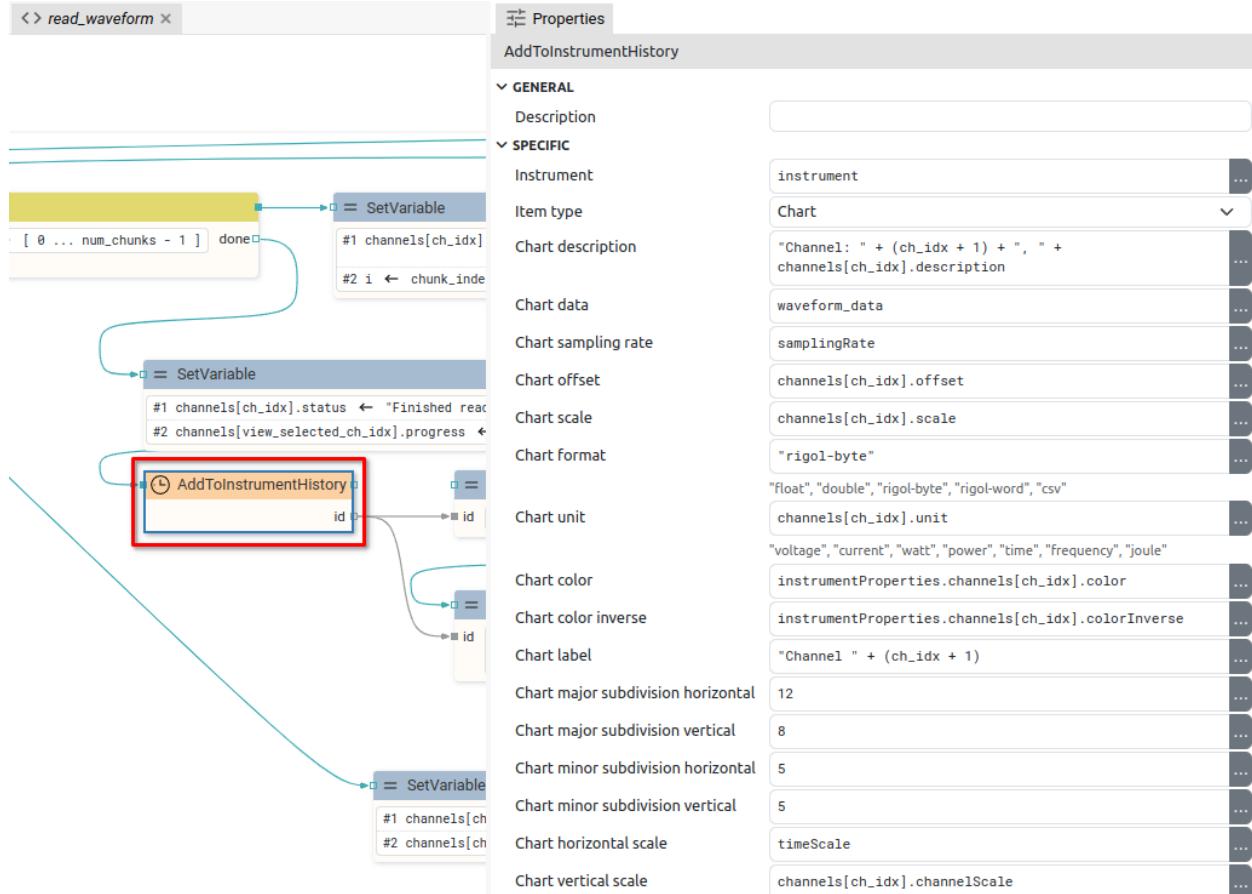
A1. AddToInstrumentHistory



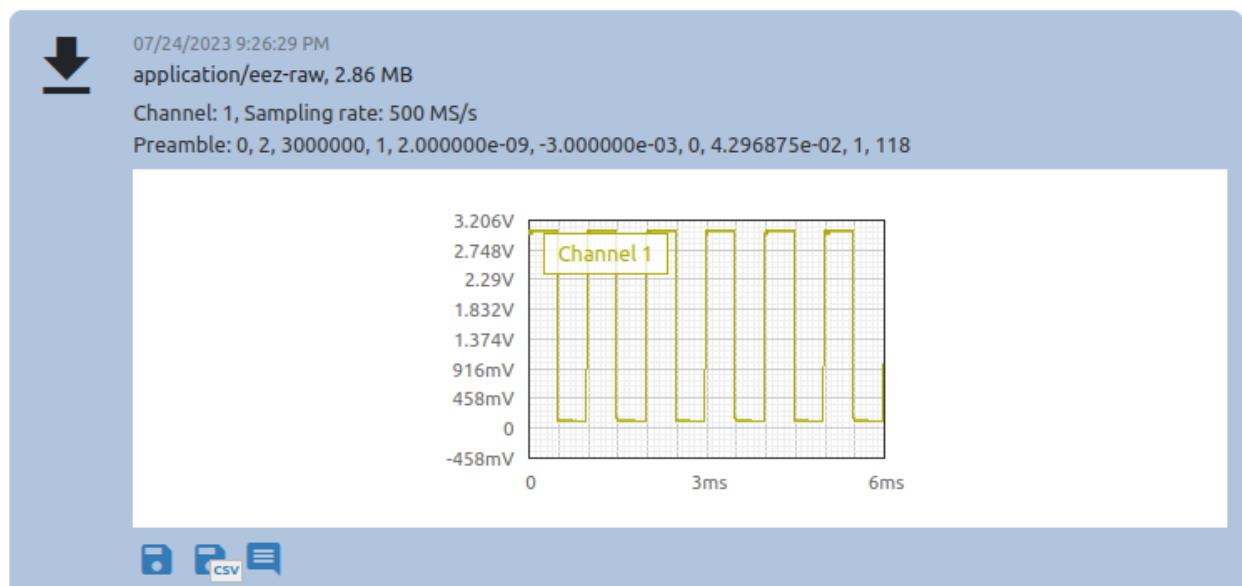
A1.1. Description

It is used to add a new item to the *History* view of the instrument. Currently, only adding chart items or widgets (Tabulator, Plotly or LineChart) are supported.

For example in the *Rigol Waveform Data* example we have this Action:



It is used to add a chart which, after successful addition, will be displayed as follows (example of test signal acquisition):



A1.2. Properties

Specific

A1.2.1. Instrument *EXPRESSION (object:Instrument)*

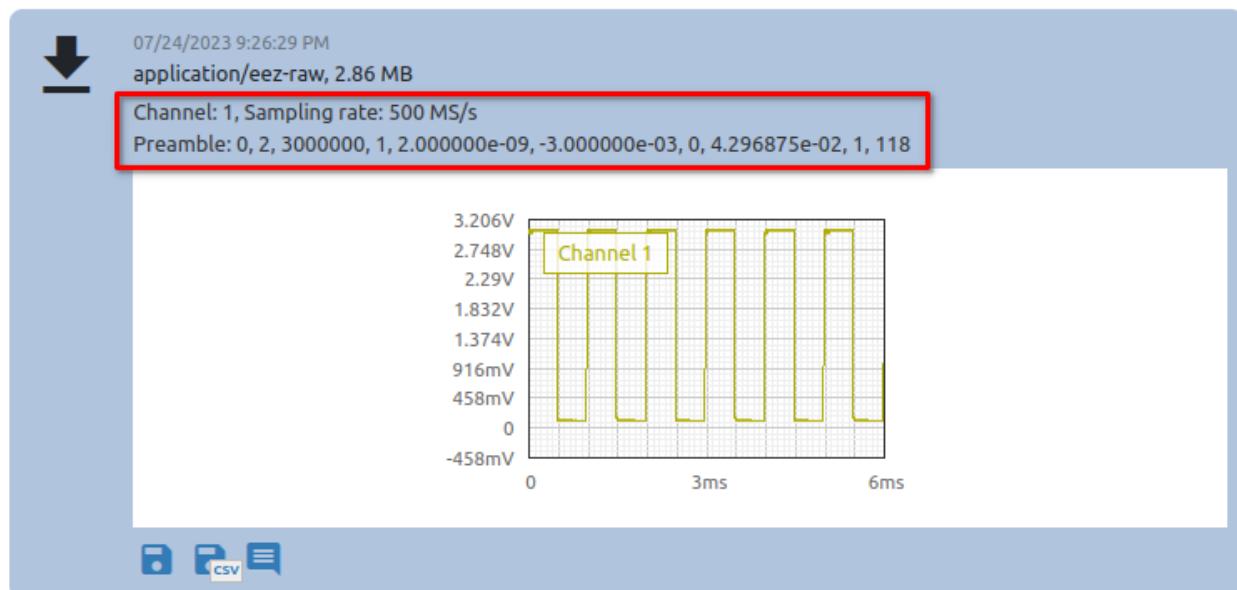
An instrument in whose *History* an item will be added.

A1.2.2. Item type *Enum*

Item type to be added. It can be "Chart" or "Widget".

A1.2.3. Chart description *EXPRESSION (string)*

Description of the chart displayed in the instrument *History*:



This property is only available when *Item type* is Chart.

A1.2.4. Chart data *EXPRESSION (blob)*

A string or blob containing the samples that will be displayed in the chart.
This property is only available when *Item type* is Chart.

A1.2.5. Chart sampling rate *EXPRESSION (float)*

Sampling rate or number of samples per second (SPS).
This property is only available when *Item type* is Chart.

A1.2.6. Chart offset *EXPRESSION (double)*

Offset value used in formula `offset + sample_value * scale` which transforms sample value to sample position on y axis in the chart.

This property is only available when *Item type* is Chart.

A1.2.7. Chart scale *EXPRESSION (double)*

When displaying samples, the formula `offset + sample_value * scale` is used.
This property is only available when *Item type* is Chart.

A1.2.8. Chart format *EXPRESSION (string)*

Format from `Chart` data. Possible values:

- `"float"` – "Chart data" must be a blob containing 32-bit, little-endian float numbers.
- `"double"` – "Chart data" must be a blob containing 64-bit, little-endian float numbers.
- `"rigol-byte"` – "Chart data" must be a blob containing 8-bit unsigned integer numbers.
- `"rigol-word"` – "Chart data" must be a blob containing 16-bit unsigned integer numbers.
- `"csv"` – "Chart data" must be a CSV string, the first column is taken.

This property is only available when `Item type` is `Chart`.

A1.2.9. Chart unit *EXPRESSION (integer)*

The unit displayed on the Y-axis. The X-axis is always time.

This property is only available when `Item type` is `Chart`.

A1.2.10. Chart color *EXPRESSION (string)*

The color of the line in the chart if a dark background is selected.

This property is only available when `Item type` is `Chart`.

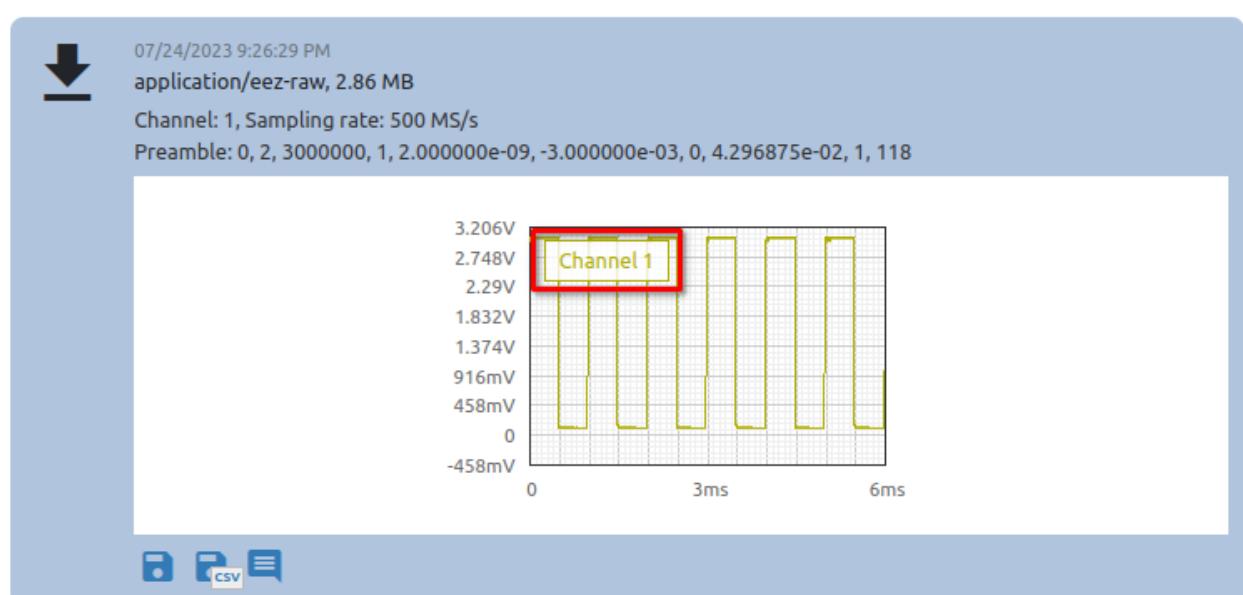
A1.2.11. Chart color inverse *EXPRESSION (string)*

The color of the line in the chart if the light background is selected.

This property is only available when `Item type` is `Chart`.

A1.2.12. Chart label *EXPRESSION (string)*

Chart label:



This property is only available when `Item type` is `Chart`.

A1.2.13. Chart major subdivision horizontal *EXPRESSION (integer)*

Axes lines subdivision:

Dynamic

Fixed

	X axis	Y axis
Major	12	by 8
Minor	5	by 5

Snap to grid

This property is only available when `Item type` is `Chart`.

A1.2.14. Chart major subdivision vertical *EXPRESSION (integer)*

Axes lines subdivision:

Dynamic

Fixed

	X axis	Y axis
Major	12	by 8
Minor	5	by 5

Snap to grid

This property is only available when `Item type` is `Chart`.

A1.2.15. Chart minor subdivision horizontal *EXPRESSION (integer)*

Axes lines subdivision:

Dynamic

Fixed

	X axis	Y axis
Major	12	by 8
Minor	5	by 5

Snap to grid

This property is only available when `Item type` is `Chart`.

A1.2.16. Chart minor subdivision vertical *EXPRESSION (integer)*

[View Options](#) [Rulers](#) [Help](#)

Axes lines subdivision:

Dynamic

Fixed

X axis	Y axis
Major 12	by 8
Minor 5	by 5

Snap to grid

This property is only available when `Item type` is `Chart`.

A1.2.17. Chart horizontal scale *EXPRESSION (double)*

The number that defines the X-axis zoom factor in the default chart view.

This property is only available when `Item type` is `Chart`.

A1.2.18. Chart vertical scale *EXPRESSION (double)*

The number that defines the Y-axis zoom factor in the default chart view.

This property is only available when `Item type` is `Chart`.

A1.2.19. Widget *EXPRESSION (widget)*

Reference to the Tabulator, Plotly or LineChart widget. See `Output widget handle` property to find out how to obtain this reference.

This property is only available when `Item type` is `Widget`.

General

A1.2.20. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A1.2.21. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A1.2.22. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

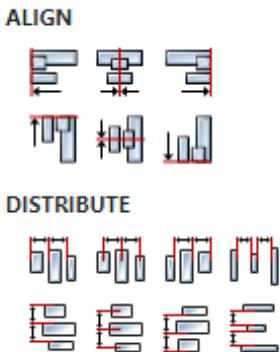
A1.2.23. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A1.2.24. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A1.3. Inputs

A1.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A1.4. Outputs

A1.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A1.4.2. id DATA(string) / OPTIONAL

ID of the added history item. We can, for example, use this data in the `Chart` Widget to display the chart history item inside the dashboard.

A1.5. Examples

- Rigol Waveform Data

A2. Animate



A2.1. Description

If this action is used inside Page or User Widget, it will move the position of the animation timeline from one position (`From` property) to another (`To` property) with given speed (`Speed` property).

If we want to instantly jump to a certain position (`To` property), then we should set the Speed to `0` - in that case the `From` property value doesn't matter (it can be set to the same value as `To` property).

The expression `Flow.pageTimelinePosition()` can be used for the `From` property and in that case the animation will start from the current position.

A2.2. Properties

Specific

A2.2.1. From *EXPRESSION (float)*

Start position set in seconds.

A2.2.2. To *EXPRESSION (float)*

End position set in seconds.

A2.2.3. Speed *EXPRESSION (float)*

Determines the duration of the animation. If set to `1` then the animation will last `From - To` seconds. If we want a twice as fast animation then it should be set to `2`, and if we want a twice slower animation then it should be set to `0.5`.

If we want the animation to last a specific time `T` then the formula $T / (From - To)$ can be used, e.g. if `T` is equal to `0.5` seconds, From `1` seconds and To `3` seconds, then `0.5 / (3 - 1)` should be set for speed, i.e. `0.25`.

If it is set to `0` then it will immediately jump to the `To` position during execution.

General

A2.2.4. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A2.2.5. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A2.2.6. Outputs *Array*

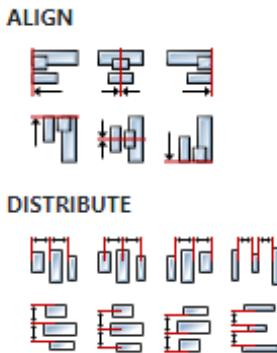
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

A2.2.7. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size**A2.2.8. Align and distribute Any**

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**A2.3. Inputs****A2.3.1. seqin SEQ / OPTIONAL**

A standard sequence input.

A2.4. Outputs**A2.4.1. seqout SEQ / OPTIONAL**

A standard sequence output. It is activated when the animation is finished, ie. when the `To` position was reached.

A2.5. Examples

- *Animation*
- *sld-eez-flow-demo*

A3. CatchError



A3.1. Description

This Action catches all errors that occurred within the Flow in which it is located, or within any of the Child flows that were created by its execution (for example, a Child flow is created when a User action is called).

A3.2. Properties

General

A3.2.1. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A3.2.2. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A3.2.3. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

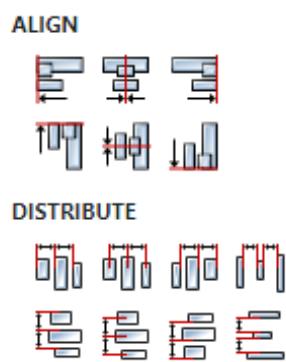
A3.2.4. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A3.2.5. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A3.3. Outputs

A3.3.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

A3.3.2. Message *DATA(string) / MANDATORY*

The output through which the description of the caught error is sent.

A4. ClipboardWrite



A4.1. Description

Writes data specified through the `data` property to the clipboard, which can be text or image.

A4.2. Properties

Specific

A4.2.1. Data *EXPRESSION (any)*

Data that is written to the clipboard. It can be of type `string` or `blob`. If it is of type 'blob', then it is assumed that it is an image.

General

A4.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A4.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A4.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

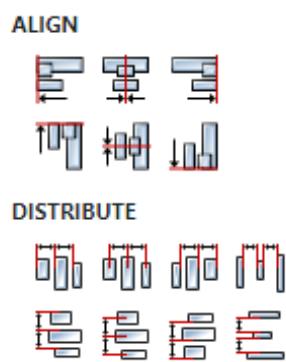
A4.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A4.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A4.3. Inputs

A4.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A4.4. Outputs

A4.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A5. CloseStream



A5.1. Description

Closes given stream. After this no new content will be received through this stream.

A5.2. Properties

Specific

A5.2.1. Stream *EXPRESSION (any)*

Stream to be closed.

General

A5.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A5.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A5.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

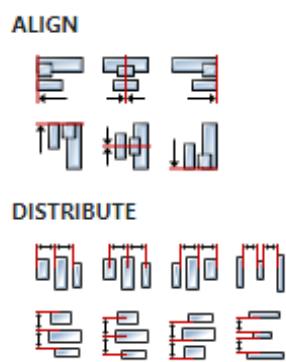
A5.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A5.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A5.3. Inputs

A5.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A5.4. Outputs

A5.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A6. CollectStream



A6.1. Description

Concatenates a stream into a string. As data from the stream comes in chunks, they are concatenated into a string and sent to the data output. During the stream lifetime, this Action can repeatedly send the currently collected string through `data`. Flow execution continues through the `seqout` output when the stream is closed.

A6.2. Properties

Specific

A6.2.1. Stream *EXPRESSION (any)*

A stream whose content will be concatenated into a string.

General

A6.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A6.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A6.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

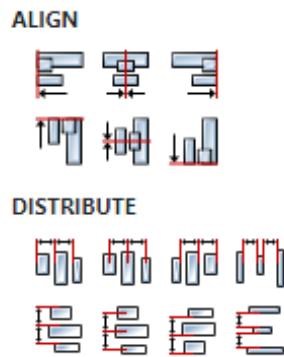
A6.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A6.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A6.3. Inputs

A6.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A6.4. Outputs

A6.4.1. seqout SEQ / OPTIONAL

A standard sequence output. Flow execution continues through this output after the stream is closed.

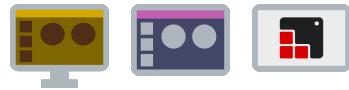
A6.4.2. data DATA(string) / MANDATORY

The concatenated string is sent through this output. During the stream lifetime, a string can be sent several times, which will contain all the data collected until then (i.e. the string will grow over time as new data arrives).

A6.5. Examples

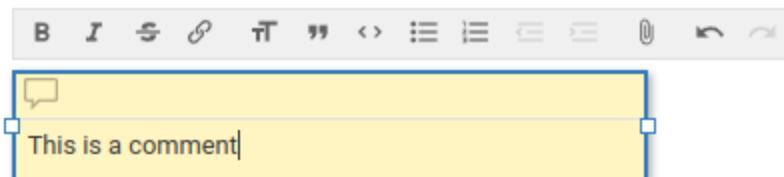
- *RegEx Stream*

A7. Comment



A7.1. Description

This Action has no effect on the Flow execution, but only serves to add comments to the Flow.



A8. Compare



A8.1. Description

Compares expressions depending on the operator and if the result is `true` Flow execution continues through `True` output, otherwise `False` output is used.

A8.2. Properties

Specific

A8.2.1. A *EXPRESSION (any)*

Expression on the left side of the comparison.

A8.2.2. B *EXPRESSION (any)*

Expression on the right side of the comparison. It is not used if the operator is `NOT`.

A8.2.3. C *EXPRESSION (any)*

This expression is used only in the case of the `BETWEEN` operator, then it is checked whether `A >= B` and `A <= C`.

A8.2.4. Operator *Enum*

It is possible to use one of the following operators:

- `=` – A is equal to B, i.e. `A == B`
- `<` – A is less than B, i.e. `A < B`
- `>` – A is greater than B, i.e. `A > B`
- `<=` – A is less or equal to B, i.e. `A <= B`
- `>=` – A is greater or equal to B, i.e. `A >= B`
- `<>` – A is different then B, i.e. `A != B`
- `NOT` – A is not true, i.e. `!A`
- `AND` – both A and B are true, i.e. `A && B`
- `OR` – either A or B is true, i.e. `A || B`
- `XOR` – either A or B is true, but not both, `A ^^ B`
- `BETWEEN` – A is between B and C, i.e. A is greater then or equal to B and A is less then or equal to C, i.e. `A >= B AND A <= C`

General

A8.2.5. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A8.2.6. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A8.2.7. Outputs *Array*

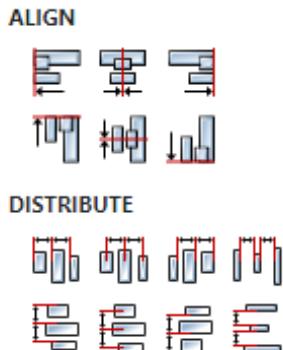
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A8.2.8. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size**A8.2.9. Align and distribute** *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**A8.3. Inputs****A8.3.1. seqin** *SEQ / OPTIONAL*

A standard sequence input.

A8.4. Outputs**A8.4.1. seqout** *SEQ / OPTIONAL*

A standard sequence output.

A8.4.2. True *SEQ / OPTIONAL*

Output that will be used to continue execution of the Flow if the value of the expression is `true`.

A8.4.3. False *SEQ / OPTIONAL*

Output that will be used to continue execution of the Flow if the value of the expression is `false`.

A9. ConnectInstrument



A9.1. Description

Initiates asynchronous connection to the instrument, i.e. the Action will not wait for us to disconnect from the instrument before exiting to `seqout`, but exits immediately. We can check whether we are connected or not with `instrument_variable.isConnected`. For example we can monitor this expression within the *Watch* Action in order to catch the moment when connection to the instrument occurred to start sending SCPI commands.

A9.2. Properties

Specific

A9.2.1. Instrument *EXPRESSION (object:Instrument)*

Instrument object to connect to.

General

A9.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A9.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A9.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

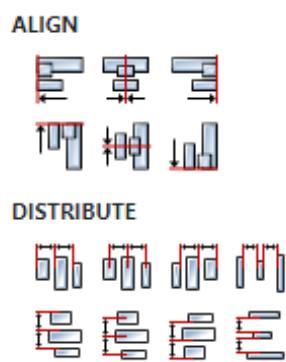
A9.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A9.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A9.3. Inputs

A9.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A9.4. Outputs

A9.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A10. Constant



A10.1. Description

Passes the set constant through the `value` data output.

A10.2. Properties

Specific

A10.2.1. Value *EXPRESSION (string)*

Expression whose result is sent to `value` output. This expression must not use variables. Some examples:

- `"string"`
- `42`
- `3.14159265`
- `true`
- `Math.sin(0.5)`

General

A10.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A10.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A10.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A10.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

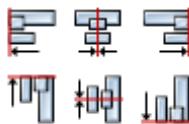
Position and size

A10.2.6. Align and distribute *Any*

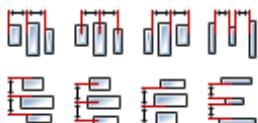
Alignment icons and component distribution. Alignment icons appear when two or more compo-

nents are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A10.3. Inputs

A10.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A10.4. Outputs

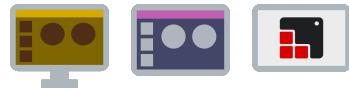
A10.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

A10.4.2. value *DATA(any) / MANDATORY*

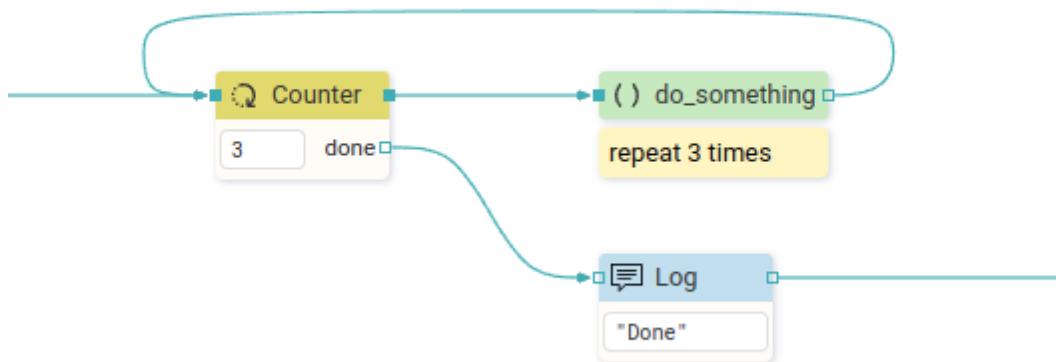
Data output through which the set constant is passed.

A11. Counter



A11.1. Description

Used to execute a specific part of the Flow a given number of times.



A11.2. Properties

Specific

A11.2.1. Count value *EXPRESSION (integer)*

Expression that defines the number of repetitions in the loop.

General

A11.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A11.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A11.2.4. Outputs *Array*

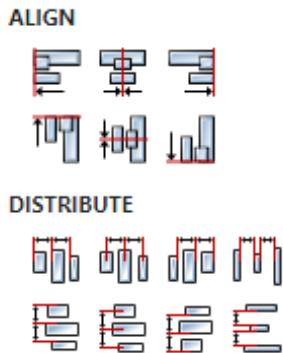
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A11.2.5. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size**A11.2.6. Align and distribute Any**

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**A11.3. Inputs****A11.3.1. seqin SEQ / MANDATORY**

A standard sequence input.

A11.4. Outputs**A11.4.1. seqout SEQ / MANDATORY**

Flow execution continues through this output until the given number of repetitions has been completed.

A11.4.2. done SEQ / OPTIONAL

Flow execution continues through this output when the given number of repetitions has been completed.

A12. CSVParse



A12.1. Description

Parses a CSV string, constructs a value of the set type and sends it through the `result` output.

A12.2. Properties

Specific

A12.2.1. Input *EXPRESSION (string)*

CSV string to be parsed.

A12.2.2. Delimiter *EXPRESSION (string)*

Defines the character used to delimit fields within a CSV record. The default delimiter is `", "`.

A12.2.3. From *EXPRESSION (integer)*

Defines the starting record to be processed. Counting is 1-based, i.e. for the first record it is necessary to set 1 (not 0).

A12.2.4. To *EXPRESSION (integer)*

Defines the last record to be processed. Counting is 1-based, i.e. for the 5th record it is necessary to set 5 (not 4).

General

A12.2.5. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A12.2.6. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A12.2.7. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

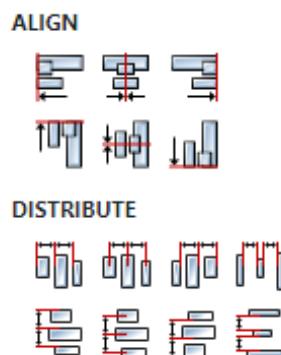
A12.2.8. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A12.2.9. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A12.3. Inputs

A12.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A12.3.2. text DATA(string) / MANDATORY

The input through which the CSV string to be parsed is received. This input can be deleted (we delete it in the Flow - Inputs list) if it is not needed, i.e. if we want to parse a string obtained by evaluating an arbitrary expression set through `Input` property.

A12.4. Outputs

A12.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A12.4.2. result DATA(any) / MANDATORY

Data output to which the constructed value is sent. The type of that value must be specified - this should be done in the Flow - Outputs section:

Inputs	
Name	Type
text	string

Outputs	
Name	Type
result	array:struct:CountryCity

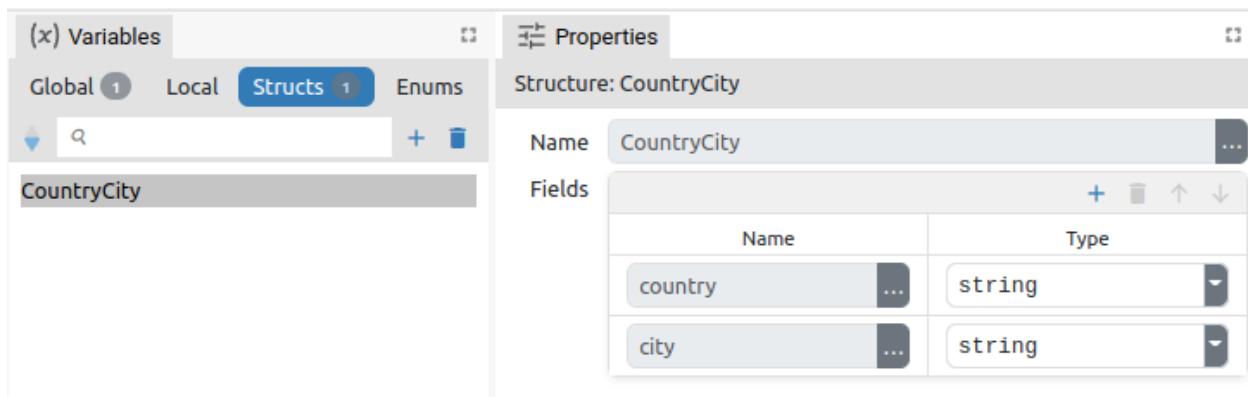
In the CSV example mentioned below, we have a CSV string that looks like this:

```
[
    {
        "country": "Afghanistan",
        "city": "Kabul"
    },
    {
        "country": "Albania",
        "city": "Tirana"
    },
    {
        "country": "Algeria",
        "city": "Alger"
    },
    ...
]
```

The constructed value returned by this Action should be of type `array:CountryCity`, where `CountryCity` is a structure that has two fields (the name of the structure `CountryCity` is arbitrarily chosen by the developer):

- `country`, whose type is `string`
- `city`, whose type is `string`

The definition of that structure looks like this in the Project editor:



A12.5. Examples

- CSV

A13. CSVStringify



A13.1. Description

Converts the Flow value to a CSV string and sends it to the `result` output.

A13.2. Properties

Specific

A13.2.1. Input *EXPRESSION (any)*

Flow value that will be converted into a CSV string.

A13.2.2. Delimiter *EXPRESSION (string)*

Defines the character used to delimit fields within a CSV record. The default delimiter is `", "`.

A13.2.3. Header *EXPRESSION (boolean)*

If it is set to `True`, the first record will contain the names of the columns.

A13.2.4. Quoted *EXPRESSION (boolean)*

If it is set to `True`, all non-empty fields will be quoted even if there are no characters that require quoting.

General

A13.2.5. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A13.2.6. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A13.2.7. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

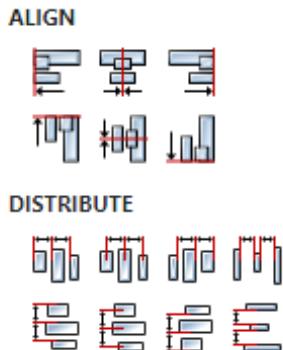
A13.2.8. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A13.2.9. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A13.3. Inputs

A13.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A13.3.2. input DATA(string) / MANDATORY

The Flow value to be converted into a CSV string is received through this Input. This Input can be deleted (we delete it in the Flow - inputs list) if it is not needed, i.e. if we want to parse the string obtained by evaluating an arbitrary expression set through the `Input` property.

A13.4. Outputs

A13.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

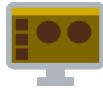
A13.4.2. result DATA(string) / MANDATORY

The constructed CSV string is sent through this output.

A13.5. Examples

- CSV

A14. DateNow



A14.1. Description

Passes the current time (data type is `Date`) through the `value` data output.

A14.2. Properties

General

A14.2.1. Description `String`

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A14.2.2. Inputs `Array`

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A14.2.3. Outputs `Array`

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A14.2.4. Catch error `Boolean`

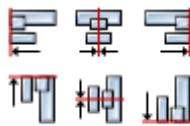
If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

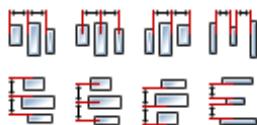
A14.2.5. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A14.3. Inputs

A14.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A14.4. Outputs

A14.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A14.4.2. value DATA(date) / MANDATORY

Data output through which the current time is passed.

A15. Delay



A15.1. Description

This Action is used when we want to insert a pause in Flow execution.

A15.2. Properties

Specific

A15.2.1.Milliseconds *EXPRESSION (integer)*

Pause duration in milliseconds before Flow execution resumes through sequential output `seqout`.

General

A15.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A15.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A15.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

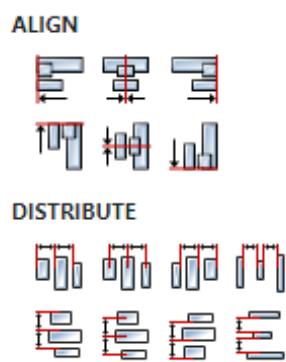
A15.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A15.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A15.3. Inputs

A15.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A15.4. Outputs

A15.4.1. seqout SEQ / MANDATORY

A standard sequence output.

A16. DisconnectInstrument



A16.1. Description

Initiates asynchronous disconnection from the instrument, i.e. the Action will not wait for us to disconnect from the instrument before exiting to `seqout`, but exits immediately. We can check whether we are disconnected or not with `instrument_variable.isConnected`. For example we can monitor this expression within the *Watch* Action in order to catch the moment when disconnection from the instrument occurred.

A16.2. Properties

Specific

A16.2.1. Instrument *EXPRESSION (object:Instrument)*

Instrument object to disconnect from.

General

A16.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A16.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A16.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

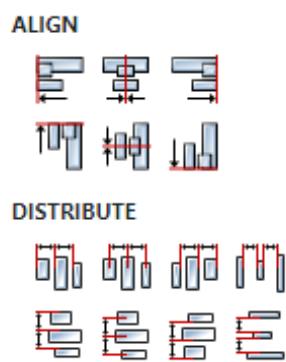
A16.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A16.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A16.3. Inputs

A16.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A16.4. Outputs

A16.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A17. DynamicCallAction



A17.1. Description

Executes a User action whose name is not known in advance, i.e. it is determined during Flow execution, for example its name can come from a variable. Such a User action must not have inputs and outputs, but only *Start* and *End* Actions.

A17.2. Properties

Specific

A17.2.1. Action *EXPRESSION (string)*

The name of the User action to be executed, obtained during Flow execution by evaluating this expression.

General

A17.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A17.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A17.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

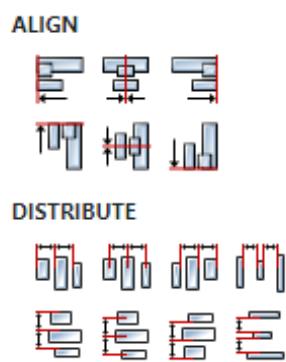
A17.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A17.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A17.3. Inputs

A17.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A17.4. Outputs

A17.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A18. End



A18.1. Description

It is used to terminate the execution of a Flow.

If it is inside the page, it means the end of the application execution. If it is a *Dashboard* project that is executed within the project editor, this means switching from *Run* mode to *Edit* mode. If it is a *Dashboard* running on the instrument, the execution will be interrupted and a *Start* button will appear with which the *Dashboard* can be restarted. If it is *Dashboard* as a standalone application then the application will be closed.

If it is used within a User action, it means the end of the execution of the User action and the activation of the standard sequence line at the point where the User action was called.

This Action has no effect if it is inside a User widget in Flow.

A18.2. Properties

General

A18.2.1. Description String

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A18.2.2. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A18.2.3. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

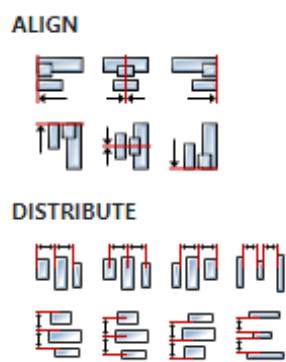
A18.2.4. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A18.2.5. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



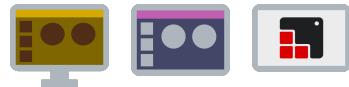
A18.3. Inputs

A18.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A18.4. Outputs

A19. Error



A19.1. Description

This Action throws an error that can then be caught via the *CatchError* action within the same flow in which this action is located, or within its parent Flow, i.e. of any ancestors Flow.

A19.2. Properties

Specific

A19.2.1. Message *EXPRESSION (string)*

A text message describing the type of error, this message will be received by the *CatchError* Action.

General

A19.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A19.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A19.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

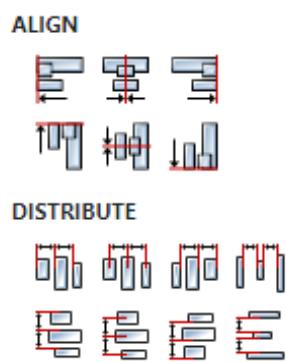
A19.2.5. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A19.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A19.3. Inputs

A19.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A20. Eval JS



A20.1. Description

It evaluates a JavaScript expression and sends the result through `result` output.

A20.2. Properties

Specific

A20.2.1. Expression *TEMPLATE LITERAL*

The JavaScript expression to be evaluated. EEZ Flow expression written inside curly brackets can be inserted in several places within the expression. For example in the JavaScript expression `Math.random() * {num_items}`, this `{num_items}` is a Flow expression, i.e. it takes the value of the `num_items` variable that comes from the Flow before handing it off to JavaScript to calculate the complete expression.

General

A20.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A20.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A20.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

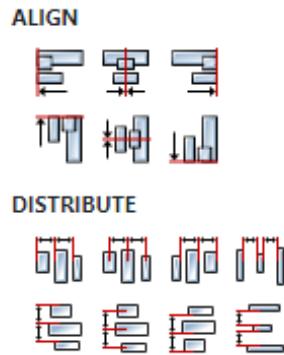
A20.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A20.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A20.3. Inputs

A20.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A20.4. Outputs

A20.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

A20.4.2. result *DATA(any) / MANDATORY*

Output through which the result of JavaScript expression evaluation is sent. By default, Type of the output is set to any, so it is preferable to change it to a specific type.

A21. Evaluate



A21.1. Description

Evaluates the given expression and passes the result to the data output.

A21.2. Properties

Specific

A21.2.1. Expression *EXPRESSION (any)*

Expression to be evaluated.

General

A21.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A21.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A21.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

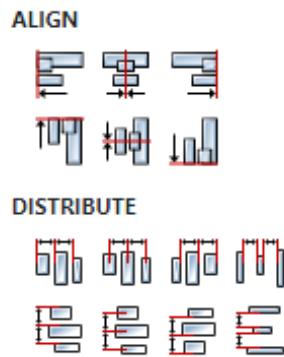
A21.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A21.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A21.3. Inputs

A21.3.1. seqin SEQ / OPTIONAL

A standard sequence input

A21.4. Outputs

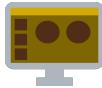
A21.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A21.4.2. result DATA(any) / MANDATORY

Output through which the value of the evaluated expression is passed.

A22. ExecuteCommand



A22.1. Description

The action is used to execute an external command, i.e. program, which can be in the PATH or the full path to the command can be specified.

A22.2. Properties

Specific

A22.2.1. Command *EXPRESSION (string)*

The name of the command, i.e. the full file path to the command to be executed.

A22.2.2. Arguments *EXPRESSION (array:string)*

Array of string arguments that is passed to the command.

General

A22.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A22.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A22.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

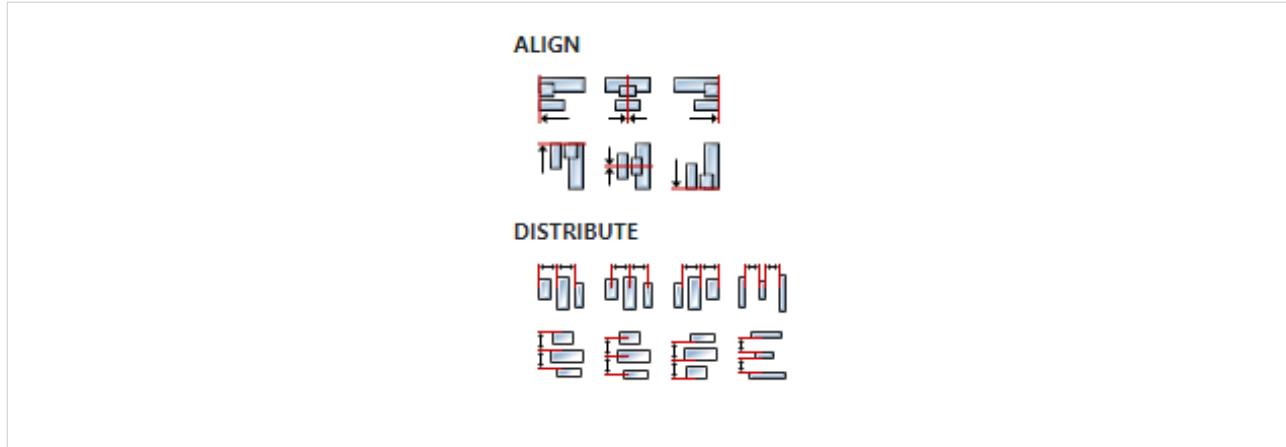
A22.2.6. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A22.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A22.3. Inputs

A22.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A22.4. Outputs

A22.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A22.4.2. stdout DATA(stream) / OPTIONAL

The `stream` value from `stdout` is sent through this output. That `stream` value can be collected into a string with the `CollectStream` Action, redirected to a `Terminal` widget, parsed with the `RegExp` Action, etc.

A22.4.3. stderr DATA(stream) / OPTIONAL

The `stream` value of `stderr` is sent through this output. That `stream` value can be collected into a string with the `CollectStream` Action, redirected to a `Terminal` widget, parsed with the `RegExp` Action, etc.

A22.4.4. finished DATA(integer) / OPTIONAL

If the command completed successfully, Flow execution continues through this output. If an error has occurred, an error is thrown that can be caught if 'Catch error' is enabled.

A22.5. Examples

- *RegExp Stream*

A23. FileAppend



A23.1. Description

Appends data to a file. It will create the file if it doesn't already exist. The data can be a string or a blob.

A23.2. Properties

Specific

A23.2.1. File path *EXPRESSION (string)*

The full path of the file to be written.

A23.2.2. Content *EXPRESSION (string)*

Content to be written. It can be a string or a blob. If the content is a blob, the `encoding` property is ignored.

A23.2.3. Encoding *EXPRESSION (string)*

Encoding type of string content. The following values are allowed: "ascii", "base64", "hex", "ucs2", "ucs-2", "utf16le", "utf-16le", "utf8", "utf-8", "binary" or "latin1".

General

A23.2.4. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A23.2.5. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A23.2.6. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A23.2.7. Catch error *Boolean*

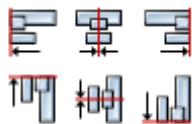
If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

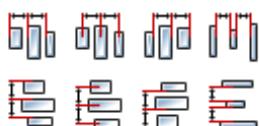
A23.2.8. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A23.3. Inputs

A23.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A23.4. Outputs

A23.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A24. FileOpenDialog



A24.1. Description

Displays the system file open dialog and sends the set file path to the `file_path` output.

A24.2. Properties

Specific

A24.2.1. Filters *EXPRESSION (array:string)*

If we want to limit which types of files appear inside the file open dialog, then we can specify the filter list as `array:string`, for example `["PNG Images|png", "JPG Images|jpg", "GIF Images|gif"]`. This is an optional property and if it is not set then all files will be displayed.

General

A24.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A24.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A24.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

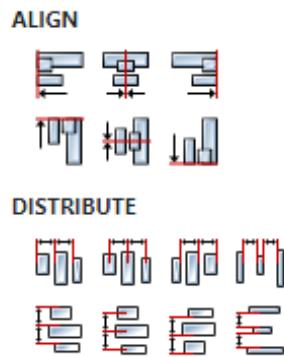
A24.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A24.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A24.3. Inputs

A24.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A24.4. Outputs

A24.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A24.4.2. file_path DATA(string) / MANDATORY

Output to which the set file path is sent.

A25. FileRead



A25.1. Description

Reads the contents of a file as either a string or blob and sends it to the `content` output

A25.2. Properties

Specific

A25.2.1. File path *EXPRESSION (string)*

The full path of the file to be read.

A25.2.2. Encoding *EXPRESSION (string)*

Encoding of the input data. Possible values are: "ascii", "base64", "hex", "ucs2", "ucs-2", "utf16le", "utf-16le", "utf8", "utf-8", "binary" or "latin1".

If encoding is "binary" then the blob value is returned, otherwise the string value is returned.

General

A25.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A25.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A25.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

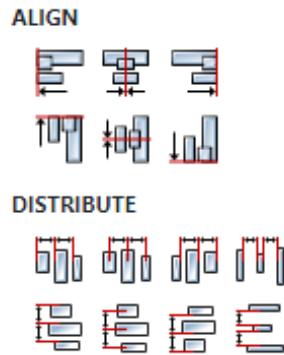
A25.2.6. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A25.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A25.3. Inputs

A25.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A25.4. Outputs

A25.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A25.4.2. content DATA(any) / MANDATORY

The read content of the file is sent through this output.

A25.5. Examples

- JSON
- CSV
- EEZ Chart

A26. FileSaveDialog



A26.1. Description

Displays the system file save dialog and sends the set file path to the `file_path` output.

A26.2. Properties

Specific

A26.2.1. File name *EXPRESSION (string)*

The file name to be used by default.

A26.2.2. Filters *EXPRESSION (array:string)*

If we want to limit which types of files appear inside the file save dialog, then we can specify the filter list as `array:string`, for example `["PNG Images|png", "JPG Images|jpg", "GIF Images|gif"]`. This is an optional property and if it is not set then all files will be displayed.

General

A26.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A26.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A26.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

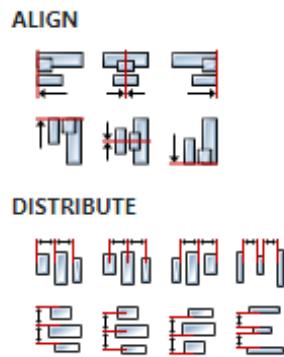
A26.2.6. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A26.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A26.3. Inputs

A26.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A26.4. Outputs

A26.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A26.4.2. file_path DATA(string) / MANDATORY

Output to which the set file path is sent.

A26.5. Examples

- *Screen Capture*

A27. **FileWrite**



A27.1. **Description**

Writes data to a file, replacing the file if it already exists. Data can be a string or a blob.

A27.2. **Properties**

Specific

A27.2.1. **File path** *EXPRESSION (string)*

The full path of the file to be written.

A27.2.2. **Content** *EXPRESSION (string)*

The content to be written can be a string or a blob. If the content is a blob, the `encoding` property is ignored.

A27.2.3. **Encoding** *EXPRESSION (string)*

Encoding of the content. Possible values are: `"ascii"`, `"base64"`, `"hex"`, `"ucs2"`, `"ucs-2"`, `"utf-16le"`, `"utf-16le"`, `"utf8"`, `"utf-8"`, `"binary"` or `"latin1"`.

General

A27.2.4. **Description** *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A27.2.5. **Inputs** *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A27.2.6. **Outputs** *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A27.2.7. **Catch error** *Boolean*

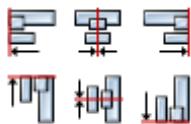
If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

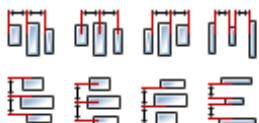
A27.2.8. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A27.3. Inputs

A27.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A27.4. Outputs

A27.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A27.5. Examples

- CSV
- Screen Capture

A28. FocusWidget



A28.1. Description

Puts widget in focus.

A28.2. Properties

Specific

A28.2.1. Widget *EXPRESSION (widget)*

Reference to the a widget. See `Output widget handle` property to find out how to obtain this reference.

General

A28.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A28.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A28.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

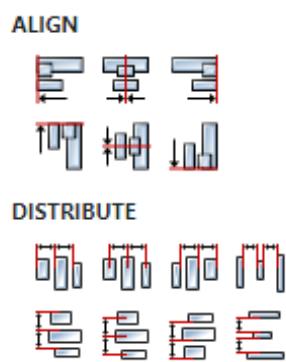
A28.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A28.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A28.3. Inputs

A28.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A28.4. Outputs

A28.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A29. GetInstrument



A29.1. Description

Retrieves an instrument object by its ID. The instrument ID can be found in these two places: Instrument Properties when the instrument is selected on the Instruments Home page and in the header of the Terminal tab of the instrument.

The screenshot shows the Instruments Home page. At the top, there are tabs for 'Instruments' (selected), 'Add Instrument', 'Deleted Instruments', 'History', 'Shortcuts and Groups', 'Notebooks', and a close button. Below the tabs, there is a list of instruments. One instrument, 'BB3 ATE', is highlighted with a blue border and labeled 'Selected instrument'. To the left of the main list, there is a card for 'BB3 #2' which is 'Disconnected'. On the right side, there is a detailed view of the selected instrument 'BB3 ATE'. This view includes an 'Actions' section with 'Open in Tab', 'Open in New Window', and 'Delete' buttons. The 'Properties' section contains the following information:

Instrument	EEZ BB3 STM32
ID	[55] Instrument ID
Label	BB3 ATE
IDN	Envox,EEZ BB3 (STM32),001C003933385107
<input type="checkbox"/> Auto connect	
Connection	
Interface	Ethernet

The screenshot shows the terminal interface for the selected instrument 'BB3 ATE'. The title bar says 'Instrument ID' with the ID '[55] BB3 ATE' highlighted with a red box. Below the title bar, there is a 'Connect' button. On the left, there is a sidebar with 'Start Page', 'Dashboard', and 'Terminal' options. The main area displays a 'LATEST HISTORY EVENT' with the timestamp '04/10/2022 5:51:31 PM' and the message 'DISCONNECTED after 58 seconds'.

Use this Action when you want to access a specific instrument, i.e. you don't want to use a dialog box as a method for selecting an instrument.

A29.2. Properties

Specific

A29.2.1. Instrument ID *EXPRESSION (string)*

The ID of the instrument whose object we want to retrieve.

General

A29.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow**A29.2.3. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A29.2.4. Outputs *Array*

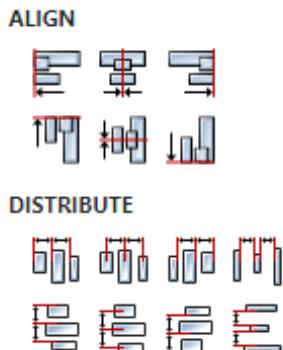
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A29.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size**A29.2.6. Align and distribute** *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**A29.3. Inputs****A29.3.1. seqin** *SEQ / MANDATORY*

A standard sequence input.

A29.4. Outputs

A29.4.1. seqout SEQ / OPTIONAL

A standard sequence output

A29.4.2. instrument DATA(*object:Instrument*) / MANDATORY

The retrieved object is sent to this output.

A30. GetInstrumentProperties



A30.1. Description

Using this Action, we can retrieve the instrument properties that are defined within the IEXT instrument extension.

For example, in the `Rigol Waveform Data` example, we want to retrieve how many channels the instrument has and what color is used for each channel. First, we can look at all the properties of the Rigol DS1000Z instrument:

The screenshot shows the Envoy software interface. On the left, there is a list of installed instruments: Rigol DS1074B (Installed), Rigol DS1074Z (Installed), and Rigol DS1074Z-S (Installed). Each entry includes the instrument name, model, version (1.0.2), and a brief description. The Rigol DS1074Z entry also shows a thumbnail image of the oscilloscope and a 'Change image' link. To the right of the list is a detailed view of the Rigol DS1074Z instrument. It includes a large image of the oscilloscope, a 'Properties' tab (which is selected and highlighted with a red arrow), and a 'Shortcuts' tab. Below the tabs is a code editor containing JSON-like configuration data. At the bottom right of the detailed view is a red 'Uninstall' button.

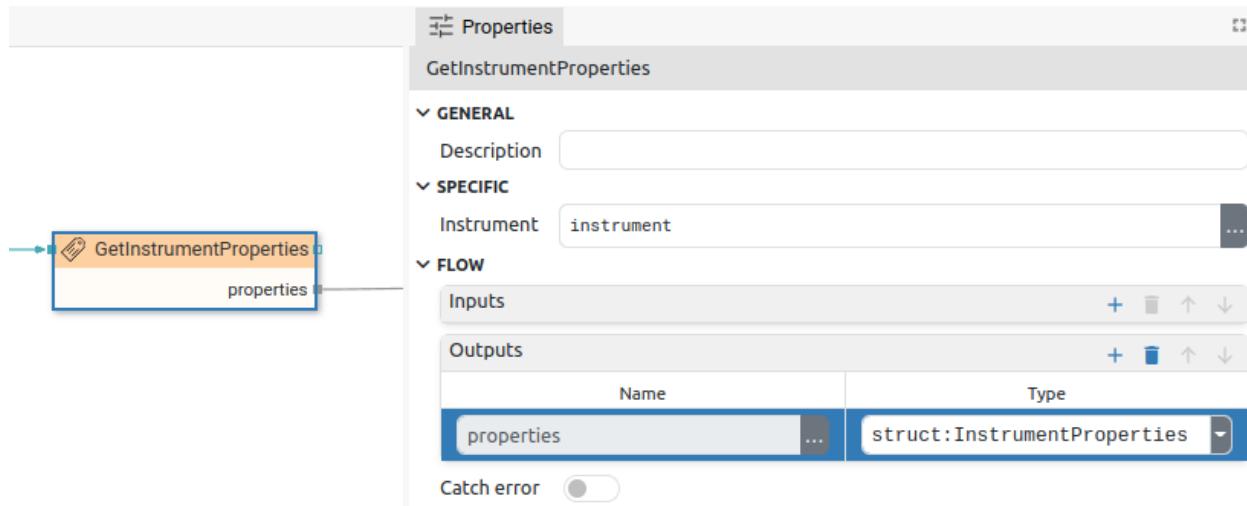
Now it is necessary to define the Flow variable type in which we want to store the properties we are interested in. In this case, we define the type `struct:InstrumentProperties` defined as follows:

The screenshot shows the Envoy Properties editor. On the left, under 'Structs', there is a list of structures: InstrumentPropertiesChannel, InstrumentProperties, and Channel. The 'InstrumentProperties' structure is currently selected. The main panel shows the structure definition: `Structure: InstrumentProperties`, `Name: InstrumentProperties`, and `Fields`. A table lists a single field: `channels` of type `array:struct:InstrumentPropertiesChannel`.

The `InstrumentProperties` structure has one member called `channels`, which is of type `array:InstrumentPropertiesChannel`, and which is defined as follows:

The screenshot shows the Envoy Properties editor. On the left, under 'Structs', there is a list of structures: InstrumentPropertiesChannel, InstrumentProperties, and Channel. The 'InstrumentPropertiesChannel' structure is currently selected. The main panel shows the structure definition: `Structure: InstrumentPropertiesChannel`, `Name: InstrumentPropertiesChannel`, and `Fields`. A table lists two fields: `color` of type `string` and `colorInverse` of type `string`.

And now using this Action in one step we can retrieve information about all channels:



After we have retrieved the properties, we can find out the number of channels with `Array.length(properties.channels)`, and the color, for example, of the 1st channel with: `properties.channels[0].color`.

A30.2. Properties

Specific

A30.2.1. Instrument *EXPRESSION (object:Instrument)*

The instrument whose properties will be retrieved.

General

A30.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A30.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A30.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A30.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that out-

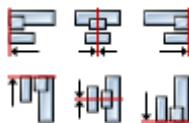
put. The data that will be passed through that output is the textual description of the error.

Position and size

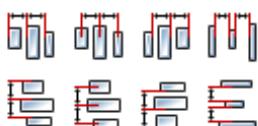
A30.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A30.3. Inputs

A30.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A30.4. Outputs

A30.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A30.4.2. properties DATA(any) / MANDATORY

Retrieved properties are sent to this output.

A30.5. Examples

- Rigol Waveform Data

A31. HTTP



A31.1. Description

Sends HTTP requests and returns the response.

A31.2. Properties

Specific

A31.2.1. Method *Enum*

HTTP methods used: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS, CONNECT or TRACE.

A31.2.2. Url *EXPRESSION (string)*

The url of the request.

A31.2.3. Headers *Array*

List of headers sent to the server. A header name and a string value should be set for each item.

A31.2.4. Body *EXPRESSION (string)*

The body of the message that is sent to the server if the POST, PUT or PATCH method is selected.

General

A31.2.5. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A31.2.6. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A31.2.7. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A31.2.8. Catch error *Boolean*

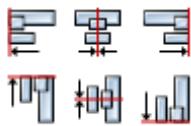
If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

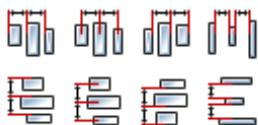
A31.2.9. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A31.3. Inputs

A31.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A31.4. Outputs

A31.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A31.4.2. status DATA(integer) / OPTIONAL

The status code ([link](#)) of the response.

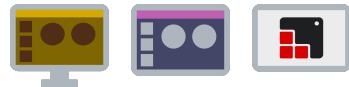
A31.4.3. result DATA(string) / OPTIONAL

Message body of received response.

A31.5. Examples

- Simple HTTP

A32. Input



A32.1. Description

Adds data input to a user action or user widget.

A32.2. Properties

Specific

A32.2.1. Name *String*

Input name.

A32.2.2. Type *String*

Input data type.

General

A32.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A32.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A32.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

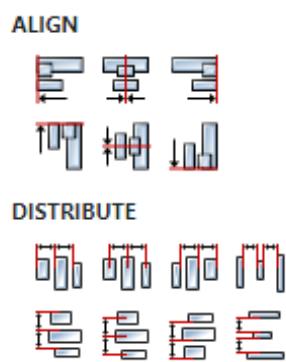
A32.2.6. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A32.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



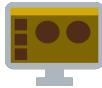
A32.3. Inputs

A32.4. Outputs

A32.4.1. seqout SEQ / MANDATORY

The data received by the caller of the user action is passed through this output.

A33. InstrumentRead



A33.1. Description

Use this to read from the Instrument. Usually, this action is used for the instruments which implements proprietary (non-SCPI) commands protocol. This action will send the read stream to the `data` output.

A33.2. Properties

Specific

A33.2.1. Instrument *EXPRESSION (object:Instrument)*

Instrument object from which we want to read data.

General

A33.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A33.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A33.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

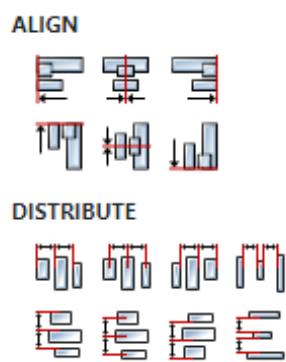
A33.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A33.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A33.3. Inputs

A33.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A33.4. Outputs

A33.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A33.4.2. data DATA(stream) / MANDATORY

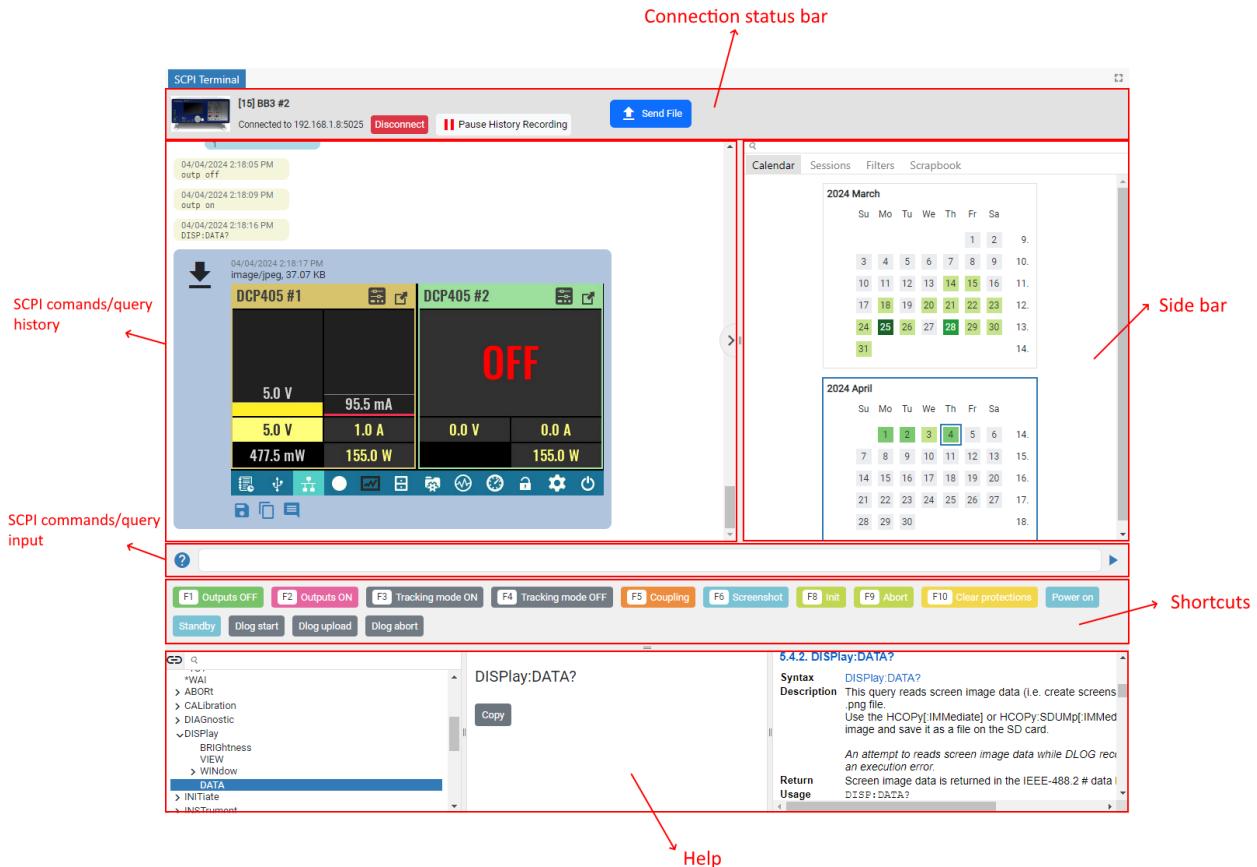
Output to which the read stream is sent.

A34. InstrumentTerminal



A34.1. Description

This widget allows interaction with the instrument. It consists of several parts, some of them can be hidden with associated properties.



A34.2. Properties

Specific

A34.2.1. Instrument *EXPRESSION (object:Instrument)*

Selected instrument object.

A34.2.2. Show connection status bar *EXPRESSION (boolean)*

Show/Hide instrument connection status bar.

A34.2.3. Show shortcuts *EXPRESSION (boolean)*

Show/Hide shortctus panel.

A34.2.4. Show help *EXPRESSION (boolean)*

Show/Hide commands help panel.

A34.2.5. Show side bar *EXPRESSION (boolean)*

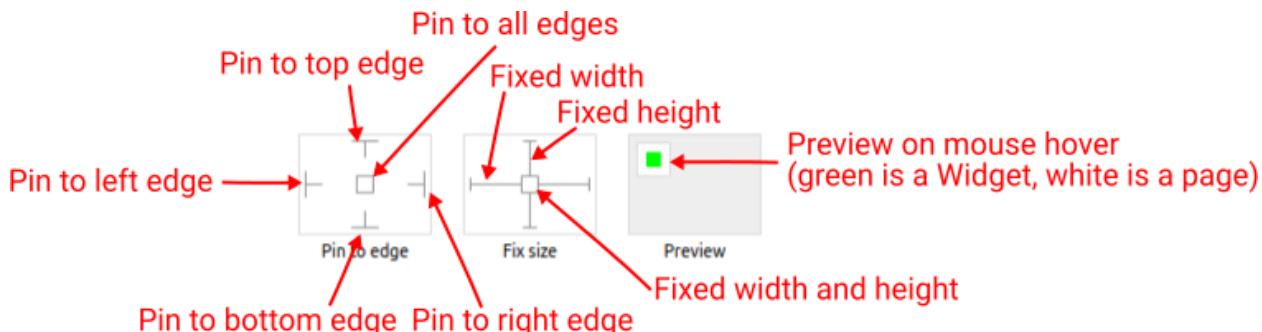
Show/Hide side bar with history search options.

A34.2.6. Visible EXPRESSION (boolean)

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size**A34.2.7. Resizing** Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

A34.2.8. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the **Top**, **Width** and **Height** properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use **+**, **-**, ***** and **/** operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

A34.2.9. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

A34.2.10. Width Number

The width of the component. It is set in pixels.

A34.2.11. Height Number

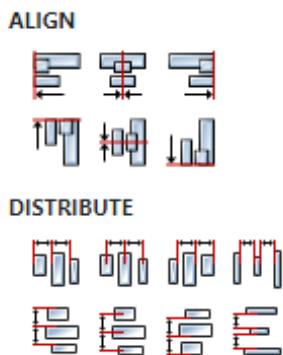
The height of the component. It is set in pixels.

A34.2.12. Absolute pos. String

The absolute position of the component in relation to the page. This property is read-only.

A34.2.13. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A34.2.14. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



Layout

A34.2.15. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Events

A34.2.16. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

A34.2.17. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the

flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

A34.2.18. Inputs `Array`

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A34.2.19. Outputs `Array`

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

A34.2.20. Catch error `Boolean`

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

A34.3. Examples

- BB3 SCPI Terminal and Dashboard

A35. InstrumentWrite



A35.1. Description

Sends a string to the instrument. Usually, this action is used for the instruments which implements proprietary (non-SCPI) commands protocol.

A35.2. Properties

Specific

A35.2.1. Instrument *EXPRESSION (object:Instrument)*

Instrument object in which we want to write a string.

A35.2.2. Data *EXPRESSION (string)*

The string that is sent to the instrument.

General

A35.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A35.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A35.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

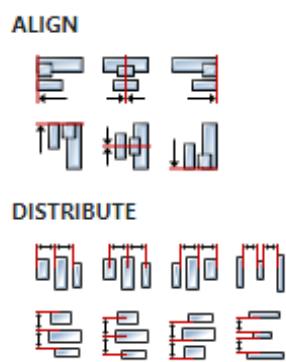
A35.2.6. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A35.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A35.3. Inputs

A35.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A35.4. Outputs

A35.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A36. IsTrue



A36.1. Description

The set expression is evaluated and if it is `true`, the Flow execution continues through the `Yes` output, otherwise on the `No` output. At least one of those two outputs must be connected by a line to an input.

By default, when this action is added to the Flow, a `Value` input is added and it is tested whether it is `true` or `false`. If we want to test another expression, we should delete that input in the Flow section of the property and enter the expression we want.

A36.2. Properties

Specific

A36.2.1. Value *EXPRESSION (boolean)*

Expression whose result is tested.

General

A36.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A36.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A36.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

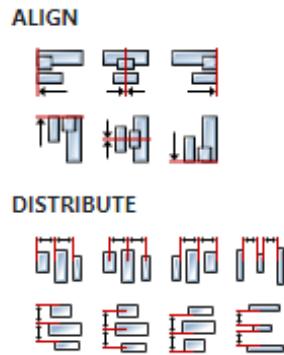
A36.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A36.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A36.3. Inputs

A36.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A36.3.2. value DATA(any) / MANDATORY

The input through which the Value to be tested is received. This input can be deleted (we delete it in the Flow - Inputs list) if it is not needed, i.e. if you want to test another expression.

A36.4. Outputs

A36.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A36.4.2. Yes SEQ / OPTIONAL

Output that will be used to continue execution of the Flow if the value of the expression is `true`.

A36.4.3. No SEQ / OPTIONAL

Output that will be used to continue execution of the Flow if the value of the expression is `false`.

A37. JSONParse



A37.1. Description

Parses a JSON string, constructs a value of the set type and sends it through the `result` output.

A37.2. Properties

Specific

A37.2.1. Value *EXPRESSION (string)*

JSON string to be parsed.

General

A37.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A37.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A37.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

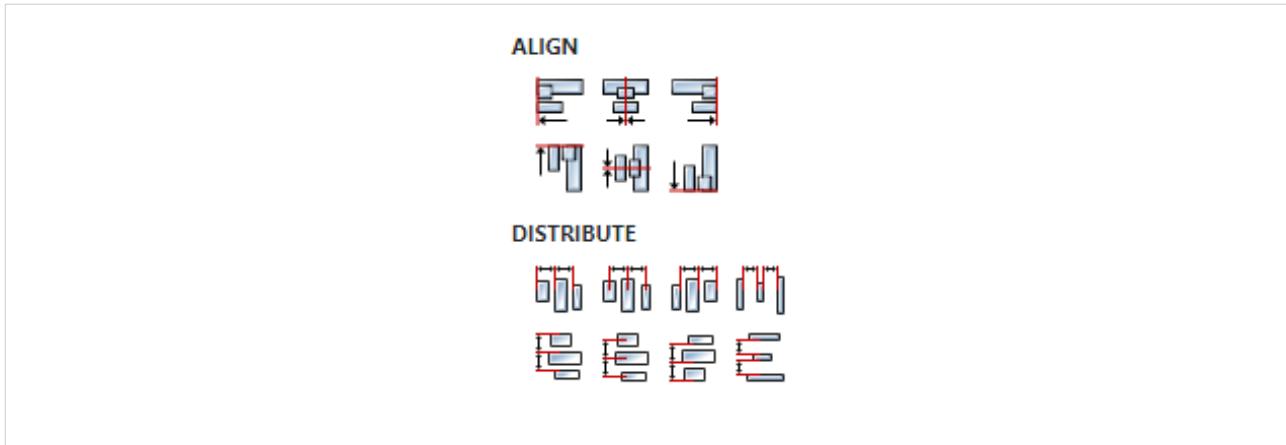
A37.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A37.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A37.3. Inputs

A37.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A37.3.2. text DATA(string) / MANDATORY

The input through which the JSON string to be parsed is received. This input can be deleted (we delete it in the Flow - Inputs list) if it is not needed, i.e. if we want to parse a string obtained by evaluating an arbitrary expression set through `Value` property.

A37.4. Outputs

A37.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A37.4.2. result DATA(json) / MANDATORY

Data output to which the constructed value is sent. The type of that value must be specified – this should be done in the Flow – Outputs section:

▼ FLOW

Inputs	
Name	Type
text	string

Outputs	
Name	Type
result	array:struct:CountryCity

In the JSON example mentioned below, we have a JSON string that looks like this:

[

```
{
    "country": "Afghanistan",
    "city": "Kabul"
},
{
    "country": "Albania",
    "city": "Tirana"
```

```

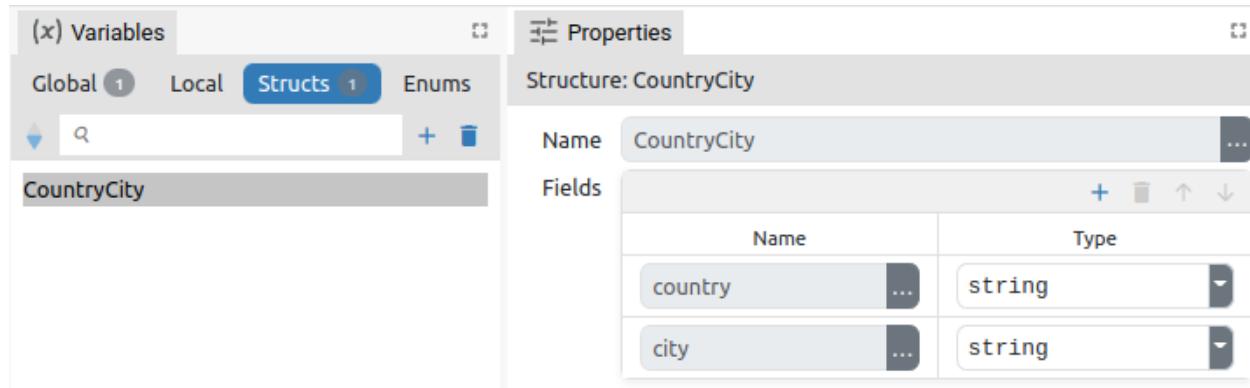
},
{
    "country": "Algeria",
    "city": "Alger"
},
...
]

```

The constructed value returned by this Action should be of type `array:CountryCity`, where `CountryCity` is a structure that has two fields (the name of the structure `CountryCity` is arbitrarily chosen by the developer):

- `country`, whose type is `string`
- `city`, whose type is `string`

The definition of that structure looks like this in the Project editor:



A37.5. Examples

- `JSON`

A38. JSONStringify



A38.1. Description

Converts the Flow `Value` to a JSON string and sends it to the `result` output.

A38.2. Properties

Specific

A38.2.1. Value *EXPRESSION (any)*

Flow value that will be converted into a JSON string.

A38.2.2. Indentation *EXPRESSION (integer)*

The indentation to be used in the generated JSON string.

General

A38.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A38.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A38.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

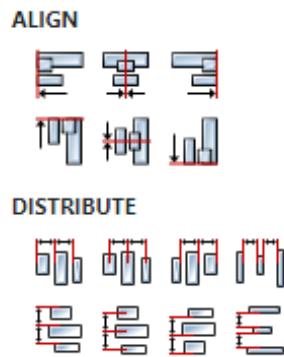
A38.2.6. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A38.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A38.3. Inputs

A38.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A38.4. Outputs

A38.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

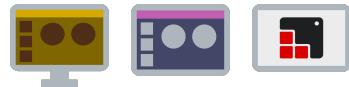
A38.4.2. result *DATA(string) / MANDATORY*

The constructed JSON string is sent through this output.

A38.5. Examples

- *JSON*

A39. Label IN



A39.1. Description

This action is used in combination with `Label OUT` action. All lines entering `Label OUT` will end up through `Label IN` with the same label name within the same flow (i.e. *Page* or *User Action*). So, "jumping" from one flow to another is not allowed. There can be multiple `Label OUT` and only one `Label IN` with the same label name.

A39.2. Properties

Specific

A39.2.1. Label *String*

The name of the label that connects the `Label IN` and `Label OUT` actions.

General

A39.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A39.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A39.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

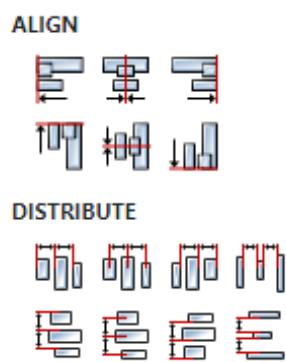
A39.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A39.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

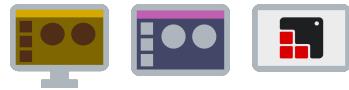


A39.3. Outputs

A39.3.1. seqout SEQ / MANDATORY

A standard sequence output.

A40. Label OUT



A40.1. Description

This action is used in combination with `Label IN` action. All lines entering `Label OUT` will end up through `Label IN` with the same label name within the same flow (i.e. *Page* or *User Action*). So, "jumping" from one flow to another is not allowed. There can be multiple `Label OUT` and only one `Label IN` with the same label name.

A40.2. Properties

Specific

A40.2.1. Label *String*

The name of the label that connects the `Label IN` and `Label OUT` actions.

General

A40.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A40.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A40.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

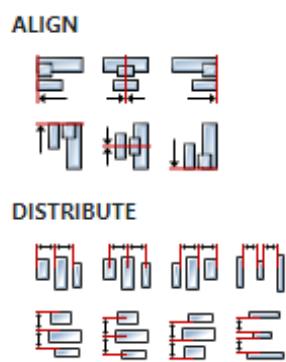
A40.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A40.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

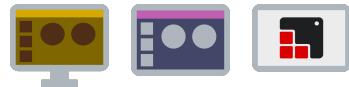


A40.3. Inputs

A40.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A41. Log



A41.1. Description

The set expression is evaluated and the result is displayed in the *Logs* panel.

A41.2. Properties

Specific

A41.2.1. Value *EXPRESSION (string)*

Expression whose result will be displayed in the *Logs* panel.

General

A41.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A41.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A41.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

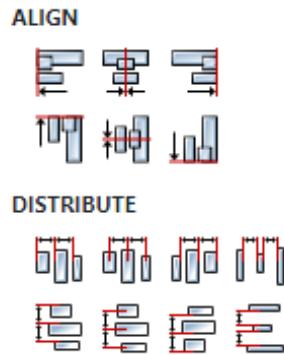
A41.2.5. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A41.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A41.3. Inputs

A41.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A41.3.2. value DATA(string) / MANDATORY

The input through which the Value that is displayed in the *Log* panel is received. This input can be deleted (it is deleted in the Flow - Inputs list) if it is not needed, i.e. if some other expression is displayed.

A41.4. Outputs

A41.4.1. seqout SEQ / OPTIONAL

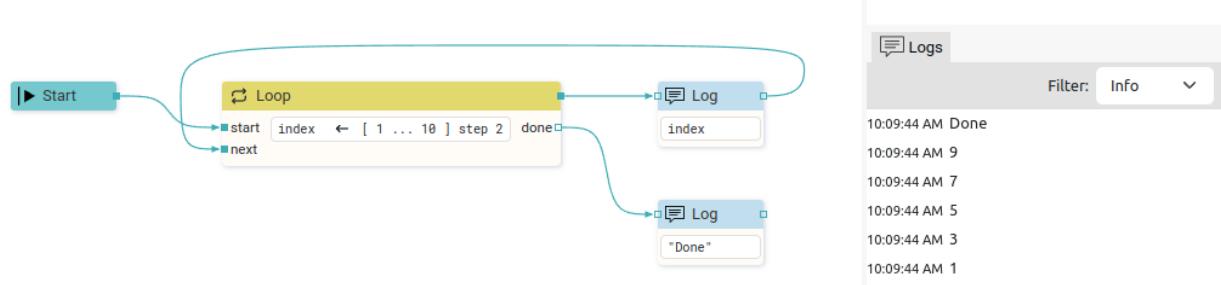
A standard sequence output.

A42. Loop



A42.1. Description

This Action is used to execute a specific part of the Flow in a loop. The Action should be placed at the beginning of the part of the Flow that will be executed in a loop and is entered at the `Start` input, and at the end of that part of the Flow it should be returned to this Action, but now through the `Next` input. Each time the Flow passes through this Action, the value of the set variable will change from the `From` to the `To` value with the `Step` value. Flow execution will go through $(\text{From} - \text{To} + 1) / \text{Math.abs}(\text{step})$ times before the iteration completes, and passes through the `Done` output. If we want to stop the iteration before the `To` value is reached, then we simply don't need to return to the `Next` input. Also, it is possible to use `SetVariable` to change the variable by which it is iterated, and thus skip one or more steps.



A42.2. Properties

Specific

A42.2.1. Variable *ASSIGNABLE EXPRESSION (integer)*

A variable that determines the number of passes through the loop and whose value will be changed and tested to see if a new iteration is needed.

A42.2.2. From *EXPRESSION (integer)*

The initial value of the variable.

A42.2.3. To *EXPRESSION (integer)*

The final value of the variable.

A42.2.4. Step *EXPRESSION (integer)*

The value by which the variable is changed on each pass. It can be a positive or negative number.

General

A42.2.5. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A42.2.6. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

A42.2.7. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

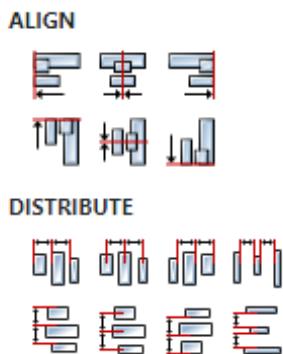
A42.2.8. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A42.2.9. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A42.3. Inputs

A42.3.1. start *SEQ / MANDATORY*

When this input is passed, the variable is set to the `From` value and Flow execution continues through `seqout`.

A42.3.2. next *SEQ / MANDATORY*

When this input is passed, the variable is changed by the `Step` value. It is tested whether it is less than or equal to `To` value if `Step` is positive, or whether it is greater than or equal to if `Step` is negative.

If the variable has not exceeded the `To` value, then Flow execution continues through `seqout`, otherwise it continues through `Done` output.

A42.4. Outputs

A42.4.1. seqout *SEQ / MANDATORY*

Flow execution continues through this output for the duration of the iteration.

A42.4.2. done *SEQ / OPTIONAL*

Flow execution continues through this output when the iteration is complete.

A42.5. Examples

- *Loop*



A43.1. Description

Performs one or more LVGL specific actions.

A43.2. Properties

Specific

A43.2.1. Actions Array

List of actions to be executed. The following actions are available:

- Change Screen: Change the screen to the specified screen
 - *Screen*: The screen to change to
 - *Fade mode*: Selection of animation when moving from the previous page to a new page
 - *Speed*: Animation duration in milliseconds
 - *Delay*: Delay in milliseconds before the animation starts.
 - *Use stack*: Put active screen on the stack.
- Change to Previous Screen: Change to the previous screen
 - *Fade mode*: Selection of animation when moving from the previous page to a new page
 - *Speed*: Animation duration in milliseconds
 - *Delay*: Delay in milliseconds before the animation starts.
- Create Screen: Create the screen ("Screens lifetime support" should be enabled in Settings - Build)
 - *Screen*: The screen to create
- Delete Screen: Delete the screen ("Screens lifetime support" should be enabled in Settings - Build)
 - *Screen*: The screen to delete
- Is Screen Created: Check if screen is created ("Screens lifetime support" should be enabled in Settings - Build)
 - *Screen*: The screen
 - *Store result into*: The boolean variable where to store the screen status
- Obj Set X: Set the x coordinate of the object
 - *Object*: The object to set the x coordinate
 - *X*: The x coordinate to set
- Obj Get X: Get the x coordinate of the object
 - *Object*: The object to get the x coordinate
 - *Store result into*: The variable to store the x coordinate
- Obj Set Y: Set the y coordinate of the object
 - *Object*: The object to set the y coordinate
 - *Y*: The y coordinate to set
- Obj Get Y: Get the y coordinate of the object
 - *Object*: The object to get the y coordinate
 - *Store result into*: The variable to store the y coordinate
- Obj Set Width: Set the width of the object
 - *Object*: The object to set the width

- *Width*: The width to set
- Obj Get Width: Get the width of the object
 - *Object*: The object to get the width
 - *Store result into*: The variable to store the width
- Obj Set Height: Set the height of the object
 - *Object*: The object to set the height
 - *Height*: The height to set
- Obj Get Height: Get the height of the object
 - *Object*: The object to get the height
 - *Store result into*: The variable to store the height
- Obj Set Style Opa: Set the opacity of the object
 - *Object*: The object to set the opacity
 - *Opacity*: The opacity to set (0-255)
- Obj Get Style Opa: Get the opacity of the object
 - *Object*: The object to get the opacity
 - *Store result into*: The variable to store the opacity
- Obj Add Style: Add a style to the object
 - *Object*: The object to add the style
 - *Style*: The style to add
- Obj Remove Style: Remove a style from the object
 - *Object*: The object to remove the style
 - *Style*: The style to remove
- Obj Set Flag Hidden: Set the hidden flag of the object
 - *Object*: The object to set the hidden flag
 - *Hidden*: The hidden flag value
- Obj Add Flag: Add a flag to the object
 - *Object*: The object to add the flag
 - *Flag*: The flag to add
- Obj Clear Flag: Clear a flag from the object
 - *Object*: The object to clear the flag
 - *Flag*: The flag to clear
- Obj Has Flag: Check if the object has the specified flag
 - *Object*: The object to check the flag
 - *Flag*: The flag to check
 - *Store result into*: The variable to store the result
- Obj Set State Checked: Set the checked state of the object
 - *Object*: The object to set the checked state
 - *Checked*: The checked state to set
- Obj Set State Disabled: Set the disabled state of the object
 - *Object*: The object to set the disabled state
 - *Disabled*: The disabled state to set
- Obj Add State: Add a state to the object
 - *Object*: The object to add the state
 - *State*: The state to add

- Obj Clear State: Clear a state from the object
 - *Object*: The object to clear the state
 - *State*: The state to clear
- Obj Has State: Check if the object has the specified state
 - *Object*: The object to check the state
 - *State*: The state to check
 - *Store result into*: The variable to store the result
- Arc Set Value: Set the value of the arc
 - *Object*: The arc to set the value
 - *Value*: The value to set
- Bar Set Value: Set the value of the bar
 - *Object*: The bar to set the value
 - *Value*: The value to set (0-100)
 - *Animated*: Use animation when setting the value
- Calendar Set Today Date: Set the today's date
 - *Object*: The calendar object
 - *Year*: Today's year
 - *Month*: Today's month [1..12]
 - *Day*: Today's day [1..31]
- Calendar Set Showed Date: Set the currently showed
 - *Object*: The calendar object
 - *Year*: Showed year
 - *Month*: Showed month [1..12]
- Calendar Set Highlighted Date: Set the highlighted date
 - *Object*: The calendar object
 - *Year*: Highlight year
 - *Month*: Highlight month [1..12]
 - *Day*: Hilighy day [1..31]
- Calendar Get Pressed Date: Get the currently pressed day
 - *Object*: The calendar object
 - *Store year into*: The integer variable where to store the year
 - *Store month into*: The integer variable where to store the month (1..12)
 - *Store day into*: The integer variable where to store the day (1..31)
- Dropdown Set Selected: Set the selected item of the dropdown
 - *Object*: The dropdown to set the selected item
 - *Selected*: The index of the selected item
- Image Set Src: Set the source image of the image
 - *Object*: The image to set the source
 - *Src*: The source image to set given as a string
- Image Set Angle: Set the angle of the image
 - *Object*: The image to set the angle
 - *Angle*: The angle to set. Angle has 0.1 degree precision, so for 45.8° set 458.
- Image Set Zoom: Set the zoom of the image

- *Object*: The image to set the zoom
- *Zoom*: The zoom to set. Set factor to 256 to disable zooming. A larger value enlarges the images (e.g. 512 double size), a smaller value shrinks it (e.g. 128 half size).
- Label Set Text: Set the text of the label
 - *Object*: The label to set the text
 - *Text*: The text to set
- Roller Set Selected: Set the selected item of the roller
 - *Object*: The roller to set the selected item
 - *Selected*: The index of the selected item
 - *Animated*: Use animation when setting the selected item
- Slider Set Value: Set the value of the slider
 - *Object*: The slider to set the value
 - *Value*: The value to set
 - *Animated*: Use animation when setting the value
- Keyboard Set Textarea: Set the textarea for the keyboard
 - *Object*: The keyboard to set the textarea
 - *Textarea*: The textarea to set
- Group Focus Obj: Focus the object
 - *Object*: The object to focus
- Group Focus Next: Focus the next object in the group
 - *Group*: The group to focus the next object
- Group Focus Prev: Focus the previous object in the group
 - *Group*: The group to focus the previous object
- Group Get Focused: Get the focused object in the group
 - *Group*: The group to get the focused object
 - *Store result into*: The variable to store the focused object
- Group Focus Freeze: Do not let to change the focus from the current object
 - *Group*: The group to freeze/unfreeze the focus
 - *Enabled*: true: freeze, false: release freezing (normal mode)
- Group Set Wrap: Set whether focus next/prev will allow wrapping from first->last or last->first object.
 - *Group*: The group to set the wrap
 - *Enabled*: true: wrap, false: no wrap
- Group Set Editing: Manually set the current mode (edit or navigate).
 - *Group*: The group to set the editing mode
 - *Enabled*: true: edit mode, false: navigate mode
- Anim X: Animate the x coordinate of the object
 - *Object*: The object to animate
 - *Start*: The start value of the animation
 - *End*: The end value of the animation
 - *Delay*: Delay in milliseconds before the animation starts
 - *Time*: Animation duration in milliseconds
 - *Relative*: Determines whether *Start* and *End* values are relative to the current value or are absolute values.
 - *Instant*: If checked apply the start value immediately, otherwise apply the start value after a

- delay when the animation really starts
- *Path*: The animation path
- Anim Y: Animate the y coordinate of the object
 - *Object*: The object to animate
 - *Start*: The start value of the animation
 - *End*: The end value of the animation
 - *Delay*: Delay in milliseconds before the animation starts
 - *Time*: Animation duration in milliseconds
 - *Relative*: Determines whether *Start* and *End* values are relative to the current value or are absolute values.
 - *Instant*: If checked apply the start value immediately, otherwise apply the start value after a delay when the animation really starts
 - *Path*: The animation path
- Anim Width: Animate the width of the object
 - *Object*: The object to animate
 - *Start*: The start value of the animation
 - *End*: The end value of the animation
 - *Delay*: Delay in milliseconds before the animation starts
 - *Time*: Animation duration in milliseconds
 - *Relative*: Determines whether *Start* and *End* values are relative to the current value or are absolute values.
 - *Instant*: If checked apply the start value immediately, otherwise apply the start value after a delay when the animation really starts
 - *Path*: The animation path
- Anim Height: Animate the height of the object
 - *Object*: The object to animate
 - *Start*: The start value of the animation
 - *End*: The end value of the animation
 - *Delay*: Delay in milliseconds before the animation starts
 - *Time*: Animation duration in milliseconds
 - *Relative*: Determines whether *Start* and *End* values are relative to the current value or are absolute values.
 - *Instant*: If checked apply the start value immediately, otherwise apply the start value after a delay when the animation really starts
 - *Path*: The animation path
- Anim Opacity: Animate the opacity of the object
 - *Object*: The object to animate
 - *Start*: The start value of the animation
 - *End*: The end value of the animation
 - *Delay*: Delay in milliseconds before the animation starts
 - *Time*: Animation duration in milliseconds
 - *Relative*: Determines whether *Start* and *End* values are relative to the current value or are absolute values.

- *Instant*: If checked apply the start value immediately, otherwise apply the start value after a delay when the animation really starts
- *Path*: The animation path
- Anim Image Zoom: Animate the zoom of the image
- *Object*: The object to animate
- *Start*: The start value of the animation
- *End*: The end value of the animation
- *Delay*: Delay in milliseconds before the animation starts
- *Time*: Animation duration in milliseconds
- *Relative*: Determines whether *Start* and *End* values are relative to the current value or are absolute values.
- *Instant*: If checked apply the start value immediately, otherwise apply the start value after a delay when the animation really starts
- *Path*: The animation path
- Anim Image Angle: Animate the angle of the image
- *Object*: The object to animate
- *Start*: The start value of the animation
- *End*: The end value of the animation
- *Delay*: Delay in milliseconds before the animation starts
- *Time*: Animation duration in milliseconds
- *Relative*: Determines whether *Start* and *End* values are relative to the current value or are absolute values.
- *Instant*: If checked apply the start value immediately, otherwise apply the start value after a delay when the animation really starts
- *Path*: The animation path

General

A43.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A43.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A43.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

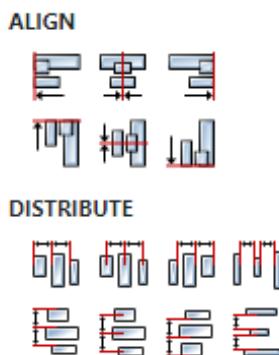
A43.2.5. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A43.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A43.3. Inputs

A43.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A43.4. Outputs

A43.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A43.5. Examples

- Change Screen

A44. Modbus



A44.1. Description

This action is used to send Modbus commands to the Modbus server. If coils are read, then the read value will be passed through output `values` as a value of type `array:boolean`, and in the case of registers, then a value of type `array:integer` will be passed through output `values`.

A44.2. Properties

Specific

A44.2.1. Connection *EXPRESSION (any)*

Serial connection used to send Modbus commands.

A44.2.2. Server address *EXPRESSION (integer)*

A number between 0 and 255 used to select the Modbus server on the serial connection.

A44.2.3. Command *Enum*

Command to be sent:

- 01 (0x01) Read Coils
- 02 (0x02) Read Discrete Inputs
- 03 (0x03) Read Holding Registers
- 04 (0x04) Read Input Registers
- 05 (0x05) Write Single Coil
- 06 (0x06) Write Single Register
- 15 (0x0F) Write Multiple Coils
- 16 (0x10) Write Multiple Registers

A44.2.4. Register address *EXPRESSION (integer)*

Register address for single write: 05 (0x05) Write Single Coil or 06 (0x06) Write Single Register.

A44.2.5. Starting register address *EXPRESSION (integer)*

The address of the first register for multiple read and write.

A44.2.6. Quantity of registers *EXPRESSION (integer)*

The register number for multiple read and write.

A44.2.7. Coil value *EXPRESSION (boolean)*

Coil value (`boolean`) that is sent during a single write (i.e. when 05 (0x05) Write Single Coil is used).

A44.2.8. Register value *EXPRESSION (integer)*

Register value (`integer`) that is sent during a single write (i.e. when 06 (0x06) Write Single Register is used).

A44.2.9. Coil values *EXPRESSION (array:boolean)*

Coil values (of type `array:boolean`) when multiple writes are performing.

A44.2.10. Register values *EXPRESSION (array:integer)*

Registers values (of type `array:integer`) when multiple writes are performing.

A44.2.11. Timeout (ms) EXPRESSION (integer)

Maximum waiting time for server response. It is set in milliseconds.

General**A44.2.12. Description String**

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow**A44.2.13. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A44.2.14. Outputs Array

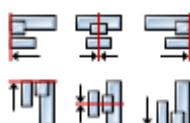
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A44.2.15. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size**A44.2.16. Align and distribute Any**

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE**

A44.3. Inputs

A44.3.1. seqin SEQ / OPTIONAL

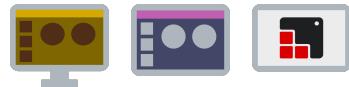
A standard sequence input.

A44.4. Outputs

A44.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A45. MQTTConnect



A45.1. Description

This Action initiates a connection to the MQTT server, and if the connection is successful, a Connect event will be sent, or an Error event if an error occurred. If an error occurred or the once established connection was interrupted, a periodic reconnect will be attempted until the connection is re-established, which will be reported by sending a Reconnect event. All this happens asynchronously in the background, until `MQTTDisconnect` is called, and any state change will be reported with an event that can be processed through the `MQTTEvent` Action.

A45.2. Properties

Specific

A45.2.1. Connection *EXPRESSION (object:MQTTConnection)*

The name of the MQTT connection that will be used to establish a connection with the server.

General

A45.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A45.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A45.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

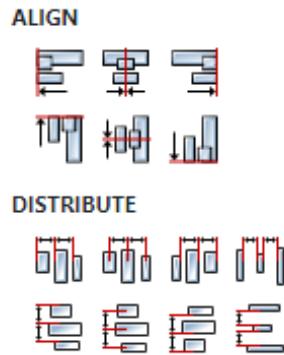
A45.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A45.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A45.3. Inputs

A45.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A45.4. Outputs

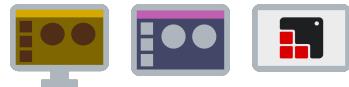
A45.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output. Flow execution continues immediately through this output, and in the background it tries to establish a connection with the server.

A45.5. Examples

- *MQTT*

A46. MQTTDisconnect



A46.1. Description

Initiates the termination of the connection with the server, which will be confirmed with the `Close` event and then the `End` event.

A46.2. Properties

Specific

A46.2.1. Connection *EXPRESSION (object:MQTTConnection)*

The name of the MQTT connection to the server to which the communication will be terminated.

General

A46.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A46.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A46.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

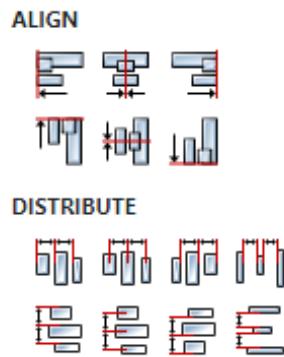
A46.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A46.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A46.3. Inputs

A46.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A46.4. Outputs

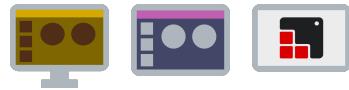
A46.4.1. seqout SEQ / OPTIONAL

A standard sequence output. Flow execution continues immediately through this output, and in the background it tries to disconnect from the server.

A46.5. Examples

- *MQTT*

A47. MQTTEvent



A47.1. Description

With this Action we can add one or more event handlers that can be received by the MQTT connection. After this Action is executed, the *MQTTConnect* Action can be called.

A47.2. Properties

Specific

A47.2.1. Connection *EXPRESSION (object:MQTTConnection)*

MQTT connection to the server whose events are to be handled.

A47.2.2. Event handlers *Array*

List of events to be handled. For each item in the list, it will be necessary to select `Event`, `Handler type` and optionally `Action`. `Event` is the type of event we want to handle and the possible values are:

- `Connect` – It is sent in case of successful connection or reconnect.
- `Reconnect` – Sent when attempting to reconnect after a connection has been terminated.
- `Close` – It is sent after the connection is terminated.
- `Disconnect` – Sent when a disconnect packet is received by the broker.
- `Offline` – Sent when the client goes offline.
- `End` – Sent when the *MQTTDisconnect* Action is performed.
- `Error` – Sent when the client cannot connect or a parsing error has occurred.
- `Message` – It is sent when the client receives a published packet from the server for the topic we previously subscribed to with the *MQTTSubscribe* Action. Data of the type `struct: $MQTTMessage` is sent through the output, it is a system structure that has these members:
 - `topic` – The name of the topic for which the packet was published.
 - `payload` – Content of the received message.

`Handler type` can be `Flow` or `Action`. If `Flow` is selected then an output will be added through which the Flow execution continues if the event is sent. If `Action` is selected, then `Action` must also be set, i.e. the name of the User action that is executed when the event is received.

General

A47.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A47.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A47.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

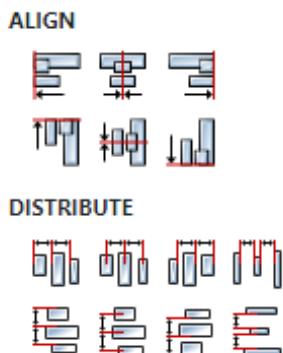
A47.2.6. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A47.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A47.3. Inputs

A47.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A47.4. Outputs

A47.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A47.5. Examples

- *MQTT*

A48. MQTTInit



A48.1. Description

Creates and initializes an MQTT connection object with connection parameters that are defined through properties. This Action must be executed first, and after it the *MQTTEvent* Action must be called.

A48.2. Properties

Specific

A48.2.1. Connection *ASSIGNABLE EXPRESSION (object:MQTTConnection)*

Connection object of type `object:MQTTConnection` which will be created and initialized.

A48.2.2. Protocol *EXPRESSION (string)*

The protocol used for the connection. Possible values are `"mqtt"` or for secure connection `"mqtts"`

A48.2.3. Host *EXPRESSION (string)*

The name of the MQTT server to connect to.

A48.2.4. Port *EXPRESSION (integer)*

The port number that will be used for the connection. The default is `1883`.

A48.2.5. User name *EXPRESSION (string)*

Username to be used for connection authorization. Can be left blank if not used.

A48.2.6. Password *EXPRESSION (string)*

User password to be used for connection authorization. Can be left blank if not used.

General

A48.2.7. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A48.2.8. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A48.2.9. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through

that output.

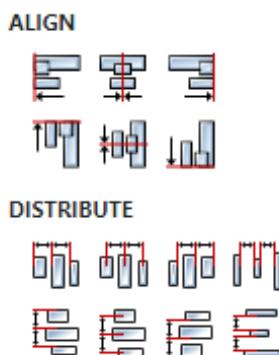
A48.2.10. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A48.2.11. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A48.3. Inputs

A48.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A48.4. Outputs

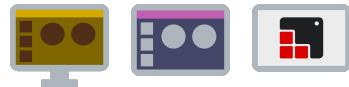
A48.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A48.5. Examples

- *MQTT*

A49. MQTT Publish



A49.1. Description

Publishing a message in the selected topic.

A49.2. Properties

Specific

A49.2.1. Connection *EXPRESSION (object:MQTTConnection)*

The name of the connection to the MQTT server.

A49.2.2. Topic *EXPRESSION (string)*

The topic under which the message will be published.

A49.2.3. Payload *EXPRESSION (string)*

Message to be published.

General

A49.2.4. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A49.2.5. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A49.2.6. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A49.2.7. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

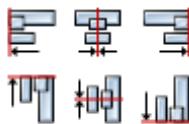
Position and size

A49.2.8. Align and distribute *Any*

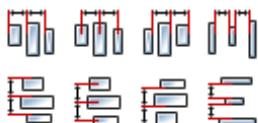
Alignment icons and component distribution. Alignment icons appear when two or more compo-

nents are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A49.3. Inputs

A49.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A49.4. Outputs

A49.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output

A49.5. Examples

- *MQTT*

A50. MQTTSubscribe



A50.1. Description

This Action must be performed immediately after successfully connecting to the MQTT server, for each topic to which we want to subscribe. If a packet has been published by the server for this topic, we will receive information about it via the *MQTTEvent* Action.

A50.2. Properties

Specific

A50.2.1. Connection *EXPRESSION (object:MQTTConnection)*

The name of the connection to the MQTT server.

A50.2.2. Topic *EXPRESSION (string)*

The name of the topic to which we want to subscribe. A subscription may be to an explicit topic, in which case only messages to that topic will be received, or it may include wildcards. Two wildcards are available, + or #. + can be used as a wildcard for a single level of hierarchy. It could be used with the topic above to get information on all computers and hard drives as follows:

sensors/+/temperature/+

As another example, for a topic of a/b/c/d, the following example subscriptions will match:

a/b/c/d
+/b/c/d
a/+/c/d
a/+/+/d
+/-/+/-

The following subscriptions will not match:

a/b/c
b/+/c/d
+/-/+/-

can be used as a wildcard for all remaining levels of hierarchy. This means that it must be the final character in a subscription. With a topic of a/b/c/d, the following example subscriptions will match:

a/b/c/d

a/#
a/b/#
a/b/c/#
+/-/c/#

General

A50.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A50.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to

check whether a data line that transmits data of that type is connected to the input or not.

A50.2.5. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

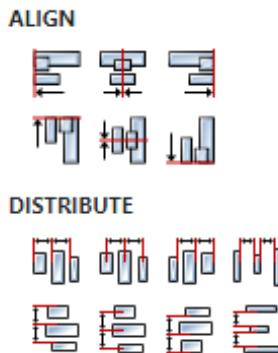
A50.2.6. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A50.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A50.3. Inputs

A50.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A50.4. Outputs

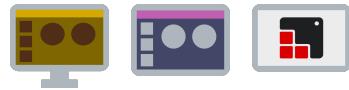
A50.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A50.5. Examples

- *MQTT*

A51. MQTTUnsubscribe



A51.1. Description

Unsubscribe from a topic.

A51.2. Properties

Specific

A51.2.1. Connection *EXPRESSION (object:MQTTConnection)*

The name of the connection to the MQTT server.

A51.2.2. Topic *EXPRESSION (string)*

Topic to unsubscribe from.

General

A51.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A51.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A51.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

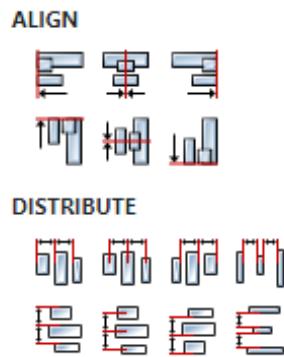
A51.2.6. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A51.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A51.3. Inputs

A51.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A51.4. Outputs

A51.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A51.5. Examples

- MQTT

A52. NoOp



A52.1. Description

This action does nothing, i.e. Flow execution continues through `seqout`.

A52.2. Properties

Specific

A52.2.1. Name *String*

The name displayed in the component view within the Flow.

General

A52.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A52.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A52.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

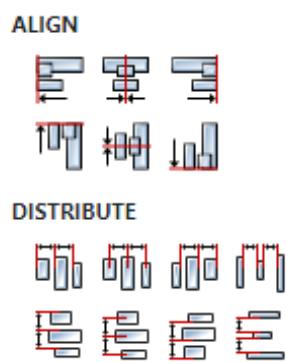
A52.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A52.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A52.3. Inputs

A52.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A52.4. Outputs

A52.4.1. seqout SEQ / OPTIONAL

A standard sequence output

A53. OnEvent



A53.1. Description

It is used to process events that can be broadcast within the page where the Action is located.

A53.2. Properties

Specific

A53.2.1. Event *Enum*

Event to be processed. The following page events are available:

- `Page open` - emitted when the page becomes active, eg when it is displayed with the 'ShowPage' Action.
- `Page close` - emitted when the page becomes inactive.
- `Keypress` - emitted when a key on the keyboard is pressed. A string with keyboard name ([link](#)) is sent to the `event` output.

General

A53.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A53.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A53.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A53.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

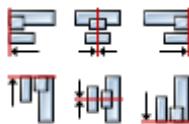
Position and size

A53.2.6. Align and distribute *Any*

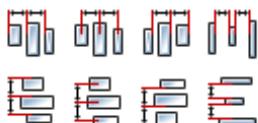
Alignment icons and component distribution. Alignment icons appear when two or more compo-

nents are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A53.3. Outputs

A53.3.1. seqout SEQ / MANDATORY

A standard sequence output. Flow execution continues through this output when the selected event is emitted.

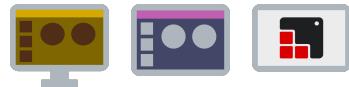
A53.3.2. event DATA(any) / OPTIONAL

Through this output, additional information (if any) is sent for the broadcast event. The `Page open` and `Page close` events do not send anything through this event, and the `Keystroke` event sends a string with key name ([link](#)).

A53.4. Examples

- *Tetris*

A54. Output



A54.1. Description

Adds data output to a user action or user widget.

A54.2. Properties

Specific

A54.2.1. Name *String*

Output name.

A54.2.2. Type *String*

Output data type.

General

A54.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A54.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A54.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

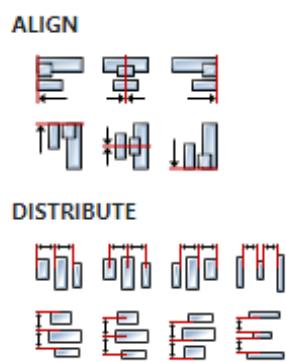
A54.2.6. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A54.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A54.3. Inputs

A54.3.1. seqin SEQ / MANDATORY

Data is received through this input, which is then forwarded to the caller of the user action.

A55. OverrideStyle



A55.1. Description

The action will replace one style with another style, so that all Widgets that use that style will use the new style after this replacement. This Action is used if you want to dynamically change the appearance of a Widget.

A55.2. Properties

Specific

A55.2.1. From *ObjectReference*

The style to be replaced.

A55.2.2. To *ObjectReference*

A new style that will replace the existing one.

General

A55.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A55.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A55.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

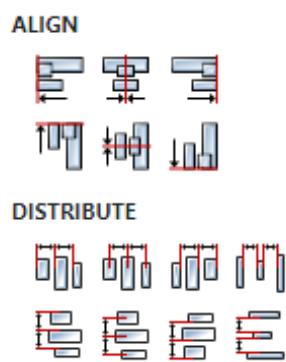
A55.2.6. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A55.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A55.3. Inputs

A55.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A55.4. Outputs

A55.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A56. PlayAudio



A56.1. Description

Play audio file.

A56.2. Properties

Specific

A56.2.1. Audio file *EXPRESSION (string)*

File path to audio file.

General

A56.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A56.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A56.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

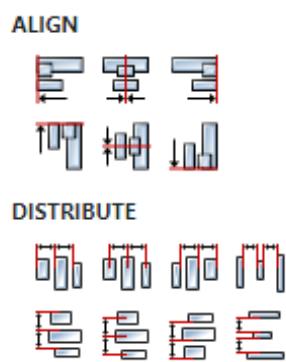
A56.2.5. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A56.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A56.3. Inputs

A56.3.1. seqin SEQ / OPTIONAL

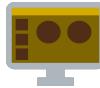
A standard sequence input.

A56.4. Outputs

A56.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A57. PrintToPDF



A57.1. Description

It is used to print the content of the Widget. Currently, only printing of Tabulator widget is supported.

A57.2. Properties

Specific

A57.2.1. Widget *EXPRESSION (widget)*

Reference to the Tabulator widget. See `Output widget handle` property to find out how to obtain this reference.

A57.2.2. Options *EXPRESSION (json)*

You can specify following print options through JSON:

- `landscape` boolean (optional) - Paper orientation. true for landscape, false for portrait. Defaults to false.
- `scale` number(optional) - Scale of the webpage rendering. Defaults to 1.
- `pageSize` string | `Size` (optional) - Specify page size of the generated PDF. Can be A0, A1, A2, A3, A4, A5, A6, Legal, Letter, Tabloid, Ledger, or an Object containing height and width in inches. Defaults to Letter.
- `margins` Object (optional)
 - `marginType` string | `Size` (optional) - Can be "default" or "custom".
 - `top` number (optional) - Top margin in inches. Defaults to 1cm (~0.4 inches).
 - `bottom` number (optional) - Bottom margin in inches. Defaults to 1cm (~0.4 inches).
 - `left` number (optional) - Left margin in inches. Defaults to 1cm (~0.4 inches).
 - `right` number (optional) - Right margin in inches. Defaults to 1cm (~0.4 inches).

For example:

```
{  
    landscape: true,  
    scale: 1,  
    pageSize: "A4",  
    margins: {  
        marginType: "custom",  
        top: 0.8,  
        bottom: 0.8,  
        left: 0.8,  
        right: 0.8  
    }  
}
```

General

A57.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A57.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A57.2.5. Outputs *Array*

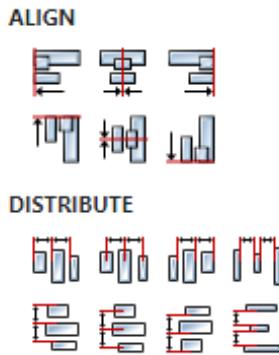
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A57.2.6. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size**A57.2.7. Align and distribute** *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**A57.3. Inputs****A57.3.1. seqin** *SEQ / OPTIONAL*

A standard sequence input.

A57.4. Outputs**A57.4.1. seqout** *SEQ / OPTIONAL*

A standard sequence output.

A57.5. Examples

- *Tabulator Examples*

A58. PythonEnd



A58.1. Description

Stops a running python script.

A58.2. Properties

Specific

A58.2.1. Handle *EXPRESSION (integer)*

The handle obtained during the execution of *PythonRun* actions, and is used to determine which script we want to stop, since multiple scripts can be executed.

General

A58.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A58.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A58.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

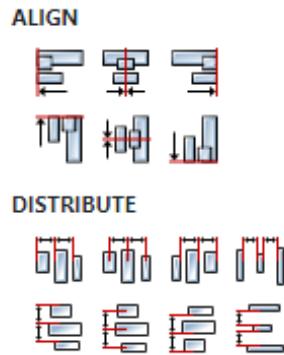
A58.2.5. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A58.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A58.3. Inputs

A58.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A58.3.2. handle *DATA(integer) / MANDATORY*

The handle can also be passed through this input. If the handle is obtained in some other way, e.g. from a variable via the `Handle` property, then this input can be removed in the "Flow - Inputs" section.

A58.4. Outputs

A58.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

A59. PythonRun



A59.1. Description

Runs a python script and sends the handle of the running script to the `handle` output. This handle is used in the `PythonEnd` Action if we want to stop a running Python script or in the `PythonSendMessage` Action if we want to send a message from Flow to a Python script, and it is needed because several scripts can be started at some point and the running script is determined through this handle.

A59.2. Properties

Specific

A59.2.1. Script source option *Enum*

The source of the python script can be specified in three ways:

- Inline script
- Inline script as expression
- Script file

A59.2.2. Inline script *Python*

If `Inline script` was selected for `Script source option`, then the source code of the script should be entered here.

A59.2.3. Inline script as expression *EXPRESSION (string)*

If `Inline script as expression` was selected for `Script source option`, then here you need to enter an expression that will return a string containing the source code of the script when evaluated.

A59.2.4. Script file *EXPRESSION (string)*

If `Script file` was selected for `Script source option`, then the file path to the '.py' file should be entered here.

A59.2.5. Python path *EXPRESSION (string)*

The full path to the python command. If the python command is already in the system path, then it can be set to an empty string, i.e. "".

General

A59.2.6. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A59.2.7. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check to*

check whether a data line that transmits data of that type is connected to the input or not.

A59.2.8. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

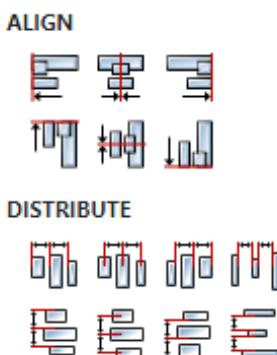
A59.2.9. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A59.2.10. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A59.3. Inputs

A59.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A59.4. Outputs

A59.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A59.4.2. handle DATA(integer) / OPTIONAL

Returns the handle of the running script used in *PythonEnd* and *PythonSendMessage* Actions.

A59.4.3. message DATA(string) / OPTIONAL

Everything that is printed to `stdout` within the running Python script will be sent through this output. In this way, the python script sends a message to Flow, and if Flow wants to send a message to the Python script, then the *PythonSendMessage* Action should be used.

A59.5. Examples

- *Charts*

A60. PythonSendMessage



A60.1. Description

Sends a message from Flow to a running Python script.

A60.2. Properties

Specific

A60.2.1. Handle *EXPRESSION (integer)*

The handle obtained during the execution of the *PythonRun* action is used to determine which script we want to send the message to, since multiple scripts can be executed at the same time.

A60.2.2. Message *EXPRESSION (string)*

Message to be sent.

General

A60.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A60.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A60.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

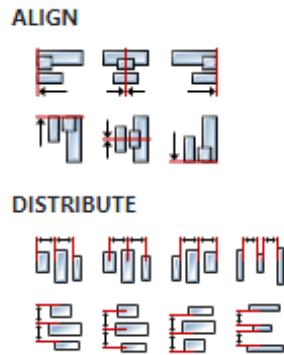
A60.2.6. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A60.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A60.3. Inputs

A60.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A60.3.2. handle DATA(integer) / MANDATORY

The handle can also be passed through this input. If the handle is obtained in some other way, e.g. from a variable via the `Handle` property, then this input can be removed in the "Flow - Inputs" section.

A60.4. Outputs

A60.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A60.5. Examples

- *Charts*

A61. ReadSetting



A61.1. Description

This action, for the defined key name, returns the saved value, or `null` if that key does not exist, from the `.eez-project-runtime-settings` file (it's the same file where persistent variables are saved).

NOTE: *WriteSetting* and *ReadSetting* Actions are used to save and retrieve from the `.eez-project-runtime-settings` file all those settings that we want to survive the *Dashboard* project restart. It is more convenient to use persistent variables, because in that case we do not have to execute a special Action for saving and

A61.2. Properties

Specific

A61.2.1. Key *EXPRESSION (string)*

A string containing the name of the key whose value is to be retrieved.

General

A61.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A61.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A61.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

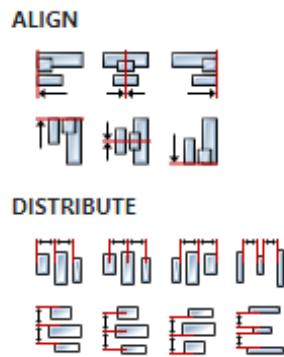
A61.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A61.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A61.3. Inputs

A61.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A61.4. Outputs

A61.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A61.4.2. value DATA(any) / MANDATORY

The obtained `Value` of the defined `Key` is sent through this output.

A62. Regexp



A62.1. Description

Searches a set string or stream, using a pattern written according to the rules of the regular expression syntax.

A62.2. Properties

Specific

A62.2.1. Pattern *EXPRESSION (string)*

Regular expression used for searching.

A62.2.2. Text *EXPRESSION (string)*

The text to be searched can be a string or a stream.

A62.2.3. Global *EXPRESSION (boolean)*

This option determines whether only the first occurrence of the pattern or every occurrence of the pattern is searched.

A62.2.4. Case insensitive *EXPRESSION (boolean)*

This option determines whether the search will be case sensitive or not.

General

A62.2.5. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A62.2.6. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A62.2.7. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

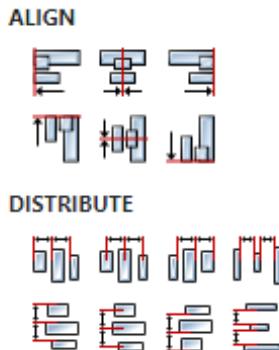
A62.2.8. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A62.2.9. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A62.3. Inputs

A62.3.1. seqin SEQ / OPTIONAL

A standard sequence input. This input needs to be used once at the beginning.

A62.3.2. next SEQ / OPTIONAL

Use this input to get the next match.

A62.3.3. stop SEQ / OPTIONAL

Use this input when we want to stop further searching, after which the Flow execution will immediately continue through the `done` output.

A62.4. Outputs

A62.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A62.4.2. match DATA(struct:\$RegExpResult) / MANDATORY

Search match in the form of `struct:$RegexpMatch` value is sent through this output. The `$RegexpMatch` structure has the following fields:

- `index` (integer) - The 0-based index of the match in the string.
- `texts` (array:string) - The array that has the matched text as the first item, and then one item for each capturing group ([link](#)) of the matched text.
- `indices` (array:array:integer) - It is an array where each entry represents the bounds of a substring match. The index of each element in this array corresponds to the index of the respective substring match in the `texts` array. In other words, the first `indices` entry represents the entire match, the second `indices` entry represents the first capturing group, etc. Each entry itself is a two-element array, where the first number represents the match's start index, and the second number, its end index.

A1.1.1. done DATA(string) / OPTIONAL

Flow execution continues through this output when the search is complete, i.e. there are no more matches.

A62.5. Examples

- *RegExp String*
- *RegExp Stream*



A63.1. Description

Executes one or more SCPI commands or queries on the selected instrument. When all commands/queries are executed Flow execution continues through `seqout` output.

A63.2. Properties

Specific

A63.2.1. Instrument *EXPRESSION (object:Instrument)*

Instrument object on which commands/queries are executed. This property is only present within the *Dashboard* project when the instrument is connected remotely, i.e. it is possible to have open connections to several instruments at the same time. If it is an *EEZ-GUI* project, then this property does not exist because we always use the device on which Flow is executed and we send SCPI commands to it.

A63.2.2. Scpi *TEMPLATE LITERAL*

List of SCPI commands/queries. Each command/query must be entered as a separate line. A Flow expression can also be inserted inside the command/query, which must be entered between two curly brackets. This is an example taken from the *BB3 Dashboard* example that uses a Flow expression within an SCPI command:

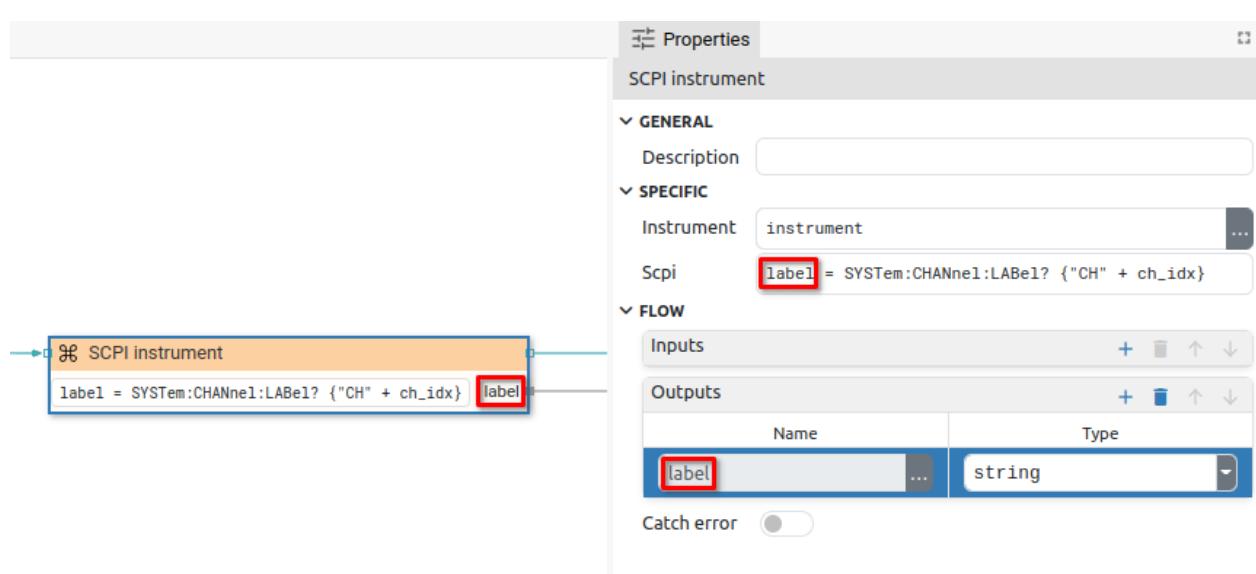


**A flow expression is inserted
inside an SCPI command/query by
entering it inside curly brackets.**

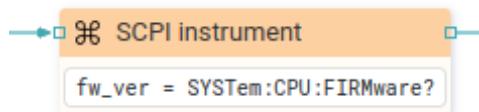
Also in the example above, a Flow `Catch Error` has been added to catch an error during the execution of the SCPI component.

For an SCPI query, it must be specified where the result is sent, and there we have two options:

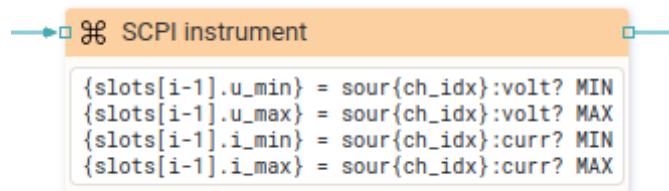
- Sending results to Flow output. It is necessary to add a new output using the "Flow - Outputs" section in the properties of this component, where it is necessary to write: `output_name=query?`. Here's an example, taken from the *BB3 Dashboard* example:



- Saving the result in a variable. The results are saved in a variable so that the query is written like this: `variable_name=query?` or `{assignable_expression}=query?`. This second form is used when it is stored, for example, in a structure member or an array. Here are examples for both forms, also taken from the *BB3 Dashboard* example:
- In this example, the result of the `SYSTem:CPU:FIRMware?` query is saved in the `fw_ver` variable. As it is the first (simple) form, then the name of the variable should not be enclosed in curly brackets.



- In this example, four SCPI queries are executed. The results are saved in the `slots` variable of the type: `array:struct:Slot`, where `slots` is a structure that has `u_min`, `u_max`, `i_min` and `i_max` members. The second form is used here and the assignable expression must be enclosed in curly brackets. Also here we have an example of using the expression `{ch_idx}` within the query itself.



A63.2.3. Timeout (ms) *EXPRESSION (integer)*

The time in milliseconds to wait for the result of the query. If the result does not expire within that time, a Timeout error is generated, which can be handled through `@Error` output if `Catch error` is enabled. If set to 'null' then the timeout as specified in the Instrument Connect dialog is used.

A63.2.4. Delay (ms) *EXPRESSION (integer)*

The minimum time specified in milliseconds that must elapse before a new SCPI command or query is sent. If set to 'null' then the delay as specified in the Instrument Connect dialog is used.

General

A63.2.5. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A63.2.6. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A63.2.7. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

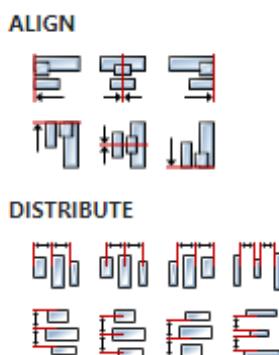
A63.2.8. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A63.2.9. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A63.3. Inputs

A63.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A63.4. Outputs

A63.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A63.5. Examples

- *BB3 Dashboard*
- *Plotly*
- *Rigol Waveform Data*
- *Screen Capture*

A64. SelectInstrument



A64.1. Description

Opens a dialog box for selecting an instrument. The selected instrument is sent to the `instrument` output.

It will not be necessary to use this Action if the global instrument object variable is set to `Persistent`, because the instrument selection dialog box will open immediately when the dashboard is started. However, if we don't want the instrument selection dialog box to open automatically at startup, then we must not enable the `Persistent` checkbox for the global instrument variable and we can use this Action later to select the desired instrument.

The screenshot shows the Project editor interface. On the left is the 'Variables' panel, listing several variables including 'instrument' (which is highlighted). On the right is the 'Properties' panel, showing the properties for the 'instrument' variable. The 'GENERAL' section contains the following fields:

Id	
Name	instrument
Description	
Type	object:Instrument
Default value	null
<input type="checkbox"/> Persistent	(unchecked)

A64.2. Properties

General

A64.2.1. Description String

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A64.2.2. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A64.2.3. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

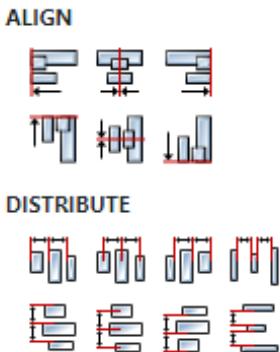
A64.2.4. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A64.2.5. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A64.3. Inputs

A64.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A64.4. Outputs

A64.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A64.4.2. instrument DATA(object:Instrument) / MANDATORY

The selected instrument is sent to this output.

A65. SelectLanguage



A65.1. Description

Changes the active language in multilanguage projects, i.e. projects that have the *Texts* feature added. After this, all texts on the page will be displayed in the newly selected language.

A65.2. Properties

Specific

A65.2.1. Language *EXPRESSION (any)*

ID of the language that will become the new active language.

General

A65.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A65.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A65.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

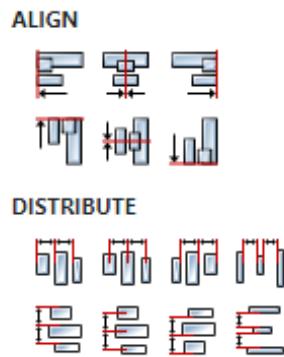
A65.2.5. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A65.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A65.3. Inputs

A65.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A65.4. Outputs

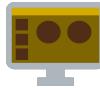
A65.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A65.5. Examples

- *Multi-Language*
- *Multi-Language Dashboard*

A66. SerialConnect



A66.1. Description

Makes a connection to the serial port. If the connection is successful, Flow execution continues through the `seqout` output, and if an error occurred, it can be caught if `Catch error` is enabled.

A66.2. Properties

Specific

A66.2.1. Connection *EXPRESSION (object:SerialConnection)*

The name of the connection to be used for serial communication.

General

A66.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A66.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A66.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

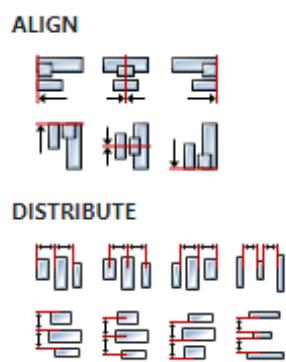
A66.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A66.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A66.3. Inputs

A66.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A66.4. Outputs

A66.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A66.5. Examples

- *SerialPort*

A67. SerialDisconnect



A67.1. Description

Performs disconnection from serial port, after which Flow execution continues through `seqout` output.

A67.2. Properties

Specific

A67.2.1. Connection *EXPRESSION (object:SerialConnection)*

The name of the connection that will be terminated.

General

A67.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A67.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A67.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

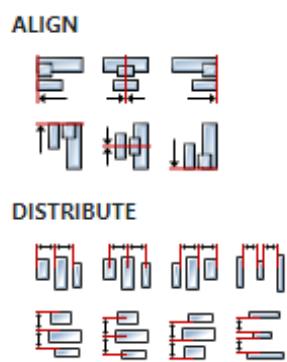
A67.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A67.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A67.3. Inputs

A67.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A67.4. Outputs

A67.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A67.5. Examples

- *SerialPort*

A68. SerialInit



A68.1. Description

Creates and initializes a Serial connection object with connection parameters that are defined through properties. This Action must be executed first, after which the *SerialConnect* Action must be called.

A68.2. Properties

Specific

A68.2.1. Connection *ASSIGNABLE EXPRESSION (object:SerialConnection)*

Connection object of type `object:SerialConnection` to be created and initialized.

A68.2.2. Port *EXPRESSION (object:string)*

Serial port name.

A68.2.3. Baud rate *EXPRESSION (object:number)*

Serial port speed.

A68.2.4. Data bits *EXPRESSION (object:number)*

Serial port data bits. Allowed values are `5`, `6`, `7` or `8`.

A68.2.5. Stop bits *EXPRESSION (object:number)*

Serial port stop bits. Allowed values are `1` or `2`.

A68.2.6. Parity *EXPRESSION (object:string)*

Serial port parity. Allowed values are `"none"`, `"even"`, `"mark"`, `"odd"` or `"space"`

General

A68.2.7. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A68.2.8. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A68.2.9. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through

that output.

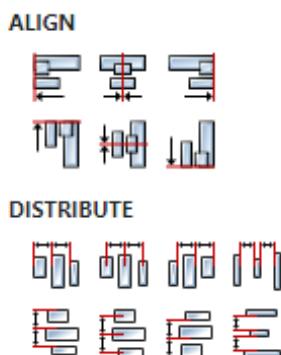
A68.2.10. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A68.2.11. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A68.3. Inputs

A68.3.1. seqin SEQ / OPTIONAL

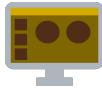
A standard sequence input.

A68.4. Outputs

A68.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A69. SerialListPorts



A69.1. Description

Retrieves the list of serial ports detected on the system and sends it through `ports` output.

A69.2. Properties

General

A69.2.1. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A69.2.2. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A69.2.3. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

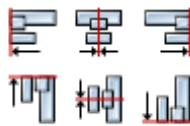
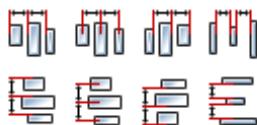
A69.2.4. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A69.2.5. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****A69.3. Inputs****A69.3.1. seqin SEQ / OPTIONAL**

A standard sequence input.

A69.4. Outputs**A69.4.1. seqout SEQ / OPTIONAL**

A standard sequence output.

A69.4.2. ports DATA(array:struct:\$SerialPort) / MANDATORY

A list of ports is sent to this output as a value of type `array:$SerialPort`. The system structure `$SerialPort` has these members:

- `manufacturer: string`. The name of the manufacturer of the device connected to the port.
- `serialNumber: string`. Port serial number.
- `path: string`. Path of the serial port, which is used in the *SerialInit* Action.

A70. SerialRead



A70.1. Description

Sends the read stream received via the selected serial connection to the `data` output.

A70.2. Properties

Specific

A70.2.1. Connection *EXPRESSION (object:SerialConnection)*

The name of the serial connection.

General

A70.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A70.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A70.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

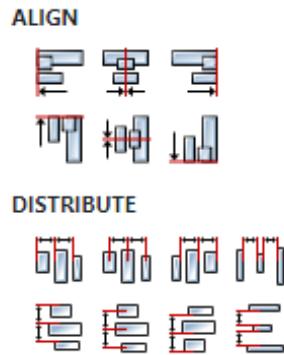
A70.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A70.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A70.3. Inputs

A70.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A70.4. Outputs

A70.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A70.4.2. data DATA(stream) / MANDATORY

Output to which the read stream is sent.

A70.5. Examples

- *SerialPort*

A71. SerialWrite



A71.1. Description

Sends a string to the serial port.

A71.2. Properties

Specific

A71.2.1. Connection *EXPRESSION (object:SerialConnection)*

The name of the serial connection.

A71.2.2. Data *EXPRESSION (string)*

The string that is sent to the serial port.

General

A71.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A71.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A71.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

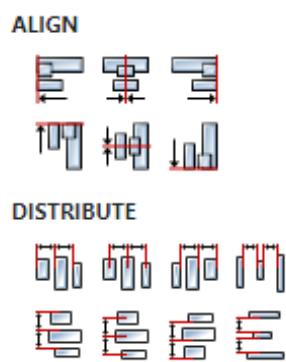
A71.2.6. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A71.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A71.3. Inputs

A71.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A71.4. Outputs

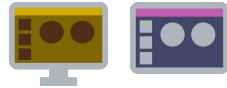
A71.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A71.5. Examples

- *SerialPort*

A72. SetPageDirection



A72.1. Description

It is used to change the page layout from LTR (left to right) to RTL (right to left) and vice versa.

A72.2. Properties

Specific

A72.2.1. Direction *Enum*

Selected page layout: `LTR` or `RTL`.

General

A72.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A72.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A72.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

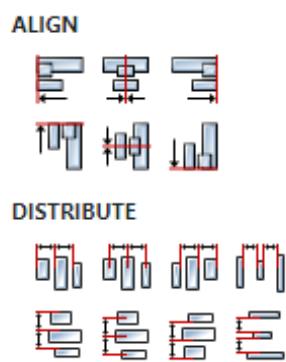
A72.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A72.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A72.3. Inputs

A72.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A72.4. Outputs

A72.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A73. SetVariable



A73.1. Description

It is used to set a new value to one or more variables.

A73.2. Properties

Specific

A73.2.1. Set variable entries Array

List of variables to be set. Each element of the list contains a given variable name to which a new value is added, which is obtained by evaluating the given expression.

General

A73.2.2. Description String

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A73.2.3. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A73.2.4. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

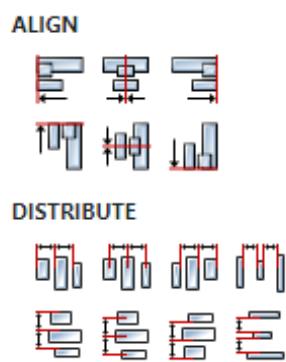
A73.2.5. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A73.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A73.3. Inputs

A73.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A73.4. Outputs

A73.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A74. ShowFileInFolder



A74.1. Description

Displays the set file in the system file manager. When possible, the set file will also be selected.

A74.2. Properties

Specific

A74.2.1. File path *EXPRESSION (string)*

Path to the file that will be displayed in the system file manager.

General

A74.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A74.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A74.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

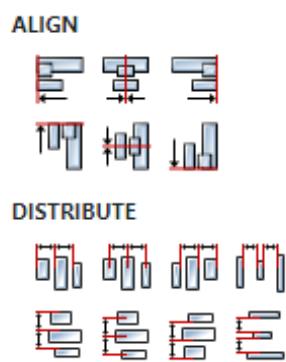
A74.2.5. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A74.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A74.3. Inputs

A74.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A74.4. Outputs

A74.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A74.5. Examples

- *Screen Capture*

A75. ShowKeyboard



A75.1. Description

Opens the keyboard page for text input. The keyboard page must be in the project and its ID must be 2. The keyboard page can also be opened with the *Input Widget*.

See in the *Keyboard, Keypad and Message Box* example how the keyboard page is defined:

{keypad_text}



A75.2. Properties

Specific

A75.2.1. Label *EXPRESSION (string)*

The label that will be displayed on the keyboard page (e.g. the name of the parameter whose value is entered).

A75.2.2. Initial text *EXPRESSION (string)*

Initial (default) text that will be displayed in the input field.

A75.2.3. Min chars *EXPRESSION (integer)*

Defines the minimum length of the entered text.

A75.2.4. Max chars *EXPRESSION (integer)*

Defines the maximum length of the entered text.

A75.2.5. Password *Boolean*

Used when entering hidden text such as a user's password. When it is enabled, every character will be replaced with * when entered.

General**A75.2.6. Description** *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow**A75.2.7. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A75.2.8. Outputs *Array*

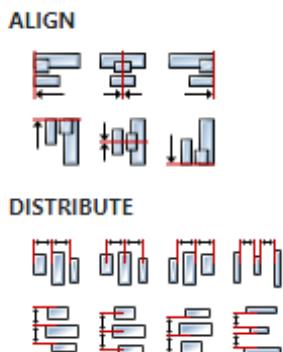
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A75.2.9. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size**A75.2.10. Align and distribute** *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**A75.3. Inputs****A75.3.1. seqin** *SEQ / MANDATORY*

A standard sequence input.

A75.4. Outputs

A75.4.1. result DATA(string) / MANDATORY

Output to which the entered text is sent.

A75.4.2. canceled DATA(null) / OPTIONAL

Flow execution continues through this output if the cancel button is pressed.

A75.5. Examples

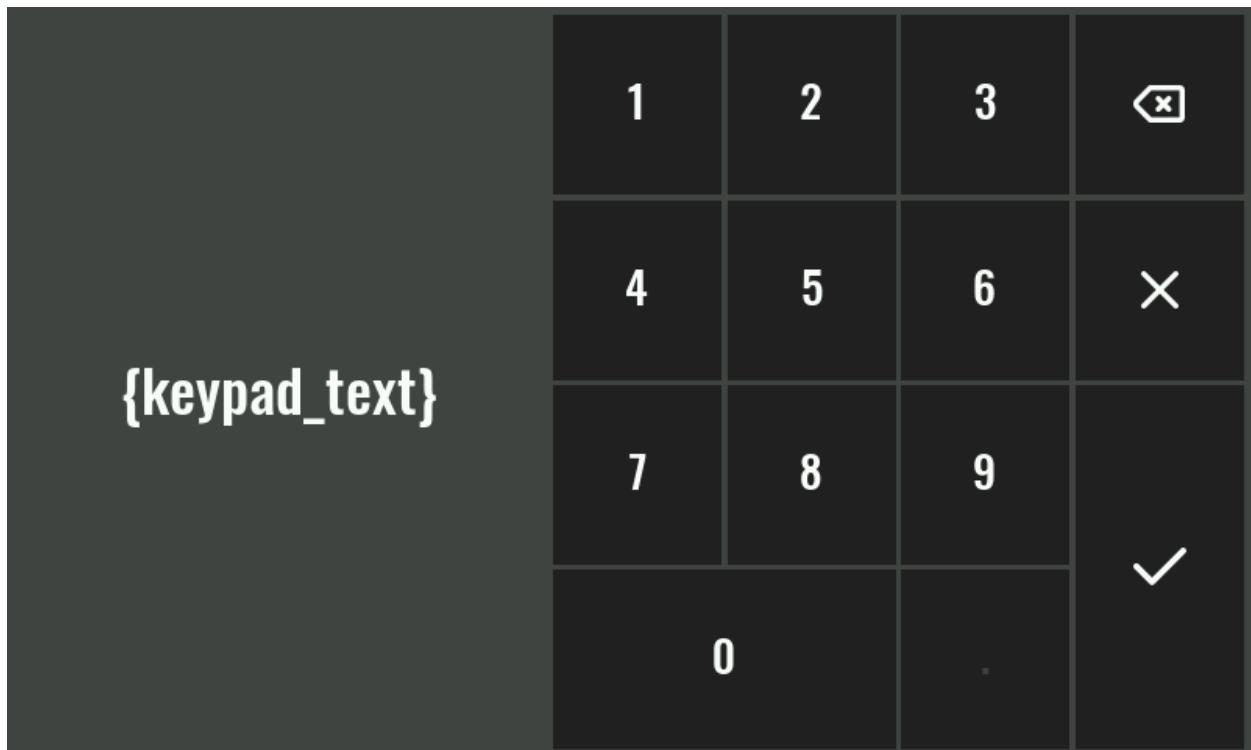
- *Keyboard, Keypad and Message Box*
- *stm32f469i-disco-eez-flow-demo*

A76. ShowKeypad



A76.1. Description

Opens the numeric keypad page for numerical input. The numeric keypad page must be in the project and its ID must be 3. The numeric keypad page can also be opened with the *Input Widget*. See in the *Keyboard, Keypad and Message Box* example how the numeric keypad page is defined:



A76.2. Properties

Specific

A76.2.1. Label *EXPRESSION (string)*

The label that will be displayed on the keyboard page (e.g. the name of the parameter whose value is entered).

A76.2.2. Initial value *EXPRESSION (float)*

Initial (default) number that will be displayed in the input field.

A76.2.3. Min *EXPRESSION (integer)*

The entered number must be greater than or equal to this number.

A76.2.4. Max *EXPRESSION (integer)*

The entered number must be less than or equal to this number.

A76.2.5. Precision *EXPRESSION (float)*

Defines the rounding precision of the entered number. For example if a maximum of two decimal digits is desired, then `0.01` should be entered here.

A76.2.6. Unit *EXPRESSION (string)*

Units that will be displayed when entering a number.

General**A76.2.7. Description** *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow**A76.2.8. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A76.2.9. Outputs *Array*

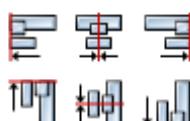
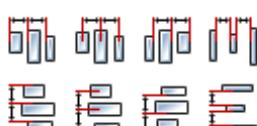
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A76.2.10. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size**A76.2.11. Align and distribute** *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE**

A76.3. Inputs

A76.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A76.4. Outputs

A76.4.1. result DATA(float) / MANDATORY

Output to which the entered numeric value is sent.

A76.4.2. canceled DATA(null) / OPTIONAL

Flow execution continues through this output if the cancel button is pressed.

A76.5. Examples

- *stm32f469i-disco-eez-flow-demo*
- *eyboard, Keypad and Message Box*

A77. ShowMessageBox



A77.1. Description

This Action is used to display *Info*, *Error* or *Question* message boxes.

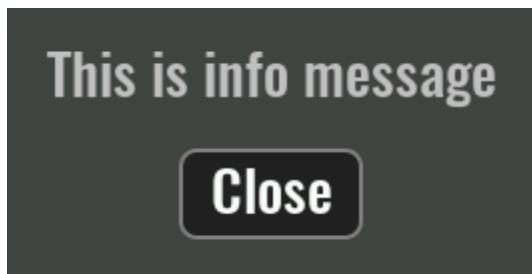
A77.2. Properties

Specific

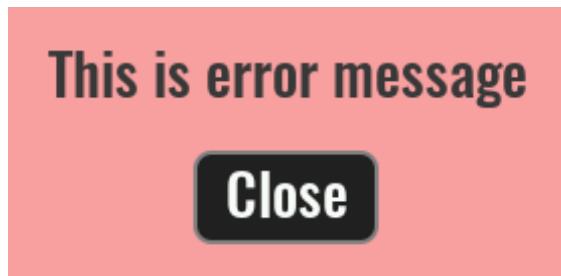
A77.2.1. Message type *Enum*

Defines the message box that will be displayed:

- `Info`



- `Error`



- `Question`



A77.2.2. Message *EXPRESSION (string)*

The content of the message to be displayed.

A77.2.3. Buttons *EXPRESSION (array:string)*

This property needs to be defined only for the *Question* message box. An array of strings is expected here, where each string is mapped to a button, eg `["Save", "Don't Save", "Cancel"]`. It is necessary to add one output in the "Flow - Outputs" section for each button, through which the Flow execution will continue if that button is pressed.

General**A77.2.4. Description** *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow**A77.2.5. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A77.2.6. Outputs *Array*

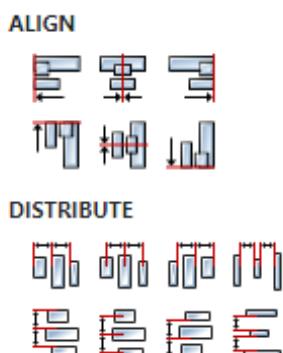
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A77.2.7. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size**A77.2.8. Align and distribute** *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**A77.3. Inputs****A77.3.1. seqin** *SEQ / MANDATORY*

A standard sequence input.

A77.4. Outputs

A77.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A77.5. Examples

- *Keyboard, Keypad and Message Box*

A78. ShowPage



A78.1. Description

This Action sets a new active page: the previous page will be hidden and the new page will be displayed.

A78.2. Properties

Specific

A78.2.1. Page *ObjectReference*

The name of the new page to be displayed.

General

A78.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A78.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A78.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

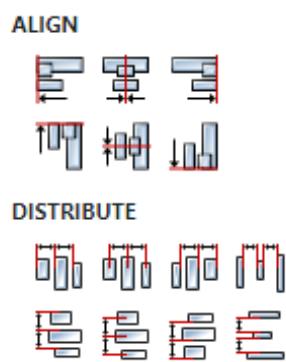
A78.2.5. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A78.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A78.3. Inputs

A78.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A78.4. Outputs

A78.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A79. SortArray



A79.1. Description

It sorts the array variable and returns the result through the data output: it does not do in-place sorting, i.e. it does not modify the content of the array variable. Allowed array types are:

- `array:integer`
- `array:float`
- `array:double`
- `array:struct`

If an array of type `array:struct` is sorted, then the `Structure name` and `Structure field name` by which it is sorted must also be specified.

There are also two options: whether Ascending/Descending sorting is desired and whether letter case is ignored if strings are sorted.

A79.2. Properties

Specific

A79.2.1. Array *EXPRESSION (array:any)*

Array variable to be sorted.

A79.2.2. Structure name *ObjectReference*

Select the name of the structure here when the array is a variable of type `array:struct`.

A79.2.3. Structure field name *Enum*

Select the name of the field to be sorted by if the array is a variable of type `array:struct`.

A79.2.4. Ascending *Boolean*

Sorting mode selection (ascending if enabled, otherwise descending).

A79.2.5. Ignore case *Boolean*

Specifies whether letter case is ignored if strings are sorted or not.

General

A79.2.6. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A79.2.7. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A79.2.8. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

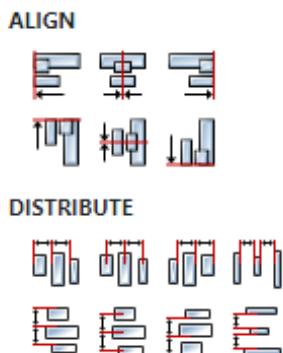
A79.2.9. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A79.2.10. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A79.3. Inputs

A79.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A79.4. Outputs

A79.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A79.4.2. result DATA(any) / MANDATORY

Output through which the sorted array is passed.

A80. Start



A80.1. Description

This action is executed first when Flow is started. Connect the output from this action to the first next action you want to perform.

A80.2. Properties

General

A80.2.1. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A80.2.2. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A80.2.3. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

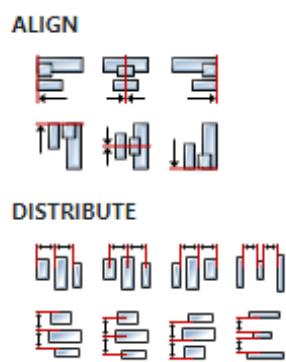
A80.2.4. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A80.2.5. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A80.3. Inputs

A80.4. Outputs

A80.4.1. seqout SEQ / MANDATORY

Connect this output to the action you want to be executed first when the Flow starts.

A81. SwitchCase



A81.1. Description

The expressions added to the `Cases` list are evaluated one by one, starting from the first one in the list. The `Then output` of the first expression whose evaluation result will be `true` will be used for the output on which the Flow execution will continue. The value `true` will be passed to that output unless a `With value` expression is defined.

During Flow execution, it may happen that none of the specified cases in the list returns `true` during evaluation. To prevent this from happening and stop further execution of the Flow, a case can be added at the end of the list in which `true` will be entered in the `When` parameter so that the result of the evaluation will always be true and it will be possible to exit through its output.

A81.2. Properties

Specific

A81.2.1. Cases Array

Each element of this list contains:

- `When` - an expression that is evaluated to see if it is `true`.
- `Then output` - the name of the output through which the execution of the Flow continues if the result of the evaluation of expression `When` is `true`.
- `With value` - optional parameter, if set as an expression, is passed to the output, if not defined `true` is passed.

General

A81.2.2. Description String

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A81.2.3. Inputs Array

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A81.2.4. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

A81.2.5. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that out-

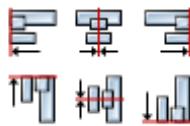
put. The data that will be passed through that output is the textual description of the error.

Position and size

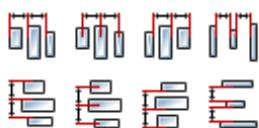
A81.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A81.3. Inputs

A81.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A81.4. Outputs

A81.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A82. TabulatorAction



A82.1. Description

Executes an action on the given tabulator widget.

A82.2. Properties

Specific

A82.2.1. Widget *EXPRESSION (widget)*

Reference to the Tabulator widget. See `Output widget handle` property to find out how to obtain this reference.

A82.2.2. Tabulator action *Enum*

Action to be executed. It can be "Get sheet data" or "Download".

A82.2.3. Lookup *EXPRESSION (string)*

If Tabulator action is "Get sheet data" then this is the sheet name you want to retrieve, if empty it will retrieve the currently active sheet.

A82.2.4. File name *EXPRESSION (string)*

If Tabulator action is "Download" then this is default download file name.

A82.2.5. Download type *Enum*

If Tabulator action is "Download" then this is type of file you want to download. Available options are: "CSV", "JSON" or "HTML".

General

A82.2.6. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A82.2.7. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A82.2.8. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

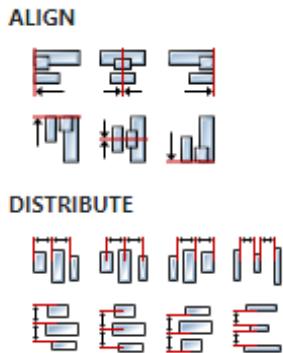
A82.2.9. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A82.2.10. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A82.3. Inputs

A82.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A82.4. Outputs

A82.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A82.5. Examples

- *Tabulator Examples*

A83. TCPConnect



A83.1. Description

Connects to the TCP server.

A83.2. Properties

Specific

A83.2.1. Socket *ASSIGNABLE EXPRESSION (object:TCPsocket)*

Socket object of type `object:TCPsocket` to be created and initialized.

A83.2.2. IP Address *EXPRESSION (object:string)*

IP address of the server.

A83.2.3. Port *EXPRESSION (object:number)*

TCP port on which server accepts connections.

General

A83.2.4. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A83.2.5. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A83.2.6. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A83.2.7. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

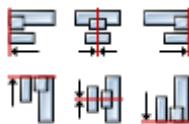
Position and size

A83.2.8. Align and distribute *Any*

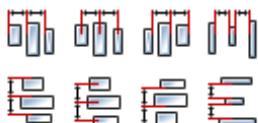
Alignment icons and component distribution. Alignment icons appear when two or more compo-

nents are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A83.3. Inputs

A83.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A83.4. Outputs

A83.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

A83.5. Examples

- *TCP Client*
- *TCP Server*

A84. TCPDisconnect



A84.1. Description

Performs disconnection from the TCP server, after which Flow execution continues through `seqout` output.

A84.2. Properties

Specific

A84.2.1. Socket *EXPRESSION (object:TCPSocket)*

The socket object that will be disconnected.

General

A84.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A84.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A84.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

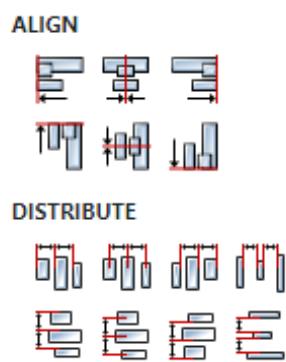
A84.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A84.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A84.3. Inputs

A84.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A84.4. Outputs

A84.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A84.5. Examples

- *TCP Client*
- *TCP Server*

A85. TCPEvent



A85.1. Description

With this Action we can add one or more event handlers that can be received by the TCP socket.

A85.2. Properties

Specific

A85.2.1. Socket *EXPRESSION (object:TCP Socket)*

The socket object on which we want to listen to events.

A85.2.2. Event handlers *Array*

List of events to be handled. For each item in the list, it will be necessary to select `Event`, `Handler type` and optionally `Action`. `Event` is the type of event we want to handle and the possible values are:

- `Ready` – Emitted when a socket is ready to be used.
- `Data` – Emitted when data is received.
- `Close` – Emitted once the socket is fully closed.
- `End` – Emitted when the other end of the socket signals the end of transmission, thus ending the readable side of the socket.
- `Error` – Emitted when an error occurs. The 'close' event will be called directly following this event.
- `Timeout` – Emitted if the socket times out from inactivity. This is only to notify that the socket has been idle. The user must manually disconnect the connection.

`Handler type` can be `Flow` or `Action`. If `Flow` is selected then an output will be added through which the Flow execution continues if the event is sent. If `Action` is selected, then `Action` must also be set, i.e. the name of the User action that is executed when the event is received.

General

A85.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A85.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A85.2.5. Outputs *Array*

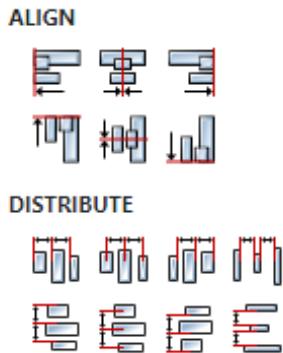
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

A85.2.6. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size**A85.2.7. Align and distribute Any**

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**A85.3. Inputs****A85.3.1. seqin SEQ / OPTIONAL**

A standard sequence input.

A85.4. Outputs**A85.4.1. seqout SEQ / OPTIONAL**

A standard sequence output.

A85.5. Examples

- *TCP Client*
- *TCP Server*

A86. TCPListen



A86.1. Description

Binds to TCP port and listen for the incomming connections.

A86.2. Properties

Specific

A86.2.1. Port *EXPRESSION (object:number)*

Port to which we bind.

A86.2.2. IP Address *EXPRESSION (object:string)*

Address to which we bind.

A86.2.3. Max. Connections *EXPRESSION (object:number)*

Max allowed active incoming connections.

General

A86.2.4. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A86.2.5. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A86.2.6. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A86.2.7. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

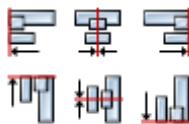
Position and size

A86.2.8. Align and distribute *Any*

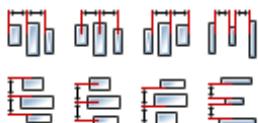
Alignment icons and component distribution. Alignment icons appear when two or more compo-

nents are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



A86.3. Inputs

A86.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A86.3.2. end SEQ / OPTIONAL

Stop listening and unbind from the port. Will trigger `close` output.

A86.4. Outputs

A86.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A86.4.2. connection DATA(*object:TCPSocket*) / MANDATORY

Output to which the socket for the incoming connection is sent.

A86.4.3. close DATA(*string*) / OPTIONAL

Will trigger when listening stops.

A86.5. Examples

- *TCP Client*
- *TCP Server*

A87. TCPWrite



A87.1. Description

Write data to the TCP socket.

A87.2. Properties

Specific

A87.2.1. Socket *EXPRESSION (object:TCP Socket)*

The socket to which data is written.

A87.2.2. Data *EXPRESSION (object:string)*

Data to be written.

General

A87.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A87.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A87.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

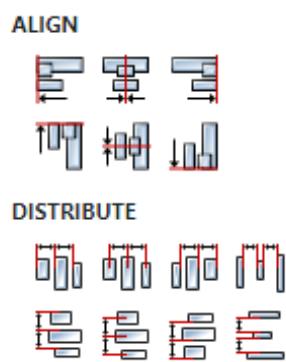
A87.2.6. Catch error *Boolean*

If this checkbox is enabled then an *@Error* output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A87.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A87.3. Inputs

A87.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A87.4. Outputs

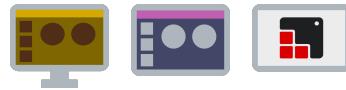
A87.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A87.5. Examples

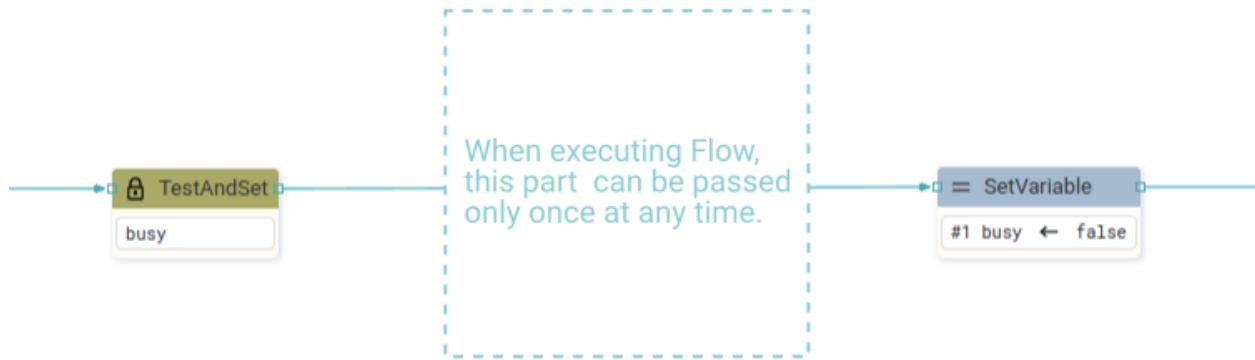
- *TCP Client*
- *TCP Server*

A88. TestAndSet



A88.1. Description

It tests the `boolean` variable and if it is `false` then it is set to `true` and output to the sequential output (`seqout`), and if it is `true` then it is put back into the Flow execution queue, i.e. this action waits until the variable becomes `false`. This testing and setup is done as a single atomic (non-interruptable) operation, so this Action is suitable for the case when you want to make sure that at some point you only go through a certain part of the Flow once. In that case, this Action should be set before entering that part of the Flow, and at the exit from the Flow, the variable should be set to `false` again with the `SetVariable` Action.



A88.2. Properties

Specific

A88.2.1. Variable `ASSIGNABLE EXPRESSION (boolean)`

The variable to be tested and set.

General

A88.2.2. Description `String`

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A88.2.3. Inputs `Array`

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A88.2.4. Outputs `Array`

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the

output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

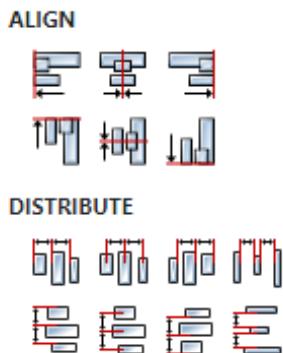
A88.2.5. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A88.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A88.3. Inputs

A88.3.1. seqin SEQ / OPTIONAL

A standard sequential input.

A88.4. Outputs

A88.4.1. seqout SEQ / OPTIONAL

Flow execution continues through this sequential output when the variable becomes `false`.

A88.5. Examples

- Tetris* In the `do_action` User action, which is called when it is detected that some key on the keyboard is pressed, the `TestAndSet` action on the `busy` variable is used at the beginning, and before the exit the `busy` variable is set to `false`. In this way, it is ensured that two Actions are not executed simultaneously.

A89. UDP In



A89.1. Description

Use this action to output message received on specified UDP port.

A89.2. Properties

Specific

A89.2.1. Listen for *Enum*

Select UDP or Multicast mode.

A89.2.2. Group *EXPRESSION (string)*

If Multicast mode is selected, specify multicast group you whish to join.

A89.2.3. Local interface *EXPRESSION (string)*

Specify local network interface for multicast group. If this option is not specified, the operating system will choose one interface and will add membership to it.

A89.2.4. On port *EXPRESSION (integer)*

The port from which we want to receive messages.

A89.2.5. Using *Enum*

Use IPV4 or IPV6 addresses

General

A89.2.6. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A89.2.7. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A89.2.8. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

A89.2.9. Catch error *Boolean*

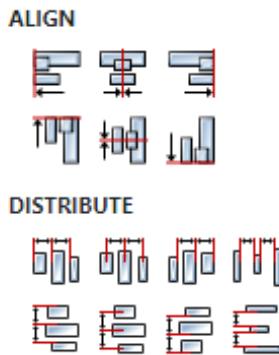
If this checkbox is enabled then an `@Error` output will be added to the component and if an error

occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A89.2.10. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A89.3. Inputs

A89.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A89.4. Outputs

A89.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A89.4.2. message DATA(struct:\$UDPMessages) / MANDATORY

Output to which the received message is sent. The type of message is `struct:$UDPMessages` with following fields:

- `payload`: message payload received as `blob`, use `Blob.toString()` to convert to the string value.
- `address`: remote IP address
- `port`: remote IP port

A89.5. Examples

- *UDP Client*
- *UDP Server*

A90. UDP Out



A90.1. Description

This actions sends message to the designated UDP host and port.

A90.2. Properties

Specific

A90.2.1. Send a *Enum*

Options to send UDP, Multicast or Broadcast message

A90.2.2. To port *EXPRESSION (integer)*

The port to which message is sent.

A90.2.3. Address *EXPRESSION (string)*

The address to which message is sent.

A90.2.4. Group *EXPRESSION (string)*

If Multicast mode is selected, specify multicast group you whish to join.

A90.2.5. Local interface *EXPRESSION (string)*

Specify local network interface for multicast group. If this option is not specified, the operating system will choose one interface and will add membership to it.

A90.2.6. Ipv *Enum*

Use IPV4 or IPV6 addresses

A90.2.7. Bind to *Enum*

Option to bind to the random or fixed port.

A90.2.8. Outport *EXPRESSION (integer)*

If fixed port option is selected then specify fixed port with this property.

A90.2.9. Payload *EXPRESSION (string)*

Message payload to be sent.

General

A90.2.10. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A90.2.11. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive addi-

tional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A90.2.12. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

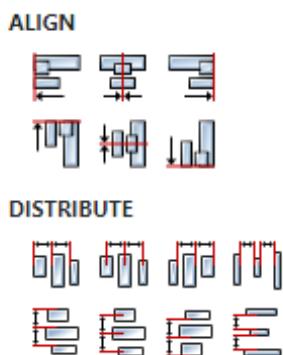
A90.2.13. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A90.2.14. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A90.3. Inputs

A90.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A90.4. Outputs

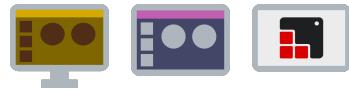
A90.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

A90.5. Examples

- *UDP Client*
- *UDP Server*

A91. Watch



A91.1. Description

This action, for the entire duration of Flow execution, evaluates the default expression in the background and if there is a change in the result, it forwards it to the data output. At the beginning, when the Flow is started, the expression is evaluated and forwarded to the data output, and later only if some change has occurred.

A91.2. Properties

Specific

A91.2.1. Expression *EXPRESSION (any)*

Expression to be evaluated.

General

A91.2.2. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A91.2.3. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

A91.2.4. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

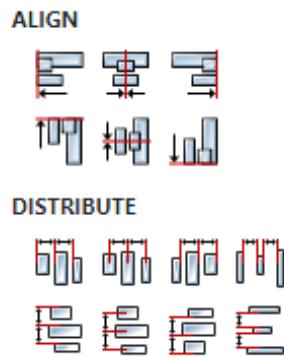
A91.2.5. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A91.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A91.3. Inputs

A91.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A91.4. Outputs

A91.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

A91.4.2. changed *DATA(any) / MANDATORY*

Output through which the value of the evolved expression is passed once at the start and later only if there was some change in the result.

A92. WriteSetting



A92.1. Description

This Action will add the set `Key` to the `.eez-project-runtime-settings` file (it's the same file where persistent variables are saved), or it will update the value with `Value` of that key if it already exists.

NOTE: `WriteSetting` and `ReadSetting` Actions are used to save and retrieve from the `eez-project-runtime-settings` file all those settings that we want to survive the `Dashboard` project restart. It is more convenient to use persistent variables, because in that case we do not have to execute a special Action for saving and retrieving.

A92.2. Properties

Specific

A92.2.1. Key *EXPRESSION (string)*

A string containing the name of the key to be added/updated.

A92.2.2. Value *EXPRESSION (any)*

The value of the key that will be created or updated.

General

A92.2.3. Description *String*

This is the description of the Action component. Description is displayed below the component in the Project editor/viewer. In the main toolbar, it is possible to hide or display descriptions of all components with one click.

Flow

A92.2.4. Inputs *Array*

Additional component inputs that the user can add as desired in order to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

A92.2.5. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

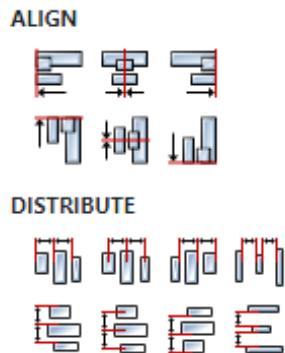
A92.2.6. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

A92.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



A92.3. Inputs

A92.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A92.4. Outputs

A92.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

EEZ Studio

Widgets

Table of Contents

<i>EEZ Studio Widgets</i>	W.1
W1. AnimationImage.....	W.5
W2. Arc.....	W.11
W3. Bar.....	W.17
W4. BarGraph.....	W.25
W5. Bitmap (Dashboard).....	W.29
W6. Bitmap (EEZ-GUI).....	W.33
W7. Button (Dashboard).....	W.37
W8. Button (EEZ-GUI).....	W.41
W9. ButtonGroup.....	W.45
W10. Button (LVGL).....	W.49
W11. ButtonMatrix.....	W.55
W12. Calendar.....	W.61
W13. Canvas.....	W.67
W14. Chart.....	W.73
W15. Checkbox (Dashboard).....	W.79
W16. Checkbox (LVGL).....	W.83
W17. Colorwheel.....	W.89
W18. Container.....	W.95
W19. Container (Dashboard).....	W.99
W20. Container (LVGL).....	W.103
W21. DisplayData.....	W.109
W22. Dropdown (Dashboard).....	W.113
W23. Dropdown (EEZ-GUI).....	W.117
W24. Dropdown (LVGL).....	W.121
W25. EEZChart.....	W.127
W26. Embedded Dashboard.....	W.133
W27. Gauge (Dashboard).....	W.137
W28. Gauge (EEZ-GUI).....	W.141
W29. Grid.....	W.145
W30. Image.....	W.151
W31. Imgbutton.....	W.157
W32. Input (EEZ-GUI).....	W.163
W33. InstrumentTerminal.....	W.167
W34. Keyboard.....	W.171
W35. Label.....	W.177
W36. Led.....	W.183
W37. Line.....	W.189
W38. LineChart (Dashboard).....	W.195
W39. LineChart (EEZ-GUI).....	W.201
W40. List.....	W.207
W41. List (Dashboard).....	W.211
W42. List (LVGL).....	W.215

W43. Lottie.....	W.221
W44. Markdown.....	W.227
W45. Menu.....	W.231
W46. MessageBox.....	W.237
W47. Meter.....	W.243
W48. MultilineText.....	W.249
W49. NumberInput.....	W.253
W50. Panel.....	W.257
W51. Plotly.....	W.263
W52. Progress (Dashboard).....	W.267
W53. Progress (EEZ-GUI).....	W.271
W54. QRCode (Dashboard).....	W.275
W55. QRCode (EEZ-GUI).....	W.279
W56. Radio.....	W.283
W57. Rectangle (Dashboard).....	W.287
W58. Rectangle (EEZ-GUI).....	W.291
W59. Roller (EEZ-GUI).....	W.295
W60. Roller (LVGL).....	W.299
W61. Scale.....	W.305
W62. ScrollBar.....	W.311
W63. Select.....	W.317
W64. Slider (Dashboard).....	W.321
W65. Slider (EEZ-GUI).....	W.325
W66. Slider (LVGL).....	W.329
W67. Span.....	W.335
W68. Spinbox.....	W.341
W69. Spinner (Dashboard).....	W.347
W70. Spinner (LVGL).....	W.351
W71. Switch (Dashboard).....	W.357
W72. Switch (EEZ-GUI).....	W.361
W73. Switch (LVGL).....	W.365
W74. Tab.....	W.371
W75. Table.....	W.377
W76. Tabulator.....	W.383
W77. Tabview.....	W.387
W78. Terminal.....	W.393
W79. Textarea.....	W.397
W80. Text (Dashboard).....	W.403
W81. Text (EEZ-GUI).....	W.407
W82. TextInput.....	W.411
W83. TileView.....	W.415
W84. ToggleButton.....	W.421
W85. UpDown.....	W.425
W86. Window.....	W.429

W1. AnimationImage



W1.1. Description

The animation image is similar to the normal 'Image' object. The only difference is that instead of one source image, you set an array of multiple source images.

You can specify a duration and repeat count.

More info ([link](#))

W1.2. Properties

Specific

W1.2.1. Images *Array*

List of images to be animated.

W1.2.2. Duration *Number*

Duration of animation given in milliseconds.

W1.2.3. Repeat infinite *Boolean*

If enabled than animation will be repeated infinite times.

W1.2.4. Repeat count *Number*

If "Repeat infinite" is disabled then use this property to control how many times animation will be repeated.

General

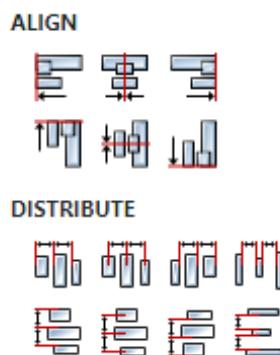
W1.2.5. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W1.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W1.2.7. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W1.2.8. Left** *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W1.2.9. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W1.2.10. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W1.2.11. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W1.2.12. Width *Number*

The width of the component. It is set in pixels.

W1.2.13. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W1.2.14. Height *Number*

The height of the component. It is set in pixels.

W1.2.15. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

W1.2.16. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W1.2.17. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W1.2.18. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W1.2.19. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W1.2.20. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W1.2.21. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W1.2.22. Click focusable *Boolean*

Add focused state to the object when clicked.

W1.2.23. Checkable *Boolean*

Toggle checked state when the object is clicked.

W1.2.24. Scrollable *Boolean*

Make the object scrollable.

W1.2.25. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W1.2.26. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W1.2.27. Scroll one *Boolean*

Allow scrolling only one snappable children.

W1.2.28. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W1.2.29. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W1.2.30. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W1.2.31. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W1.2.32. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W1.2.33. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W1.2.34. Event bubble Boolean

Propagate the events to the parent too.

W1.2.35. Gesture bubble Boolean

Propagate the gestures to the parent.

W1.2.36. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W1.2.37. Ignore layout Boolean

Make the object positionable by the layouts.

W1.2.38. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W1.2.39. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W1.2.40. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W1.2.41. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States

W1.2.42. Checked *EXPRESSION (boolean)*

Toggled or checked state.

W1.2.43. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W1.2.44. Disabled *EXPRESSION (boolean)*

Disabled state

W1.2.45. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W1.2.46. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W1.2.47. Pressed *Boolean*

Being pressed.

Events**W1.2.48. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W1.2.49. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W1.2.50. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W1.2.51. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W2. Arc



W2.1. Description

The Arc consists of a background and a foreground arc. The foreground (indicator) can be touch-adjusted.

More info ([link](#))

W2.2. Properties

Specific

W2.2.1. Range min *EXPRESSION (integer)*

The minimum value that can be selected by the `Value` property.

W2.2.2. Range min type *Enum*

Defines whether the `Range min` will be given as a Literal or as an Expression.

W2.2.3. Range max *EXPRESSION (integer)*

The maximum value that can be selected by the `Value` property.

W2.2.4. Range max type *Enum*

Defines whether the `Range max` will be given as a Literal or as an Expression.

W2.2.5. Value *EXPRESSION (integer)*

The value, in the range given by `Range min` and `Range max`, which sets the size of foreground (indicator) arc relative to the background arc.

W2.2.6. Value type *Enum*

Defines whether the `Value` will be given as a Literal or as an Expression.

W2.2.7. Bg start angle *Number*

Start angle of the background arc. Zero degrees is at the middle right (3 o'clock) of the object and the degrees are increasing in clockwise direction. The angles should be in the `[0, 360]` range.

W2.2.8. Bg end angle *Number*

End angle of the background arc. Zero degrees is at the middle right (3 o'clock) of the object and the degrees are increasing in clockwise direction. The angles should be in the `[0, 360]` range.

W2.2.9. Mode *Enum*

The arc can be one of the following modes:

- `NORMAL` – The indicator arc is drawn from the minimum value to the current.
- `REVERSE` – The indicator arc is drawn counter-clockwise from the maximum value to the current.
- `SYMMETRICAL` – The indicator arc is drawn from the middle point to the current value.

W2.2.10. Rotation *Number*

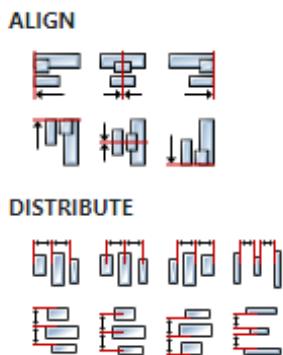
An offset to the 0 degree position.

General**W2.2.11. Name** *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size**W2.2.12. Align and distribute** *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W2.2.13. Center widget** *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W2.2.14. Left** *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W2.2.15. Left unit *Enum*

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W2.2.16. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W2.2.17. Top unit *Enum*

The following options are available:

- px – Top is set in pixels.
- % – The top is set as a percentage in relation to the parent height.

W2.2.18. Width *Number*

The width of the component. It is set in pixels.

W2.2.19. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W2.2.20. Height *Number*

The height of the component. It is set in pixels.

W2.2.21. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W2.2.22. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W2.2.23. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W2.2.24. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W2.2.25. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W2.2.26. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W2.2.27. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W2.2.28. Click focusable *Boolean*

Add focused state to the object when clicked.

W2.2.29. Checkable *Boolean*

Toggle checked state when the object is clicked.

W2.2.30. Scrollable *Boolean*

Make the object scrollable.

W2.2.31. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W2.2.32. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W2.2.33. Scroll one *Boolean*

Allow scrolling only one snappable children.

W2.2.34. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W2.2.35. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W2.2.36. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W2.2.37. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W2.2.38. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W2.2.39. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W2.2.40. Event bubble Boolean

Propagate the events to the parent too.

W2.2.41. Gesture bubble Boolean

Propagate the gestures to the parent.

W2.2.42. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W2.2.43. Ignore layout Boolean

Make the object positionable by the layouts.

W2.2.44. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W2.2.45. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W2.2.46. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W2.2.47. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W2.2.48. Checked EXPRESSION (boolean)**

Toggled or checked state.

W2.2.49. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W2.2.50. Disabled EXPRESSION (boolean)

Disabled state

W2.2.51. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W2.2.52. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W2.2.53. Pressed Boolean

Being pressed.

Events**W2.2.54. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W2.2.55. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W2.2.56. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W2.2.57. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W2.3. Examples

- *LVGL Widgets Demo*
- *Smart Home*

W3. Bar



W3.1. Description

The bar object has a background and an indicator on it. The width of the indicator is set according to the current value of the bar.

Vertical bars can be created if the width of the object is smaller than its height.

Not only the end, but also the start value of the bar can be set, which changes the start position of the indicator.

More info ([link](#))

W3.2. Properties

Specific

W3.2.1. Min Number

The minimum value that `Value` and `Value start` can contain.

W3.2.2. Max Number

The maximum value that `Value` and `Value start` can contain.

W3.2.3. Mode Enum

Bar mode options:

- `NORMAL` – A normal bar.
- `SYMMETRICAL` – Draw the indicator from the zero value to current value. Requires a negative minimum range and positive maximum range.
- `RANGE` – Allows setting the start value (`Value start` property) and end value (`Value` property).

W3.2.4. Value EXPRESSION (integer)

The end value on the bar.

W3.2.5. Value type Enum

Select between `Literal` and `Expression`. If `Expression` is selected then `Value` can be evaluated from the expression.

W3.2.6. Value start EXPRESSION (integer)

The start value on the bar if `RANGE` mode is selected.

W3.2.7. Value start type Enum

Select between `Literal` and `Expression`. If `Expression` is selected then `Value start` can be evaluated from the expression.

W3.2.8. Enable animation Boolean

If enabled then value change will be animated. Duration of animation is controlled with the style property ("Miscellaneous" section) "Anim time" in LVGL 8.4 or "Anim duration" in LVGL 9.1.

General

W3.2.9. Name *String*

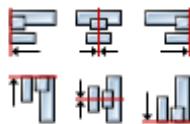
Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

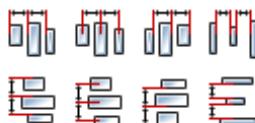
W3.2.10. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



W3.2.11. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W3.2.12. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2`.

W3.2.13. Left unit Enum

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W3.2.14. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W3.2.15. Top unit Enum

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W3.2.16. Width Number

The width of the component. It is set in pixels.

W3.2.17. Width unit Enum

The following options are available:

- `px` – Width is given in pixels.

- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W3.2.18. Height *Number*

The height of the component. It is set in pixels.

W3.2.19. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W3.2.20. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W3.2.21. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W3.2.22. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W3.2.23. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W3.2.24. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W3.2.25. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W3.2.26. Click focusable *Boolean*

Add focused state to the object when clicked.

W3.2.27. Checkable *Boolean*

Toggle checked state when the object is clicked.

W3.2.28. Scrollable *Boolean*

Make the object scrollable.

W3.2.29. Scroll elastic Boolean

Allow scrolling inside but with slower speed.

W3.2.30. Scroll momentum Boolean

Make the object scroll further when "thrown".

W3.2.31. Scroll one Boolean

Allow scrolling only one snappable children.

W3.2.32. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W3.2.33. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W3.2.34. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W3.2.35. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W3.2.36. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W3.2.37. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W3.2.38. Event bubble Boolean

Propagate the events to the parent too.

W3.2.39. Gesture bubble Boolean

Propagate the gestures to the parent.

W3.2.40. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W3.2.41. Ignore layout Boolean

Make the object positionable by the layouts.

W3.2.42. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W3.2.43. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W3.2.44. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars

- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W3.2.45. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States

W3.2.46. Checked *EXPRESSION (boolean)*

Toggled or checked state.

W3.2.47. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W3.2.48. Disabled *EXPRESSION (boolean)*

Disabled state

W3.2.49. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W3.2.50. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W3.2.51. Pressed *Boolean*

Being pressed.

Events

W3.2.52. Event handlers *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W3.2.53. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W3.2.54. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W3.2.55. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W3.3. Examples

- *Dashboard Widgets Demo*

W4. BarGraph



W4.1. Description

This Widget displays the default value through the `Data` property as a bar and as text (if selected). Also, if set, it will show two lines at the default positions (`Threshold1` and `Threshold2`), e.g. to mark some critical values.

W4.2. Properties

Specific

W4.2.1. Data *EXPRESSION (any)*

This is the value within the range `[Min, Max]` for which the bar and text will be rendered.

W4.2.2. Orientation *Enum*

Defines the orientation of the Widget, the following options are available:

- `Left right` – as the value set through `Data` increases from Min to Max, the bar inside the graph also increases from the left side to the right side.
- `Right left` – the bar grows from right to left
- `Top bottom` – the bar grows from top to bottom
- `Bottom top` – the bar grows from bottom to top

W4.2.3. Display value *Boolean*

When checked, `Data` value will also be displayed as text.

W4.2.4. Threshold1 *EXPRESSION (any)*

An optional value within the range `[Min, Max]` at whose position a line will be drawn in the default style (`Threshold1`). It is used to mark some critical/important value within the bar graph.

W4.2.5. Threshold2 *EXPRESSION (any)*

An optional value within the range `[Min, Max]` at whose position a line will be drawn in the default style (`Threshold2`). It is used to mark some critical/important value within the bar graph.

W4.2.6. Min *EXPRESSION (any)*

The minimum value that `Data` can contain.

W4.2.7. Max *EXPRESSION (any)*

The maximum value that `Data` can contain.

W4.2.8. Refresh rate *EXPRESSION (any)*

Similar to the case of the `DisplayData` Widget, it defines the speed at which the text will be refreshed.

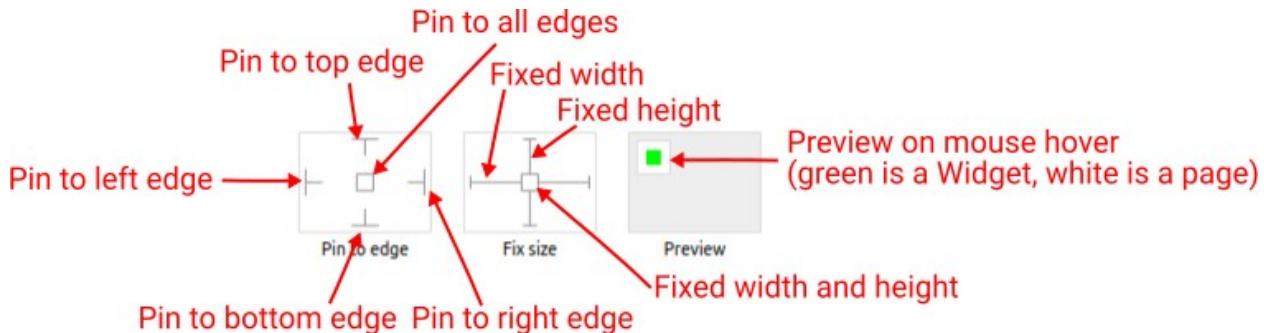
W4.2.9. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W4.2.10. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

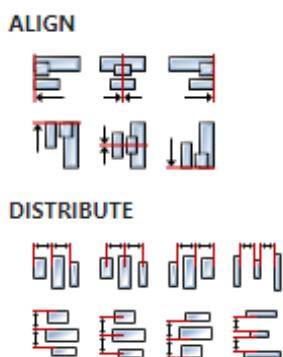
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W4.2.11. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W4.2.12. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W4.2.13. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W4.2.14. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W4.2.15. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W4.2.16. Width *Number*

The width of the component. It is set in pixels.

W4.2.17. Height *Number*

The height of the component. It is set in pixels.

W4.2.18. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W4.2.19. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W4.2.20. Default** *Object*

Style used when rendering of the Widget.

W4.2.21. Text *Object*

Style used to render the text inside the Widget.

W4.2.22. Threshold1 *Object*

Style used to render the `Threshold1` value.

W4.2.23. Threshold2 *Object*

Style used to render the `Threshold2` value.

Events**W4.2.24. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.

- **Handler type** – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W4.2.25. Inputs `Array`

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W4.2.26. Outputs `Array`

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W4.2.27. Catch error `Boolean`

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W4.3. Examples

- `eez-gui-widgets-demo`

W5. Bitmap (Dashboard)



W5.1. Description

This Widget displays a bitmap. If we know in advance which bitmap we want to display, then it is necessary to use the `Bitmap` property, where the selection is called the bitmap, and if the bitmap is known only during execution because, for example, it comes from some variable, then it is necessary to use the `Data` property.

W5.2. Properties

Specific

W5.2.1. Data *EXPRESSION (any)*

There are several options for choosing which bitmap to display:

- If the default value is of type `integer` then it is the index of the bitmap to be displayed. It is necessary to use the functions `Flow.getBitmapIndex({<bitmapName>})`, which receives `bitmapName`, i.e. the name of the bitmap, and returns the index of the bitmap. In this way, we can choose or change which bitmap will be displayed in the runtime, because, for example, '`bitmapName`' can come from a variable.
- If the default value is of type `string` then it is assumed that the bitmap is encoded according to the Data URI Scheme ([link](#)) rules.
- If the default value is of type `blob` then the bitmap is defaulted to its binary notation (see *Screen Capture* example).

W5.2.2. Bitmap *ObjectReference*

The name of the bitmap to be displayed.

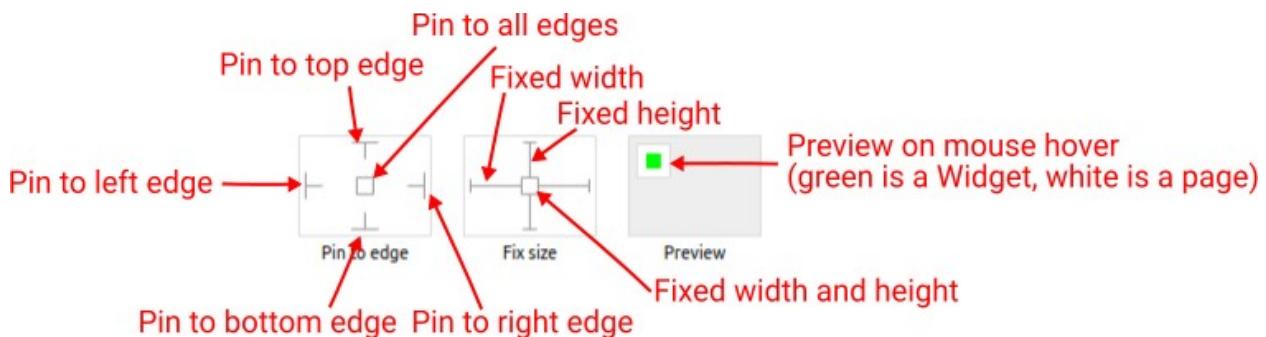
W5.2.3. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W5.2.4. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge

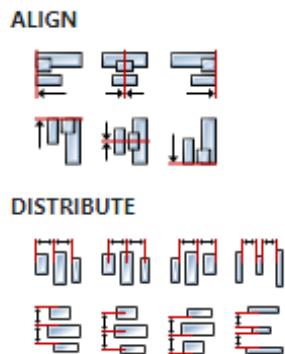
of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W5.2.5. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W5.2.6. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W5.2.7. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W5.2.8. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W5.2.9. Width Number

The width of the component. It is set in pixels.

W5.2.10. Height Number

The height of the component. It is set in pixels.

W5.2.11. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W5.2.12. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W5.2.13. Default** *Object*

Style used when rendering the background of the Widget.

Events**W5.2.14. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W5.2.15. Output widget handle** *Boolean*

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W5.2.16. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W5.2.17. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W5.2.18. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error

occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W5.3. Examples

- *Dashboard Widgets Demo*
- *Screen Capture*

W6. Bitmap (EEZ-GUI)



W6.1. Description

This Widget displays a bitmap.

W6.2. Properties

Specific

W6.2.1. Data *EXPRESSION (integer)*

Index of the bitmap to be displayed. It is necessary to use the functions `Flow.getBitmapIndex({<bitmapName>})`, which receives `bitmapName`, i.e. the name of the bitmap, and returns the index of the bitmap. In this way, we can choose or change which bitmap will be displayed in the runtime, because, for example, '`bitmapName`' can come from a variable.

W6.2.2. Bitmap *ObjectReference*

The name of the bitmap to be displayed.

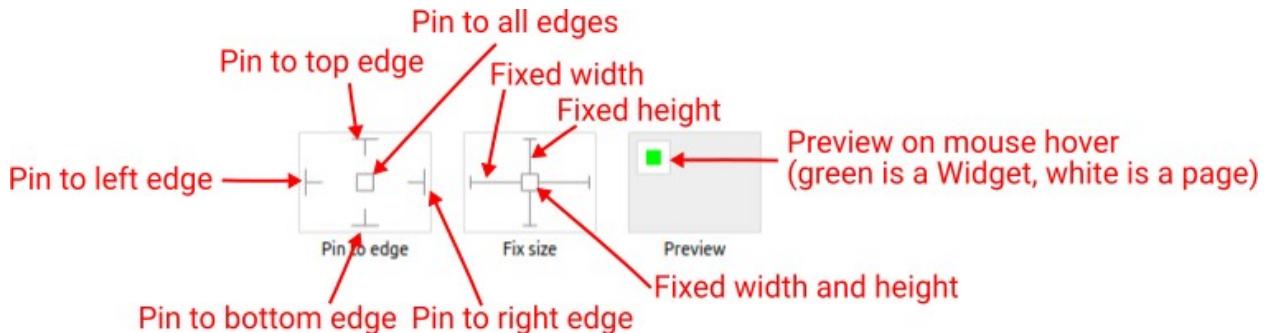
W6.2.3. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W6.2.4. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

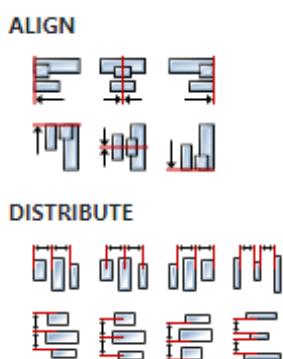
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W6.2.5. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W6.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W6.2.7. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W6.2.8. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W6.2.9. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W6.2.10. Width *Number*

The width of the component. It is set in pixels.

W6.2.11. Height *Number*

The height of the component. It is set in pixels.

W6.2.12. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W6.2.13. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W6.2.14. Default** *Object*

Style used when rendering the background of the Widget.

Events**W6.2.15. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W6.2.16. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W6.2.17. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

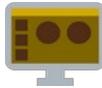
W6.2.18. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W6.3. Examples

- *eez-gui-widgets-demo*

W7. Button (Dashboard)



W7.1. Description

This Widget is used when an action needs to be performed via the GUI, clicking on it generates a `CLICKED` event, which in this case is added to the list of event handlers by default. The widget has two states enabled and disabled, which is set via the `Enabled` property. Each state has its own style, `Default` style for the enabled state and `Disabled` style for the disabled state.

W7.2. Properties

Specific

W7.2.1. Label *EXPRESSION (any)*

The text that will be displayed inside the button.

W7.2.2. Enabled *EXPRESSION (any)*

If it is true, then the button is enabled, otherwise it will be disabled.

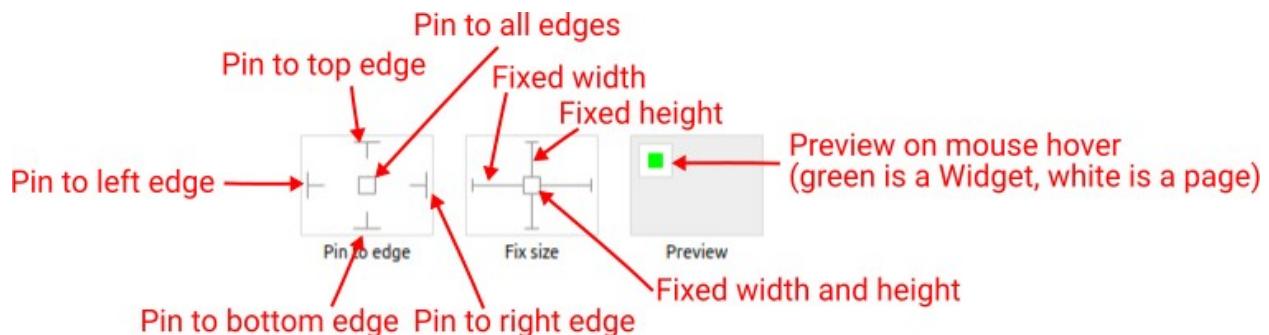
W7.2.3. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W7.2.4. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



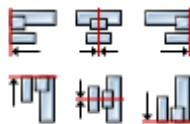
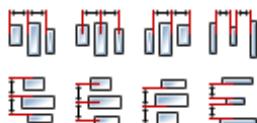
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

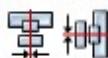
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W7.2.5. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W7.2.6. Center widget** Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W7.2.7. Left** Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W7.2.8. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W7.2.9. Width Number

The width of the component. It is set in pixels.

W7.2.10. Height Number

The height of the component. It is set in pixels.

W7.2.11. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W7.2.12. Tab title** EXPRESSION (string)

If this widget is a child of a container with layout set to Docking Manager, use this property to set the title of the tab that contains this widget.

Style

W7.2.13. Default Object

Style to be used for rendering if the Widget is enabled.

W7.2.14. Disabled Object

Style to be used for rendering if the Widget is disabled.

Events**W7.2.15. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W7.2.16. Output widget handle Boolean**

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W7.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W7.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W7.2.19. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W7.3. Examples

- *eez-gui-widgets-demo*

W8. Button (EEZ-GUI)



W8.1. Description

This Widget is used when an action needs to be performed via the GUI, clicking on it generates a `CLICKED` event, which in this case is added to the list of event handlers by default. The widget has two states enabled and disabled, which is set via the `Enabled` property. Each state has its own style, `Default` style for the enabled state and `Disabled` style for the disabled state.

W8.2. Properties

Specific

W8.2.1. Label *EXPRESSION (any)*

The text that will be displayed inside the button.

W8.2.2. Enabled *EXPRESSION (any)*

If it is true, then the button is enabled, otherwise it will be eisabled.

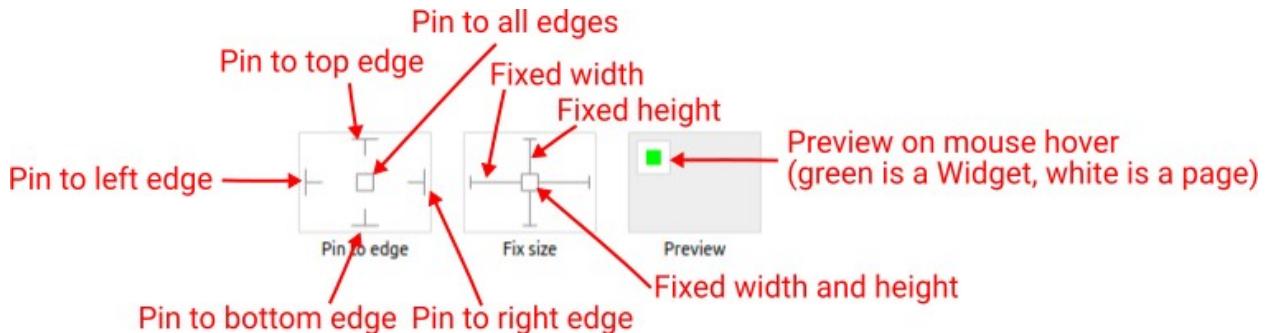
W8.2.3. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W8.2.4. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

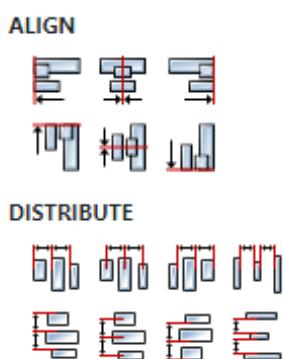
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W8.2.5. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W8.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W8.2.7. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W8.2.8. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W8.2.9. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W8.2.10. Width *Number*

The width of the component. It is set in pixels.

W8.2.11. Height *Number*

The height of the component. It is set in pixels.

W8.2.12. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W8.2.13. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W8.2.14. Default** *Object*

Style to be used for rendering if the Widget is enabled.

W8.2.15. Disabled *Object*

Style to be used for rendering if the Widget is disabled.

Events**W8.2.16. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W8.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W8.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W8.2.19. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W8.3. Examples

- `eez-gui-widgets-demo`

W9. ButtonGroup



W9.1. Description

Shows a group of buttons. The total number of buttons and their labels are defined with `Button labels`. Only one of those buttons can be selected, which is defined by the `Selected button` item. If the button is selected, then `Selected` style is used, otherwise `Default` style is used when rendering an individual button.

W9.2. Properties

Specific

W9.2.1. Button labels *EXPRESSION (any)*

Specifies the labels of all buttons. The number of elements in this string array defines how many buttons will be displayed.

W9.2.2. Selected button *EXPRESSION (any)*

Determines which button is selected. It is a zero-based integer, which means that if its value is 0, the first button will be selected, if its value is 1, the second button will be selected, etc. If we want no button to be selected, we will use the value -1.

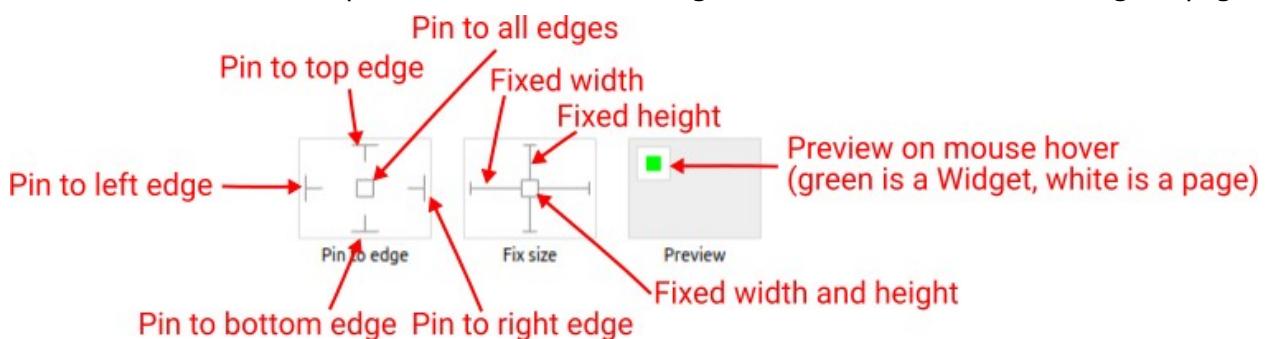
W9.2.3. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W9.2.4. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge*.

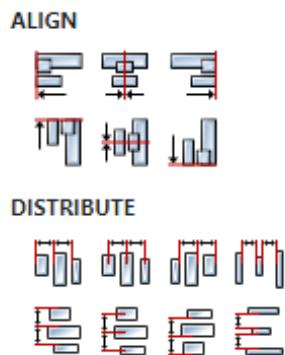
and *Fix width*.

W9.2.5. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W9.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W9.2.7. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W9.2.8. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W9.2.9. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W9.2.10. Width Number

The width of the component. It is set in pixels.

W9.2.11. Height Number

The height of the component. It is set in pixels.

W9.2.12. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W9.2.13. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W9.2.14. Default Object

Style is used to render a button that is not selected.

W9.2.15. Selected Object

Style used to render the selected button.

Events

W9.2.16. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W9.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W9.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W9.2.19. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W9.3. Examples

- [eez-gui-widgets-demo](#)

W10. Button (LVGL)



W10.1. Description

This Widget is used when an action needs to be performed via the GUI, clicking on it generates a `CLICKED` event.

More info ([link](#))

W10.2. Properties

General

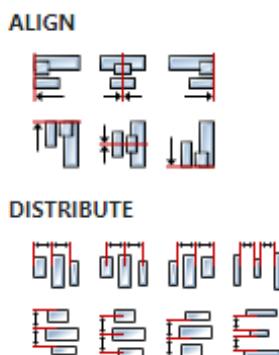
W10.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W10.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W10.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W10.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W10.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W10.2.6. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W10.2.7. Top unit Enum

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W10.2.8. Width Number

The width of the component. It is set in pixels.

W10.2.9. Width unit Enum

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W10.2.10. Height Number

The height of the component. It is set in pixels.

W10.2.11. Height unit Enum

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W10.2.12. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W10.2.13. Use style ObjectReference

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W10.2.14. Hidden EXPRESSION (boolean)

Make the object hidden.

W10.2.15. Hidden flag type Enum

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W10.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W10.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W10.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W10.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W10.2.20. Scrollable *Boolean*

Make the object scrollable.

W10.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W10.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W10.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W10.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W10.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W10.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W10.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W10.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W10.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W10.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W10.2.31. Gesture bubble *Boolean*

Propagate the gestures to the parent.

W10.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W10.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W10.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W10.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W10.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W10.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States

W10.2.38. Checked EXPRESSION (boolean)

Toggled or checked state.

W10.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W10.2.40. Disabled EXPRESSION (boolean)

Disabled state

W10.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W10.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W10.2.43. Pressed Boolean

Being pressed.

Events

W10.2.44. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W10.2.45. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W10.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W10.2.47. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W10.3. Examples

- LVGL Widgets Demo

W11. ButtonMatrix



W11.1. Description

The Button Matrix object is a lightweight way to display multiple buttons in rows and columns. More info ([link](#))

W11.2. Properties

Specific

W11.2.1. Buttons *Array*

List of buttons. Each button has the following properties:

- New line: if enabled then this is not actual button, but it introduces line break in button matrix.
- Text: Label of the button
- Width: The buttons' width can be set relative to the other button in the same row. E.g. in a line with two buttons: btnA, width = 1 and btnB, width = 2, btnA will have 33 % width and btnB will have 66 % width.
- HIDDEN Makes a button hidden (hidden buttons still take up space in the layout, they are just not visible or clickable)
- NO_REPEAT Disable repeating when the button is long pressed
- DISABLED Makes a button disabled Like LV_STATE_DISABLED on normal objects
- CHECKABLE Enable toggling of a button. I.e. LV_STATE_CHECKED will be added/removed as the button is clicked
- CHECKED Make the button checked. It will use the LV_STATE_CHECKED styles.
- CLICK_TRIG Enabled: send LV_EVENT_VALUE_CHANGE on CLICK, Disabled: send LV_EVENT_VALUE_CHANGE on PRESS
- POPOVER Show the button label in a popover when pressing this key
- RECOLOR Enable recoloring of button texts with #. E.g. "It's #ff0000 red#"
- CUSTOM_1 Custom free to use flag
- CUSTOM_2 Custom free to use flag

W11.2.2. One check *Boolean*

The "One check" feature can be enabled to allow only one button to be checked at a time.

General

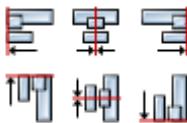
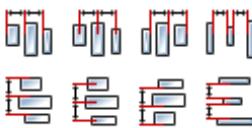
W11.2.3. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W11.2.4. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W11.2.5. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

W11.2.6. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W11.2.7. Left unit Enum

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W11.2.8. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W11.2.9. Top unit Enum

The following options are available:

- px – Top is set in pixels.
- % – The top is set as a percentage in relation to the parent height.

W11.2.10. Width Number

The width of the component. It is set in pixels.

W11.2.11. Width unit Enum

The following options are available:

- px – Width is given in pixels.
- % – Width is given as a percentage in relation to the parent width.
- content – Width is automatically set to fit the entire content in width.

W11.2.12. Height *Number*

The height of the component. It is set in pixels.

W11.2.13. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W11.2.14. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W11.2.15. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W11.2.16. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W11.2.17. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W11.2.18. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W11.2.19. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W11.2.20. Click focusable *Boolean*

Add focused state to the object when clicked.

W11.2.21. Checkable *Boolean*

Toggle checked state when the object is clicked.

W11.2.22. Scrollable *Boolean*

Make the object scrollable.

W11.2.23. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W11.2.24. Scroll momentum Boolean

Make the object scroll further when "thrown".

W11.2.25. Scroll one Boolean

Allow scrolling only one snappable children.

W11.2.26. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W11.2.27. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W11.2.28. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W11.2.29. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W11.2.30. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W11.2.31. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W11.2.32. Event bubble Boolean

Propagate the events to the parent too.

W11.2.33. Gesture bubble Boolean

Propagate the gestures to the parent.

W11.2.34. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W11.2.35. Ignore layout Boolean

Make the object positionable by the layouts.

W11.2.36. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W11.2.37. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W11.2.38. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W11.2.39. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W11.2.40. Checked** *EXPRESSION (boolean)*

Toggled or checked state.

W11.2.41. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W11.2.42. Disabled *EXPRESSION (boolean)*

Disabled state

W11.2.43. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W11.2.44. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W11.2.45. Pressed *Boolean*

Being pressed.

Events**W11.2.46. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W11.2.47. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W11.2.48. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W11.2.49. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W12. Calendar



W12.1. Description

This Widget displays a calendar.
More info ([link](#))

W12.2. Properties

Specific

W12.2.1. Year *Number*

Initially selected year.

W12.2.2. Month *Number*

Initially selected month.

W12.2.3. Day *Number*

Initially selected day.

General

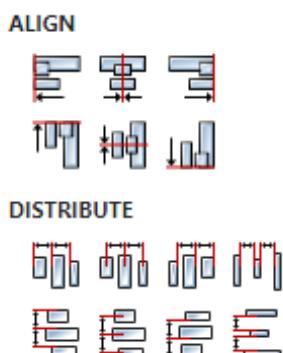
W12.2.4. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W12.2.5. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W12.2.6. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W12.2.7. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W12.2.8. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W12.2.9. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W12.2.10. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W12.2.11. Width *Number*

The width of the component. It is set in pixels.

W12.2.12. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W12.2.13. Height *Number*

The height of the component. It is set in pixels.

W12.2.14. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W12.2.15. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W12.2.16. Use style ObjectReference

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W12.2.17. Hidden EXPRESSION (boolean)**

Make the object hidden.

W12.2.18. Hidden flag type Enum

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W12.2.19. Clickable EXPRESSION (boolean)

Make the object clickable by input devices.

W12.2.20. Clickable flag type Enum

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W12.2.21. Click focusable Boolean

Add focused state to the object when clicked.

W12.2.22. Checkable Boolean

Toggle checked state when the object is clicked.

W12.2.23. Scrollable Boolean

Make the object scrollable.

W12.2.24. Scroll elastic Boolean

Allow scrolling inside but with slower speed.

W12.2.25. Scroll momentum Boolean

Make the object scroll further when "thrown".

W12.2.26. Scroll one Boolean

Allow scrolling only one snappable children.

W12.2.27. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W12.2.28. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W12.2.29. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W12.2.30. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W12.2.31. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W12.2.32. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W12.2.33. Event bubble Boolean

Propagate the events to the parent too.

W12.2.34. Gesture bubble Boolean

Propagate the gestures to the parent.

W12.2.35. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W12.2.36. Ignore layout Boolean

Make the object positionable by the layouts.

W12.2.37. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W12.2.38. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W12.2.39. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W12.2.40. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W12.2.41. Checked EXPRESSION (boolean)**

Toggled or checked state.

W12.2.42. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W12.2.43. Disabled *EXPRESSION (boolean)*

Disabled state

W12.2.44. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W12.2.45. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W12.2.46. Pressed *Boolean*

Being pressed.

Events**W12.2.47. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W12.2.48. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W12.2.49. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W12.2.50. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W12.3. Examples

- *LVGL Widgets Demo*

W13. Canvas



W13.1. Description

A Canvas inherits from Image where the user can draw anything. Rectangles, texts, images, lines, arcs can be drawn here using lvgl's drawing engine. All this must be implemented in your custom code. More info ([link](#))

W13.2. Properties

General

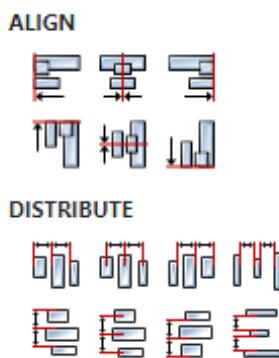
W13.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W13.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W13.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W13.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W13.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W13.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W13.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W13.2.8. Width *Number*

The width of the component. It is set in pixels.

W13.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W13.2.10. Height *Number*

The height of the component. It is set in pixels.

W13.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W13.2.12. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W13.2.13. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W13.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W13.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W13.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W13.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W13.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W13.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W13.2.20. Scrollable *Boolean*

Make the object scrollable.

W13.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W13.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W13.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W13.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W13.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W13.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W13.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W13.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W13.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W13.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W13.2.31. Gesture bubble *Boolean*

Propagate the gestures to the parent.

W13.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W13.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W13.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W13.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W13.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W13.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States

W13.2.38. Checked EXPRESSION (boolean)

Toggled or checked state.

W13.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W13.2.40. Disabled EXPRESSION (boolean)

Disabled state

W13.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W13.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W13.2.43. Pressed Boolean

Being pressed.

Events

W13.2.44. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W13.2.45. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W13.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W13.2.47. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W14. Chart



W14.1. Description

Charts are a basic object to visualize data points.
More info ([link](#))

W14.2. Properties

General

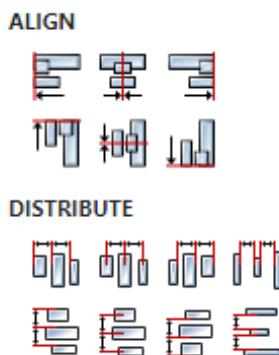
W14.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W14.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W14.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W14.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W14.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.

- `%` – Left is set as a percentage in relation to the parent width.

W14.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W14.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W14.2.8. Width *Number*

The width of the component. It is set in pixels.

W14.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W14.2.10. Height *Number*

The height of the component. It is set in pixels.

W14.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W14.2.12. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W14.2.13. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W14.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W14.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W14.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W14.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W14.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W14.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W14.2.20. Scrollable *Boolean*

Make the object scrollable.

W14.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W14.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W14.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W14.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W14.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W14.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W14.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W14.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W14.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W14.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W14.2.31. Gesture bubble *Boolean*

Propagate the gestures to the parent.

W14.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W14.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W14.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W14.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W14.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W14.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W14.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W14.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W14.2.40. Disabled EXPRESSION (boolean)

Disabled state

W14.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W14.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W14.2.43. Pressed Boolean

Being pressed.

Events

W14.2.44. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W14.2.45. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W14.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

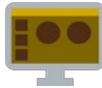
W14.2.47. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W14.3. Examples

- Dashboard Widgets Demo

W15. Checkbox (Dashboard)



W15.1. Description

Checkbox Widget is used when we want a turn ON or turn OFF option.

W15.2. Properties

Specific

W15.2.1. Value *EXPRESSION (any)*

Boolean variable in which `true` is stored when the switch is ON and `false` when the switch is OFF.

W15.2.2. Label *EXPRESSION (string)*

Label displayed next to the checkbox.

W15.2.3. Enabled *EXPRESSION (any)*

If it is true, then the checkbox is enabled, otherwise it will be disabled.

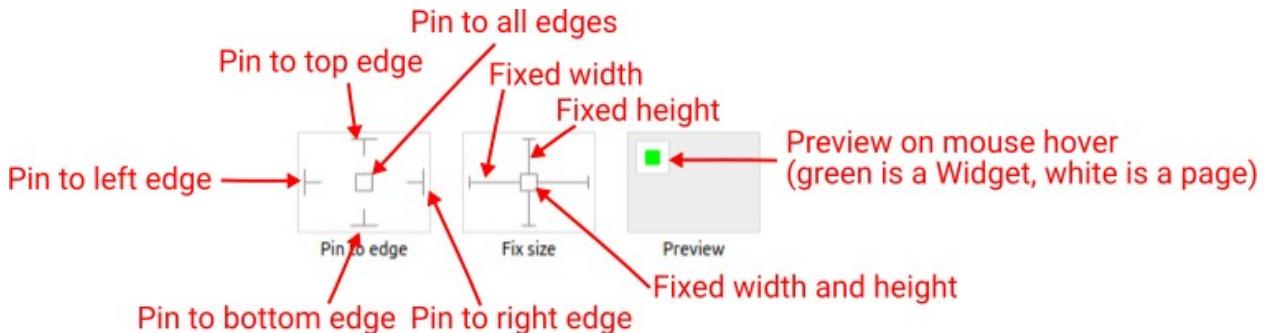
W15.2.4. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W15.2.5. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



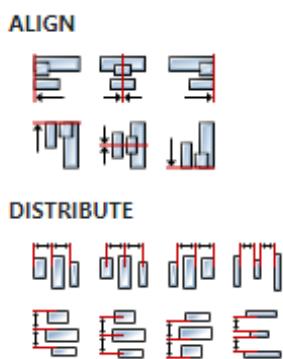
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W15.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W15.2.7. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W15.2.8. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), sim-

ple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W15.2.9. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W15.2.10. Width Number

The width of the component. It is set in pixels.

W15.2.11. Height Number

The height of the component. It is set in pixels.

W15.2.12. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W15.2.13. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W15.2.14. Default Object

Style used when rendering of the Widget.

Events

W15.2.15. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W15.2.16. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory ac-`

tion component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W15.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W15.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W15.2.19. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W15.3. Examples

- *Dashboard Widgets Demo*

W16. Checkbox (LVGL)



W16.1. Description

Checkbox Widget is used when we want a turn ON or turn OFF option.
More info ([link](#))

W16.2. Properties

Specific

W16.2.1. Text *String*

Label displayed next to the checkbox.

General

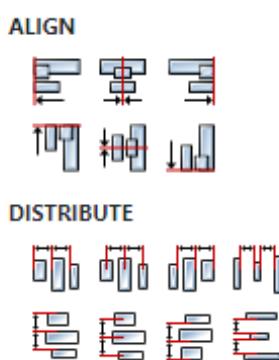
W16.2.2. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W16.2.3. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W16.2.4. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W16.2.5. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*`

and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W16.2.6. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W16.2.7. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W16.2.8. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W16.2.9. Width *Number*

The width of the component. It is set in pixels.

W16.2.10. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W16.2.11. Height *Number*

The height of the component. It is set in pixels.

W16.2.12. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W16.2.13. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W16.2.14. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W16.2.15. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W16.2.16. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W16.2.17. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W16.2.18. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W16.2.19. Click focusable *Boolean*

Add focused state to the object when clicked.

W16.2.20. Checkable *Boolean*

Toggle checked state when the object is clicked.

W16.2.21. Scrollable *Boolean*

Make the object scrollable.

W16.2.22. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W16.2.23. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W16.2.24. Scroll one *Boolean*

Allow scrolling only one snappable children.

W16.2.25. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W16.2.26. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W16.2.27. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W16.2.28. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W16.2.29. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W16.2.30. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W16.2.31. Event bubble Boolean

Propagate the events to the parent too.

W16.2.32. Gesture bubble Boolean

Propagate the gestures to the parent.

W16.2.33. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W16.2.34. Ignore layout Boolean

Make the object positionable by the layouts.

W16.2.35. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W16.2.36. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W16.2.37. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W16.2.38. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W16.2.39. Checked EXPRESSION (boolean)**

Toggled or checked state.

W16.2.40. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W16.2.41. Disabled EXPRESSION (boolean)

Disabled state

W16.2.42. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W16.2.43. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W16.2.44. Pressed Boolean

Being pressed.

Events**W16.2.45. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W16.2.46. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W16.2.47. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W16.2.48. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W16.3. Examples

- *Dashboard Widgets Demo*

W17. Colorwheel



W17.1. Description

This widget allows the user to select a color.
More info ([link](#))

W17.2. Properties

Specific

W17.2.1. Mode *Enum*

Select which part of the color will be changed with the Widget:

- HUE
- SATURATION
- VALUE

W17.2.2. Fixed mode *Boolean*

The color mode can be fixed (so as to not change with long press) using this item.

General

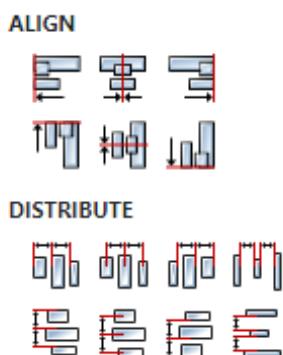
W17.2.3. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W17.2.4. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W17.2.5. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W17.2.6. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2`.

W17.2.7. Left unit Enum

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W17.2.8. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W17.2.9. Top unit Enum

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W17.2.10. Width Number

The width of the component. It is set in pixels.

W17.2.11. Width unit Enum

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W17.2.12. Height Number

The height of the component. It is set in pixels.

W17.2.13. Height unit Enum

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W17.2.14. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set

the title of the tab that contains this widget.

Style

W17.2.15. Use style ObjectReference

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W17.2.16. Hidden EXPRESSION (boolean)

Make the object hidden.

W17.2.17. Hidden flag type Enum

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W17.2.18. Clickable EXPRESSION (boolean)

Make the object clickable by input devices.

W17.2.19. Clickable flag type Enum

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W17.2.20. Click focusable Boolean

Add focused state to the object when clicked.

W17.2.21. Checkable Boolean

Toggle checked state when the object is clicked.

W17.2.22. Scrollable Boolean

Make the object scrollable.

W17.2.23. Scroll elastic Boolean

Allow scrolling inside but with slower speed.

W17.2.24. Scroll momentum Boolean

Make the object scroll further when "thrown".

W17.2.25. Scroll one Boolean

Allow scrolling only one snappable children.

W17.2.26. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W17.2.27. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W17.2.28. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W17.2.29. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W17.2.30. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W17.2.31. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W17.2.32. Event bubble Boolean

Propagate the events to the parent too.

W17.2.33. Gesture bubble Boolean

Propagate the gestures to the parent.

W17.2.34. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W17.2.35. Ignore layout Boolean

Make the object positionable by the layouts.

W17.2.36. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W17.2.37. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W17.2.38. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W17.2.39. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W17.2.40. Checked EXPRESSION (boolean)**

Toggled or checked state.

W17.2.41. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W17.2.42. Disabled *EXPRESSION (boolean)*

Disabled state

W17.2.43. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W17.2.44. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W17.2.45. Pressed *Boolean*

Being pressed.

Events**W17.2.46. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W17.2.47. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W17.2.48. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

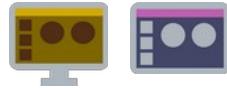
W17.2.49. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W17.3. Examples

- *Dashboard Widgets Demo*

W18. Container



W18.1. Description

It is used to group several Widgets, and it is used when we want to additionally organize a page that contains a large number of Widgets or if we want to perform some operation on several Widgets at once, e.g. hide using the `Visible` property of the Container. When the Widget is inside the Container, then its left and top coordinates are relative to the left and top of the Container, which means that when the Container is moved, all the Widgets inside it are also moved. Widgets are added to the Container via the *Widgets Structure* panel using drag and drop.

W18.2. Properties

Specific

W18.2.1. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Layout

W18.2.2. Layout *Enum*

Determines how Child widgets are positioned within this container:

- `Static` – Child widgets are positioned within the Container using their left and top properties.
- `Horizontal` – Child widgets are positioned from left to right (or vice versa if RTL is selected in the `SetPageDirection` action) and in order according to the order set through the *Widgets Structure* panel. So, if this option is selected, then the left property of the Child widget is not used. If a Child widget is hidden, then it is skipped and its position is taken by the next visible Widget in the list.
- `Vertical` – Child widgets are positioned from top to bottom and in order according to the order set through the *Widgets Structure* panel. So, if this option is selected, then the top property of the Child widget is not used. If a Child widget is hidden, then it is skipped and its position is taken by the next visible Widget from the list.
- `Docking Manager` – Each child widget is located inside separate tab and these tabs can be arranged within container boundaries in an arbitrary way. For example, they can be grouped inside tab strips or docked at any position inside container. This options is only available for Dashboard projects.

W18.2.3. Edit layout *Any*

If `Layout` property is set to `Docking Manager` then this button opens the editor for configuring initial position of tabs within container. Please note, user can later change the layout configuration when dashboard is running. User changes are saved in `.eez-project-runtime-settings` file, created at the same location where the `.eez-project` file is.

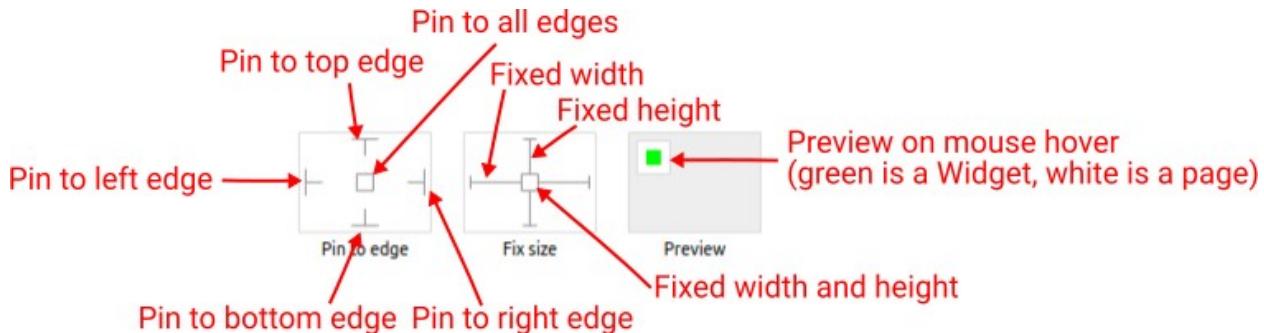
W18.2.4. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Position and size

W18.2.5. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

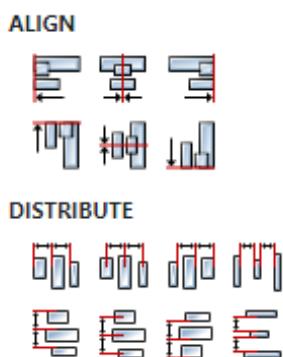
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W18.2.6. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W18.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W18.2.8. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W18.2.9. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W18.2.10. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W18.2.11. Width *Number*

The width of the component. It is set in pixels.

W18.2.12. Height *Number*

The height of the component. It is set in pixels.

W18.2.13. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

General**W18.2.14. Name** *String*

Optional name to display in the *Widgets Structure* panel in the editor. If not set then `Container` is displayed.

Style**W18.2.15. Default** *Object*

Style used when rendering the background of the Widget.

Events**W18.2.16. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W18.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W18.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W18.2.19. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W18.3. Examples

- *eez-gui-widgets-demo*

W19. Container (Dashboard)



W19.1. Description

It is used to group several Widgets, and it is used when we want to additionally organize a page that contains a large number of Widgets or if we want to perform some operation on several Widgets at once, e.g. hide using the `Visible` property of the Container. When the Widget is inside the Container, then its left and top coordinates are relative to the left and top of the Container, which means that when the Container is moved, all the Widgets inside it are also moved. Widgets are added to the Container via the *Widgets Structure* panel using drag and drop.

W19.2. Properties

Specific

W19.2.1. Visible EXPRESSION (boolean)

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Layout

W19.2.2. Layout Enum

Determines how Child widgets are positioned within this container:

- `Static` – Child widgets are positioned within the Container using their left and top properties.
- `Horizontal` – Child widgets are positioned from left to right (or vice versa if RTL is selected in the `SetPageDirection` action) and in order according to the order set through the *Widgets Structure* panel. So, if this option is selected, then the left property of the Child widget is not used. If a Child widget is hidden, then it is skipped and its position is taken by the next visible Widget in the list.
- `Vertical` – Child widgets are positioned from top to bottom and in order according to the order set through the *Widgets Structure* panel. So, if this option is selected, then the top property of the Child widget is not used. If a Child widget is hidden, then it is skipped and its position is taken by the next visible Widget from the list.
- `Docking Manager` – Each child widget is located inside separate tab and these tabs can be arranged within container boundaries in an arbitrary way. For example, they can be grouped inside tab strips or docked at any position inside container. This options is only available for Dashboard projects.

W19.2.3. Edit layout Any

If `Layout` property is set to `Docking Manager` then this button opens the editor for configuring initial position of tabs within container. Please note, user can later change the layout configuration when dashboard is running. User changes are saved in `.eez-project-runtime-settings` file, created at the same location where the `.eez-project` file is.

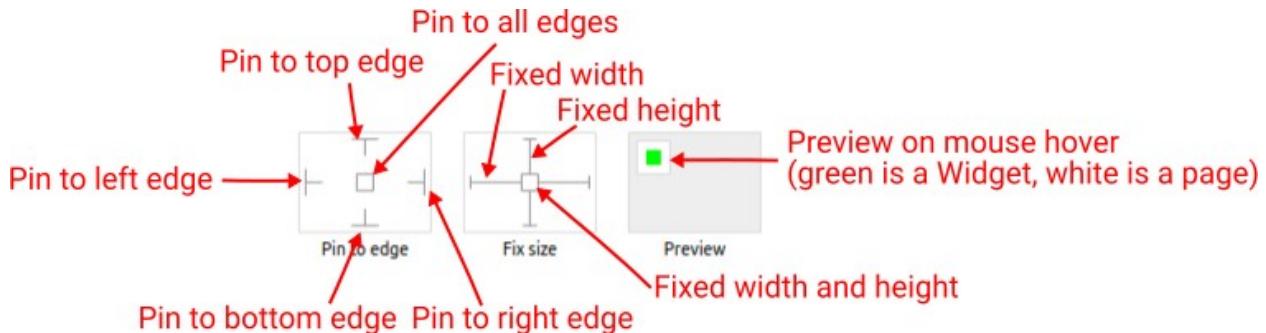
W19.2.4. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Position and size

W19.2.5. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

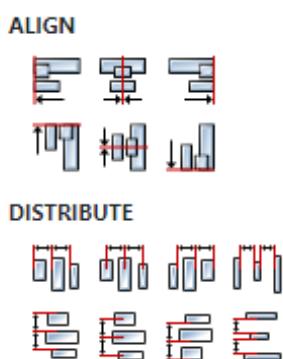
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W19.2.6. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W19.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W19.2.8. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W19.2.9. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W19.2.10. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W19.2.11. Width *Number*

The width of the component. It is set in pixels.

W19.2.12. Height *Number*

The height of the component. It is set in pixels.

W19.2.13. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

General**W19.2.14. Name** *String*

Optional name to display in the *Widgets Structure* panel in the editor. If not set then `Container` is displayed.

Style**W19.2.15. Default** *Object*

Style used when rendering the background of the Widget.

Events**W19.2.16. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W19.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W19.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W19.2.19. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W19.3. Examples

- `eez-gui-widgets-demo`

W20. Container (LVGL)



W20.1. Description

Use this widget as a container for other widgets.

W20.2. Properties

General

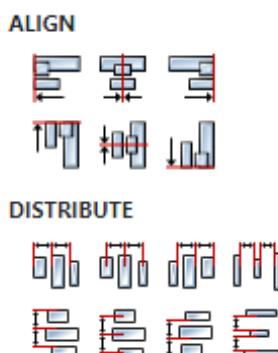
W20.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W20.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W20.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W20.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W20.2.5. Left unit *Enum*

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W20.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W20.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W20.2.8. Width *Number*

The width of the component. It is set in pixels.

W20.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W20.2.10. Height *Number*

The height of the component. It is set in pixels.

W20.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W20.2.12. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W20.2.13. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W20.2.14. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W20.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W20.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W20.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W20.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W20.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W20.2.20. Scrollable *Boolean*

Make the object scrollable.

W20.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W20.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W20.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W20.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W20.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W20.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W20.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W20.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W20.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W20.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W20.2.31. Gesture bubble *Boolean*

Propagate the gestures to the parent.

W20.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W20.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W20.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W20.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W20.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W20.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W20.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W20.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W20.2.40. Disabled EXPRESSION (boolean)

Disabled state

W20.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W20.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W20.2.43. Pressed Boolean

Being pressed.

Events

W20.2.44. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W20.2.45. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W20.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W20.2.47. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W21. DisplayData



W21.1. Description

Similar to the `Text` Widget, but it has some more options that are set via the `Display option` and `Refresh rate` properties.

W21.2. Properties

Specific

W21.2.1. Data `EXPRESSION (any)`

An expression that, when calculated, is converted into a string and displayed inside the widget.

W21.2.2. Display option `Enum`

If the calculated `Data` is a floating point number, then with this property we can choose which part of the floating point number is displayed:

- `All` – displays the entire floating point number
- `Integer` – displays only the whole part (integer) of the number
- `Fraction` – displays only decimals (fractions) of a number

W21.2.3. Refresh rate `EXPRESSION (any)`

This property defines how often the content of this widget will be refreshed. It is set in milliseconds. If the `Data` changes with a high frequency and if the content of this widget is renewed with that frequency (e.g. if the Refresh rate is set to 0) then it will be a problem to see that content, therefore it is recommended to increase the Refresh rate, e.g. at 200 ms.

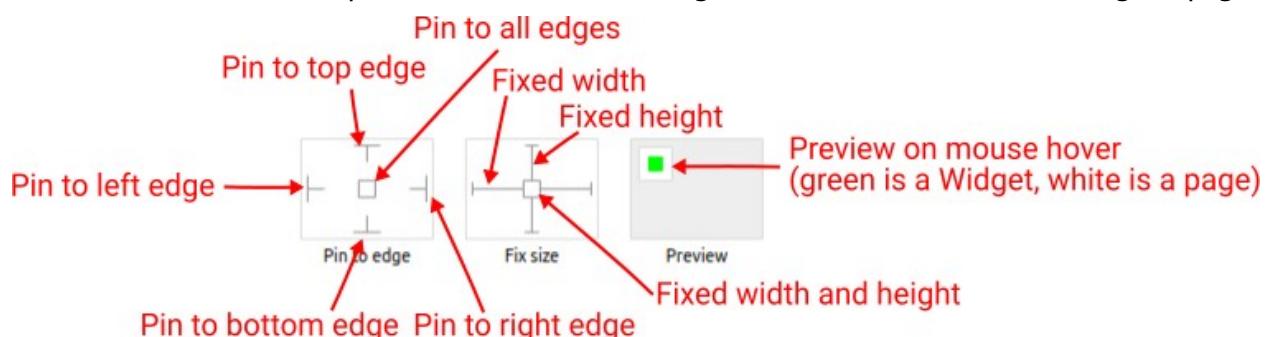
W21.2.4. Visible `EXPRESSION (boolean)`

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W21.2.5. Resizing `Any`

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the `Pin to edge` option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the `Scale to fit` option is selected. E.g. if we selected `Pin to top edge` then the distance between the top edge of the page and the top edge

of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

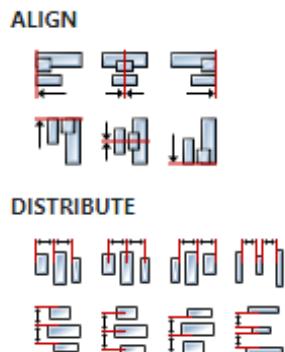
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W21.2.6. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W21.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W21.2.8. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W21.2.9. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the *Top*, *Width* and *Height* properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W21.2.10. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W21.2.11. Width Number

The width of the component. It is set in pixels.

W21.2.12. Height Number

The height of the component. It is set in pixels.

W21.2.13. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W21.2.14. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W21.2.15. Default** *Object*

Style used when rendering of the Widget.

W21.2.16. Focused *Object*

Style to be used for rendering if the Widget is in focus.

Events**W21.2.17. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W21.2.18. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W21.2.19. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

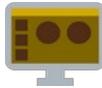
W21.2.20. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W21.3. Examples

- *eez-gui-widgets-demo*

W22. Dropdown (Dashboard)



W22.1. Description

We can use this widget when we need to select one option from the list.

W22.2. Properties

Specific

W22.2.1. Data *EXPRESSION (integer)*

The variable in which the zero-based index of the selected option will be stored.

W22.2.2. Options *EXPRESSION (any)*

List of options.

W22.2.3. Enabled *EXPRESSION (any)*

If it is true, then the dropdown is enabled, otherwise it will be disabled.

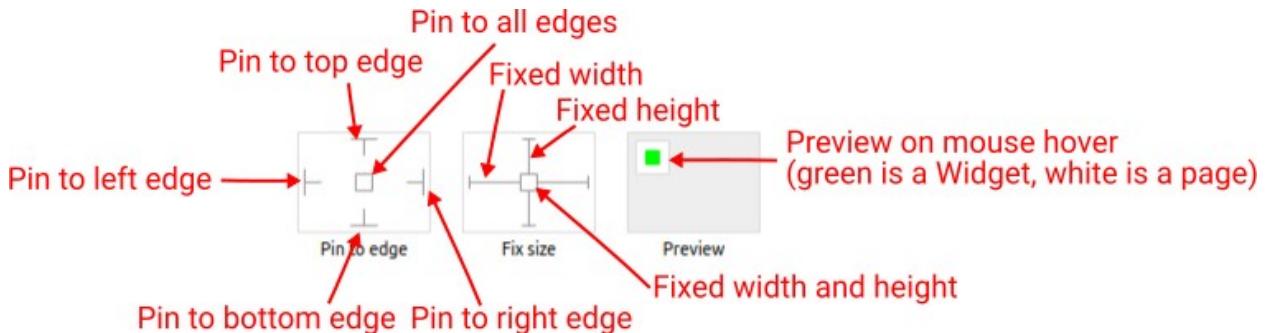
W22.2.4. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W22.2.5. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



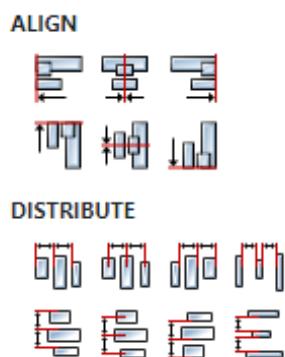
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W22.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W22.2.7. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W22.2.8. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), sim-

ple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W22.2.9. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W22.2.10. Width Number

The width of the component. It is set in pixels.

W22.2.11. Height Number

The height of the component. It is set in pixels.

W22.2.12. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W22.2.13. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W22.2.14. Default Object

Style used when rendering of the Widget.

Events

W22.2.15. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W22.2.16. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory ac-`

tion component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W22.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W22.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W22.2.19. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W22.3. Examples

- *Dashboard Widgets Demo*

W23. Dropdown (EEZ-GUI)



W23.1. Description

We can use this widget when we need to select one option from the list.

W23.2. Properties

Specific

W23.2.1. Data *EXPRESSION (integer)*

The variable in which the zero-based index of the selected option will be stored.

W23.2.2. Options *EXPRESSION (any)*

List of options.

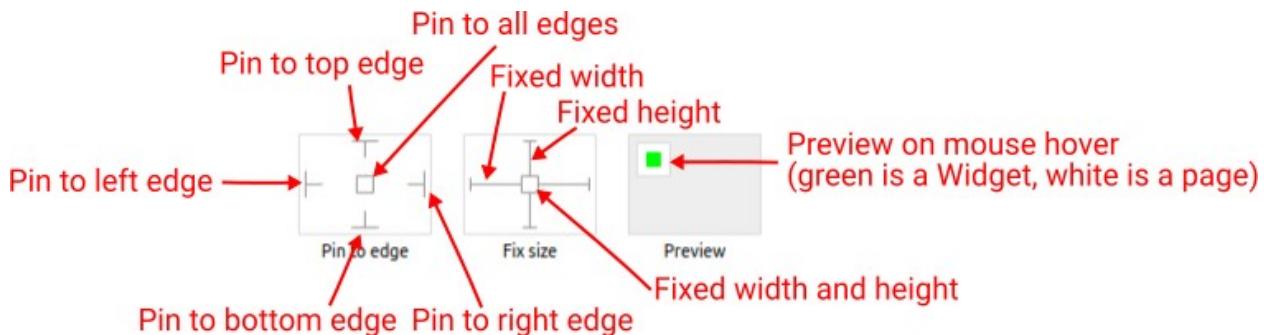
W23.2.3. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W23.2.4. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

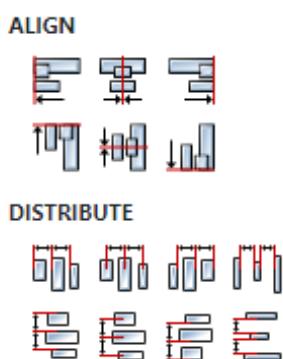
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W23.2.5. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W23.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W23.2.7. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W23.2.8. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W23.2.9. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W23.2.10. Width *Number*

The width of the component. It is set in pixels.

W23.2.11. Height *Number*

The height of the component. It is set in pixels.

W23.2.12. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W23.2.13. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W23.2.14. Default** *Object*

Style used when rendering of the Widget.

Events**W23.2.15. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W23.2.16. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W23.2.17. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W23.2.18. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W23.3. Examples

- *eez-gui-widgets-demo*

W24. Dropdown (LVGL)



W24.1. Description

The drop-down list allows the user to select one value from a list.
More info ([link](#))

W24.2. Properties

Specific

W24.2.1. Options *EXPRESSION (array:string)*

List of options.

W24.2.2. Options type *Enum*

Select between Literal and Expression. If Literal is selected then Options are entered one option per line. If Expression is selected then options are evaluated from Options expression which must be of type array:string.

W24.2.3. Selected *EXPRESSION (integer)*

The zero-based index of the selected option.

W24.2.4. Selected type *Enum*

Select between Literal and Assignable. If Assignable is selected then Options can be variable in which the zero-based index of the selected option will be stored.

General

W24.2.5. Name *String*

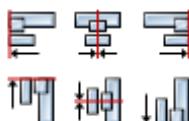
Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

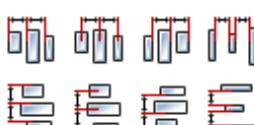
W24.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



W24.2.7. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W24.2.8. Left** *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W24.2.9. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W24.2.10. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W24.2.11. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W24.2.12. Width *Number*

The width of the component. It is set in pixels.

W24.2.13. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W24.2.14. Height *Number*

The height of the component. It is set in pixels.

W24.2.15. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

W24.2.16. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W24.2.17. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W24.2.18. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W24.2.19. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W24.2.20. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W24.2.21. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W24.2.22. Click focusable *Boolean*

Add focused state to the object when clicked.

W24.2.23. Checkable *Boolean*

Toggle checked state when the object is clicked.

W24.2.24. Scrollable *Boolean*

Make the object scrollable.

W24.2.25. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W24.2.26. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W24.2.27. Scroll one *Boolean*

Allow scrolling only one snappable children.

W24.2.28. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W24.2.29. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W24.2.30. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W24.2.31. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W24.2.32. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W24.2.33. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W24.2.34. Event bubble *Boolean*

Propagate the events to the parent too.

W24.2.35. Gesture bubble *Boolean*

Propagate the gestures to the parent.

W24.2.36. Adv hitest *Boolean*

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W24.2.37. Ignore layout *Boolean*

Make the object positionable by the layouts.

W24.2.38. Floating *Boolean*

Do not scroll the object when the parent scrolls and ignore layout.

W24.2.39. Overflow visible *Boolean*

Do not clip the children's content to the parent's boundary.

W24.2.40. Scrollbar mode *Enum*

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W24.2.41. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States

W24.2.42. Checked *EXPRESSION (boolean)*

Toggled or checked state.

W24.2.43. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W24.2.44. Disabled *EXPRESSION (boolean)*

Disabled state

W24.2.45. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W24.2.46. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W24.2.47. Pressed *Boolean*

Being pressed.

Events**W24.2.48. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W24.2.49. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W24.2.50. Outputs *Array*

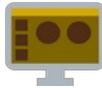
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W24.2.51. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W24.3. Examples

- *Dashboard Widgets Demo*



W25.1. Description

Displays a line chart using the same Widget as in the Instrument *History* panel.

W25.2. Properties

Specific

W25.2.1. Chart mode *Enum*

The following modes are available:

- `Single chart` – Displays a single chart.
- `Multiple charts` – Displays multiple charts.
- `EEZ DLOG` – Displays the chart given by the EEZ DLOG file format.
- `Instrument History Item` – Displays a chart from the instrument history.

W25.2.2. Chart data *EXPRESSION (any)*

If `Chart mode` is set to `Single chart`, then a string, array or blob containing the samples that will be displayed in the chart should be set here. If `Chart mode` is set to `EEZ DLOG` then the content of the EEZ DLOG file should be set here (e.g. it can be read with `FileRead` Action, see *EEZ Chart example*).

This property is not used when the `Chart mode` is `Multiple charts` or `Instrument History item`.

W25.2.3. Format *EXPRESSION (string)*

Format of `Data` property. Possible values:

- `"float"` – "Chart data" must be a blob containing 32-bit, little-endian float numbers, or `array:float`
- `"double"` – "Chart data" must be a blob containing 64-bit, little-endian float numbers, or `array:float`
- `"rigol-byte"` – "Chart data" must be a blob containing 8-bit unsigned integer numbers
- `"rigol-word"` – "Chart data" must be a blob containing 16-bit unsigned integer numbers
- `"csv"` – "Chart data" must be a CSV string, the first column is taken

This property is only used when the `Chart mode` is `Single chart`.

W25.2.4. Sampling rate *EXPRESSION (integer)*

Sampling rate or number of samples per second (SPS).

This property is only used when the `Chart mode` is `Single chart`.

W25.2.5. Unit name *EXPRESSION (integer)*

The unit displayed on the Y-axis. The X-axis is always time.

This property is only used when the `Chart mode` is `Single chart`.

W25.2.6. Color *EXPRESSION (string)*

The color of the line in the chart.

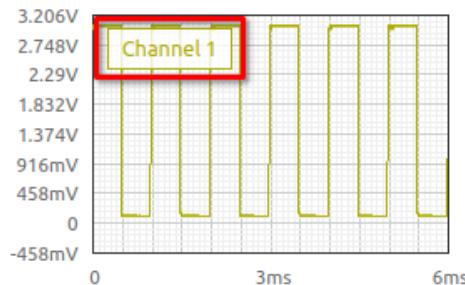
This property is only used when the `Chart mode` is `Single chart`.

W25.2.7. Label *EXPRESSION (string)*

Chart label:



07/24/2023 9:26:29 PM
 application/eez-raw, 2.86 MB
 Channel: 1, Sampling rate: 500 MS/s
 Preamble: 0, 2, 3000000, 1, 2.000000e-09, -3.000000e-03, 0, 4.296875e-02, 1, 118



This property is only used when the `Chart mode` is `Single chart`.

W25.2.8. Offset *EXPRESSION (string)*

Offset value used in formula `offset + sample_value * scale` which transforms sample value to sample position on y axis in the chart.

This property is only used when the `Chart mode` is `Single chart`.

W25.2.9. Scale *EXPRESSION (string)*

When displaying samples, the formula `offset + sample_value * scale` is used.

This property is only used when the `Chart mode` is `Single chart`.

W25.2.10. Charts *Array*

List of chart definitions when `Chart mode` is set to `Multiple charts`. Each definition contains these properties:

- `Chart data`
- `Format`
- `Sampling rate`
- `Unit`
- `Color`
- `Label`
- `Offset`
- `Scale`

These properties have the same meaning as the corresponding property when `Single chart` mode is selected.

W25.2.11. History item ID *EXPRESSION (string)*

This ID is obtained using `AddToInstrumentHistory` action through `id` output of that action.

This property is only used when the `Chart mode` is `Instrument History Item`.

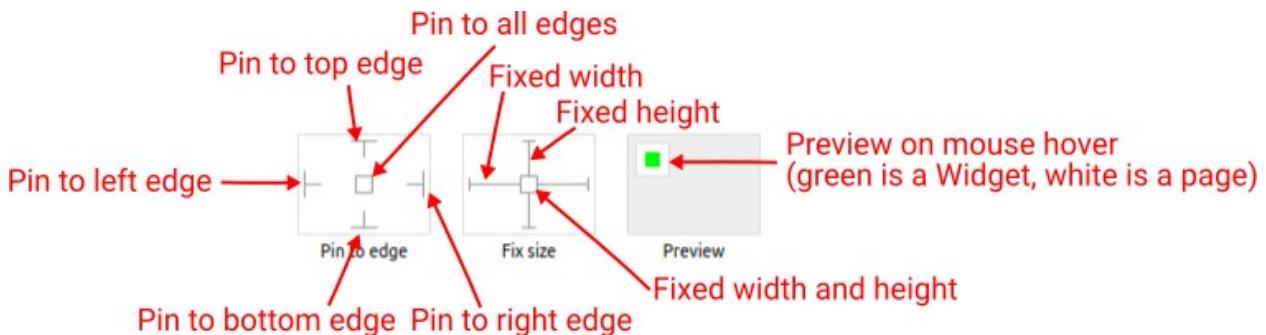
W25.2.12. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W25.2.13. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



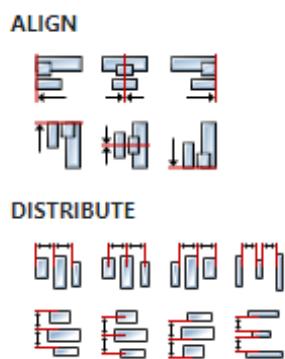
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W25.2.14. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W25.2.15. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W25.2.16. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), sim-

ple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W25.2.17. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W25.2.18. Width Number

The width of the component. It is set in pixels.

W25.2.19. Height Number

The height of the component. It is set in pixels.

W25.2.20. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W25.2.21. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W25.2.22. Default Object

Style used when rendering of the Widget.

Events

W25.2.23. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W25.2.24. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory ac-`

tion component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W25.2.25. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W25.2.26. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W25.2.27. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W25.3. Examples

- *Line Chart*
- *EEZ Chart*
- *Rigol Waveform Data*

W26. Embedded Dashboard



W26.1. Description

Use this widget to embed external dashboard project.

W26.2. Properties

Specific

W26.2.1. Dashboard *EXPRESSION (string)*

Location of the external dashboard project. It can be absolute file path, relative file path or HTTP(S) URL.

W26.2.2. Open dashboard *Any*

Use this button as convenience shortcut to open external dashboard project inside Project Editor. Not available if external dashboard location is given by HTTP(S) URL.

W26.2.3. Dashboard parameters *Array*

List of parameters where each parameter is specified by the name and value. Each parameter name must correspond to the global variable name in the external dashboard. When host dashboard runs, parameter values will be continuously evaluated at the host dashboard side and associated global variables inside external dashboard will be changed.

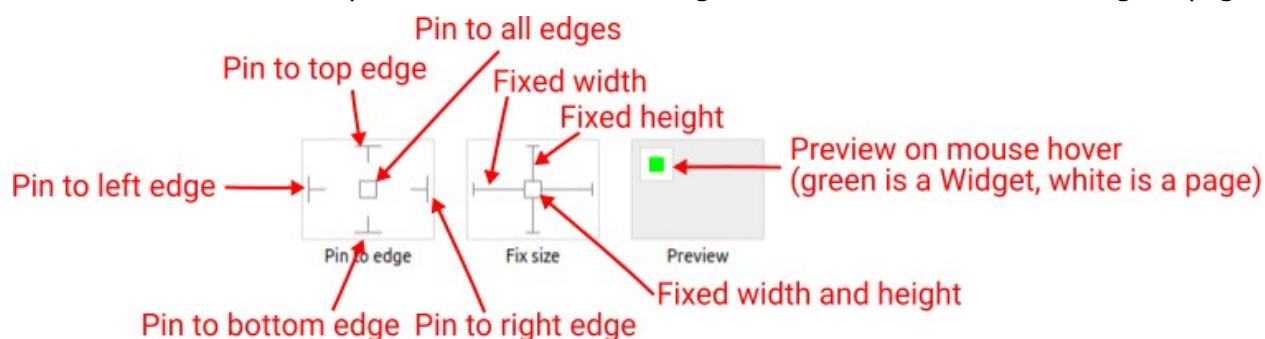
W26.2.4. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W26.2.5. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



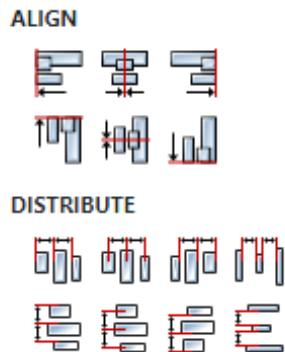
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W26.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W26.2.7. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W26.2.8. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W26.2.9. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W26.2.10. Width Number

The width of the component. It is set in pixels.

W26.2.11. Height Number

The height of the component. It is set in pixels.

W26.2.12. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W26.2.13. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set

the title of the tab that contains this widget.

Events

W26.2.14. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W26.2.15. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W26.2.16. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W26.2.17. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

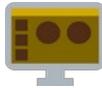
W26.2.18. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W26.3. Examples

- BB3 SCPI Terminal and Dashboard

W27. Gauge (Dashboard)



W27.1. Description

Displays the value selected through the `Data` property inside the gauge chart.

W27.2. Properties

Specific

W27.2.1. Data *EXPRESSION (any)*

The value within the `[Min, Max]` range that is displayed in the Widget.

W27.2.2. Title *String*

The name displayed above the gauge chart.

W27.2.3. Min range *EXPRESSION (float)*

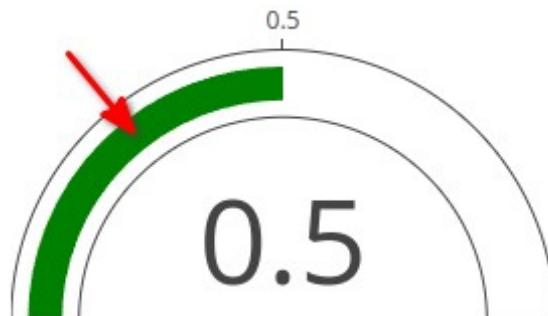
The minimum value that `Data` can contain.

W27.2.4. Max range *EXPRESSION (float)*

The maximum value that `Data` can contain.

W27.2.5. Color *Color*

The color of the arc bar inside the chart.



W27.2.6. Margin *Object*

Manually selected margin values between the Widget borders and the chart itself within the Widget.

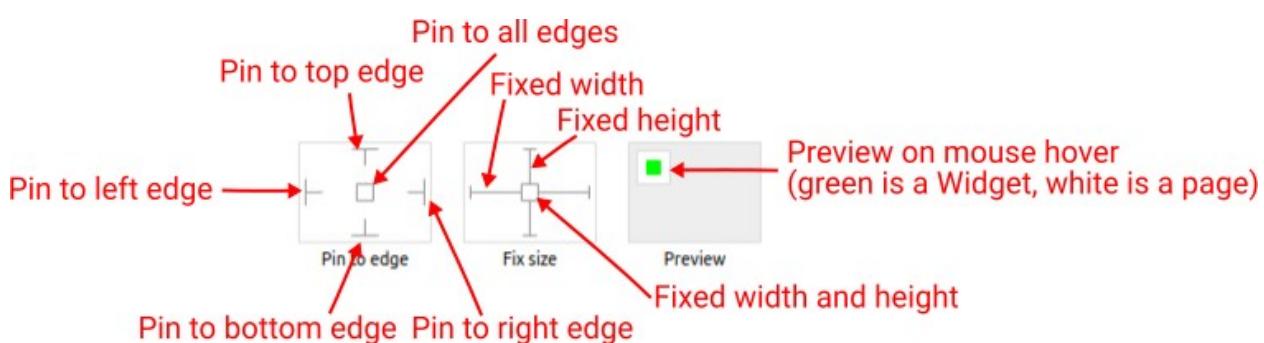
W27.2.7. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W27.2.8. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



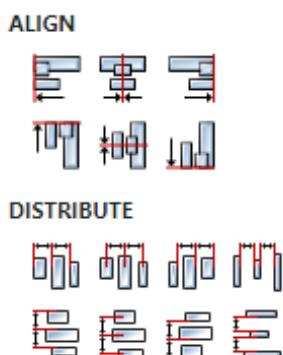
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W27.2.9. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W27.2.10. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W27.2.11. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the **Top**, **Width** and **Height** properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use **+**, **-**, ***** and **/** operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W27.2.12. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W27.2.13. Width *Number*

The width of the component. It is set in pixels.

W27.2.14. Height *Number*

The height of the component. It is set in pixels.

W27.2.15. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W27.2.16. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W27.2.17. Default** *Object*

Style used when rendering the background of the Widget.

Events**W27.2.18. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W27.2.19. Output widget handle** *Boolean*

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W27.2.20. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W27.2.21. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W27.2.22. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W27.3. Examples

- *Dashboard Widgets Demo*

W28. Gauge (EEZ-GUI)



W28.1. Description

Displays the value selected through the `Data` property inside the gauge chart. Also, if it is set, it will also show the threshold line at the given positions, for example to mark some critical value.

W28.2. Properties

Specific

W28.2.1. Data *EXPRESSION (any)*

The value within the `[Min, Max]` range that is displayed in the Widget.

W28.2.2. Min *EXPRESSION (any)*

The minimum value that `Data` can contain.

W28.2.3. Max *EXPRESSION (any)*

The maximum value that `Data` can contain.

W28.2.4. Threshold *EXPRESSION (any)*

An optional value within the range `[Min, Max]` at the position of which a line will be drawn in the default style (Threshold style).

W28.2.5. Unit *EXPRESSION (any)*

The unit that will be displayed to the right of the `Data` value.

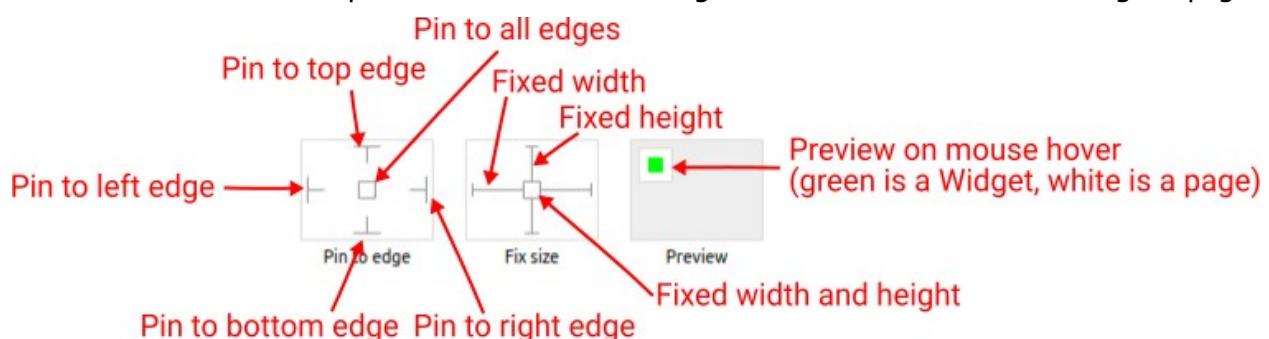
W28.2.6. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W28.2.7. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If

Pin to top edge is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

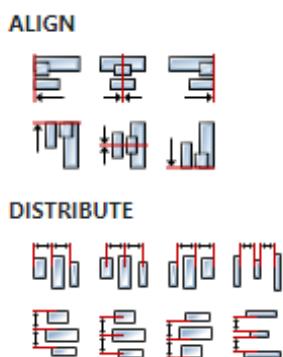
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W28.2.8. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W28.2.9. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W28.2.10. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W28.2.11. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W28.2.12. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W28.2.13. Width Number

The width of the component. It is set in pixels.

W28.2.14. Height Number

The height of the component. It is set in pixels.

W28.2.15. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W28.2.16. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W28.2.17. Default** *Object*

Style used when rendering the background of the Widget.

W28.2.18. Bar *Object*

Style used to render the bar inside the Widget.

W28.2.19. Value *Object*

Style used to render the value specified through `Data`.

W28.2.20. Ticks *Object*

The style used to render the ticks on the scale.

W28.2.21. Threshold *Object*

Style used to render the threshold line.

Events**W28.2.22. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W28.2.23. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W28.2.24. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W28.2.25. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W28.3. Examples

- `eez-gui-widgets-demo`

W29. Grid



W29.1. Description

Use this Widget when you want to display the same Widget multiple times inside the grid. This Widget has one Child widget under it, and the number of times it will be displayed depends on the `Data` property.

Multiplied Widgets depending on the `Grid flow` property can be displayed by filling the rows first:

Widget #0	Widget #1	Widget #2	Widget #3
Widget #4	Widget #5	Widget #6	Widget #7
Widget #8	Widget #9	Widget #10	Widget #11
Widget #12	Widget #13	Widget #14	Widget #15
Widget #16	Widget #17	Widget #18	Widget #19
Widget #20	Widget #21	Widget #22	Widget #23

... or columns:

Widget #0	Widget #6	Widget #12	Widget #18
Widget #1	Widget #7	Widget #13	Widget #19
Widget #2	Widget #8	Widget #14	Widget #20
Widget #3	Widget #9	Widget #15	Widget #21
Widget #4	Widget #10	Widget #16	Widget #22
Widget #5	Widget #11	Widget #17	Widget #23

It wouldn't be very useful if multiplied Widgets always had the same content, that's why there is a system variable `$index` that tells us in which order the Widget is rendered. That variable is zero based, that means when its value is 0 then the first Widget is rendered, when its value is 1 then the second Widget is rendered and so on. That `$index` can then be used within the expression of the property of the Child widget, and in this way it is achieved that each rendered Widget has different content (e.g. the contents shown above were created in such a way that we defined for the `Text` widget that the displayed text is calculated from the following expression: `"Widget # " + $index`).

W29.2. Properties

Specific

W29.2.1. Data *EXPRESSION (any)*

Determines how many times the Child widget will be multiplied, i.e. the number of elements in the grid. The value of this property can be an integer and then it is the number of elements, and if the value of this property is an array, then the number of elements in the list is equal to the number of elements in that array.

In the case of EEZ-GUI projects, the value of this property can also be `struct:$ScrollbarState`. The same structure is used for the `ScrollBar` Widget, which can then be connected to the `Grid` Widget via the `struct:$ScrollbarState` variable and thus enable scrolling of the list in case the total number of list elements is greater than the number of elements that fit within the `Grid` Widget.

More about the `struct:$ScrollbarState` system structure can be found in the `ScrollBar` Widget documentation.

W29.2.2. Grid flow *Enum*

Defines the grid filling method. We need to select `Row` if we want it to be filled row by row. We will select `Column` if we want it to be filled column by column.

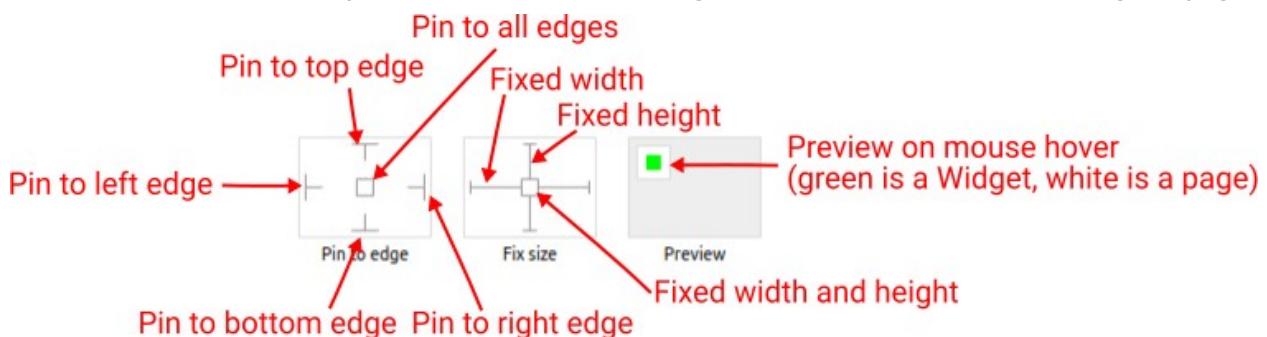
W29.2.3. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W29.2.4. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the `Pin to edge` option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the `Scale to fit` option is selected. E.g. if we selected `Pin to top edge` then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If

Pin to top edge is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

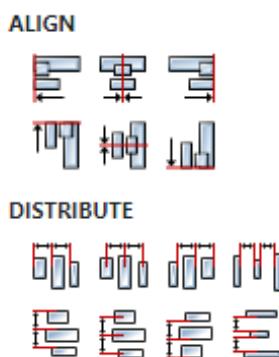
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W29.2.5. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W29.2.6. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W29.2.7. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W29.2.8. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W29.2.9. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W29.2.10. Width Number

The width of the component. It is set in pixels.

W29.2.11. Height Number

The height of the component. It is set in pixels.

W29.2.12. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W29.2.13. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W29.2.14. Default** *Object*

Style used when rendering the background of the Widget.

Events**W29.2.15. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W29.2.16. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W29.2.17. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W29.2.18. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W29.3. Examples

- *eez-gui-widgets-demo*
- *Tetris*

W30. Image



W30.1. Description

This Widget displays an image.
More info ([link](#))

W30.2. Properties

Specific

W30.2.1. Image *ObjectReference*

The name of the bitmap to be displayed.

W30.2.2. Pivot X *Number*

X position of the center of rotation. If left blank, the center of rotation is in the middle of the Widget.

W30.2.3. Pivot Y *Number*

Y position of the center of rotation. If left blank, the center of rotation is in the middle of the Widget.

W30.2.4. Scale *Number*

Scale factor. Set factor to `256` to disable zooming. A larger value enlarges the images (e.g. `512` double size), a smaller value shrinks it (e.g. `128` half size). Fractional scale works as well, e.g. `281` for 10% enlargement.

W30.2.5. Rotation *Number*

Rotation angle, angle has 0.1 degree precision, so for 45.8° set `458`. Image is rotated around the center of rotation which is defined with `Pivot X` and `Pivot Y` properties.

W30.2.6. Inner align *Enum*

By default the image widget's width and height will be sized automatically according to the image source. If the widget's width or height is set the larger value the `thisinner_align` property tells how to align the image source inside the widget.

General

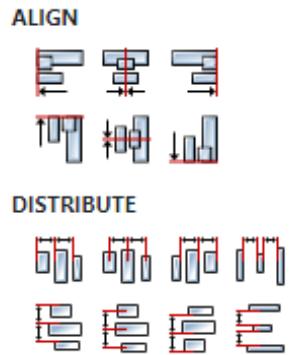
W30.2.7. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W30.2.8. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W30.2.9. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W30.2.10. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W30.2.11. Left unit Enum

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W30.2.12. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W30.2.13. Top unit Enum

The following options are available:

- px – Top is set in pixels.
- % – The top is set as a percentage in relation to the parent height.

W30.2.14. Width Number

The width of the component. It is set in pixels.

W30.2.15. Width unit Enum

The following options are available:

- px – Width is given in pixels.
- % – Width is given as a percentage in relation to the parent width.
- content – Width is automatically set to fit the entire content in width.

W30.2.16. Height *Number*

The height of the component. It is set in pixels.

W30.2.17. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W30.2.18. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W30.2.19. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W30.2.20. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W30.2.21. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W30.2.22. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W30.2.23. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W30.2.24. Click focusable *Boolean*

Add focused state to the object when clicked.

W30.2.25. Checkable *Boolean*

Toggle checked state when the object is clicked.

W30.2.26. Scrollable *Boolean*

Make the object scrollable.

W30.2.27. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W30.2.28. Scroll momentum Boolean

Make the object scroll further when "thrown".

W30.2.29. Scroll one Boolean

Allow scrolling only one snappable children.

W30.2.30. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W30.2.31. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W30.2.32. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W30.2.33. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W30.2.34. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W30.2.35. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W30.2.36. Event bubble Boolean

Propagate the events to the parent too.

W30.2.37. Gesture bubble Boolean

Propagate the gestures to the parent.

W30.2.38. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W30.2.39. Ignore layout Boolean

Make the object positionable by the layouts.

W30.2.40. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W30.2.41. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W30.2.42. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W30.2.43. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W30.2.44. Checked** *EXPRESSION (boolean)*

Toggled or checked state.

W30.2.45. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W30.2.46. Disabled *EXPRESSION (boolean)*

Disabled state

W30.2.47. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W30.2.48. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W30.2.49. Pressed *Boolean*

Being pressed.

Events**W30.2.50. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W30.2.51. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W30.2.52. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W30.2.53. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W30.3. Examples

- *LVGL Widgets Demo*

W31. Imgbutton



W31.1. Description

The `Imgbutton` is very similar to the simple `Button` Widget. The only difference is that it displays user-defined images in each state instead of drawing a rectangle.

More info ([link](#))

W31.2. Properties

Specific

W31.2.1. Released image *ObjectReference*

The image for the `Released` state.

W31.2.2. Pressed image *ObjectReference*

The image for the `Pressed` state.

W31.2.3. Disabled image *ObjectReference*

The image for the `Disabled` state.

W31.2.4. Checked released image *ObjectReference*

The image when the widget is in the both `Checked` and `Released` state.

W31.2.5. Checked pressed image *ObjectReference*

The image when the widget is in the both `Checked` and `Pressed` state.

W31.2.6. Checked disabled image *ObjectReference*

The image when the Widget is in the both `Checked` and `Disabled` state.

General

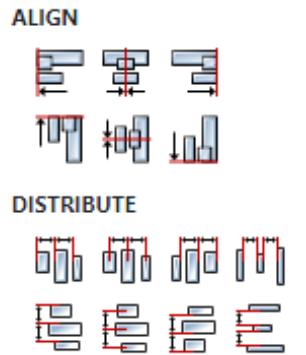
W31.2.7. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W31.2.8. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W31.2.9. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W31.2.10. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W31.2.11. Left unit Enum

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W31.2.12. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W31.2.13. Top unit Enum

The following options are available:

- px – Top is set in pixels.
- % – The top is set as a percentage in relation to the parent height.

W31.2.14. Width Number

The width of the component. It is set in pixels.

W31.2.15. Width unit Enum

The following options are available:

- px – Width is given in pixels.
- % – Width is given as a percentage in relation to the parent width.
- content – Width is automatically set to fit the entire content in width.

W31.2.16. Height *Number*

The height of the component. It is set in pixels.

W31.2.17. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W31.2.18. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W31.2.19. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W31.2.20. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W31.2.21. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W31.2.22. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W31.2.23. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W31.2.24. Click focusable *Boolean*

Add focused state to the object when clicked.

W31.2.25. Checkable *Boolean*

Toggle checked state when the object is clicked.

W31.2.26. Scrollable *Boolean*

Make the object scrollable.

W31.2.27. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W31.2.28. Scroll momentum Boolean

Make the object scroll further when "thrown".

W31.2.29. Scroll one Boolean

Allow scrolling only one snappable children.

W31.2.30. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W31.2.31. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W31.2.32. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W31.2.33. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W31.2.34. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W31.2.35. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W31.2.36. Event bubble Boolean

Propagate the events to the parent too.

W31.2.37. Gesture bubble Boolean

Propagate the gestures to the parent.

W31.2.38. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W31.2.39. Ignore layout Boolean

Make the object positionable by the layouts.

W31.2.40. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W31.2.41. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W31.2.42. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W31.2.43. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W31.2.44. Checked *EXPRESSION (boolean)***

Toggled or checked state.

W31.2.45. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W31.2.46. Disabled *EXPRESSION (boolean)*

Disabled state

W31.2.47. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W31.2.48. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W31.2.49. Pressed *Boolean*

Being pressed.

Events**W31.2.50. Event handlers *Array***

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W31.2.51. Inputs *Array***

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W31.2.52. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W31.2.53. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W31.3. Examples

- *LVGL Widgets Demo*



W32.1. Description

The widget is used when we want to enter a number or text. In order for this widget to work, the project must define a page for entering text and a page for entering numbers. See some of the examples listed under *Examples* of how these pages are defined.

W32.2. Properties

Specific

W32.2.1. Data *EXPRESSION (any)*

The variable in which the entered number or text will be stored.

W32.2.2. Input type *Enum*

Choose whether `Number` or `Text` is entered.

W32.2.3. Min *EXPRESSION (any)*

If `Input type` is set to `Number` then this number represents the minimum number that needs to be entered, and if it is set to `Text` then this property represents the minimum number of characters that need to be entered.

W32.2.4. Max *EXPRESSION (any)*

If `Input type` is set to `Number` then this number represents the maximum number that needs to be entered, and if it is set to `Text` then this property represents the maximum number of characters that need to be entered.

W32.2.5. Precision *EXPRESSION (any)*

If `Input type` is set to `Number` then this property defines the precision of the entered number. If a number with higher precision (more decimal places) is entered, then the number will be rounded to this precision. For example if we set it to `0.01` then the number will be rounded to two decimal places.

W32.2.6. Unit *EXPRESSION (any)*

If `Input type` is set to `Number` then this property defines the unit that will be used, i.e. printed to the right of the numerical value.

W32.2.7. Password *Boolean*

If `Input type` is set to `Text` and a password is entered, then this property should be enabled so that `*` is displayed instead of characters when entering the password.

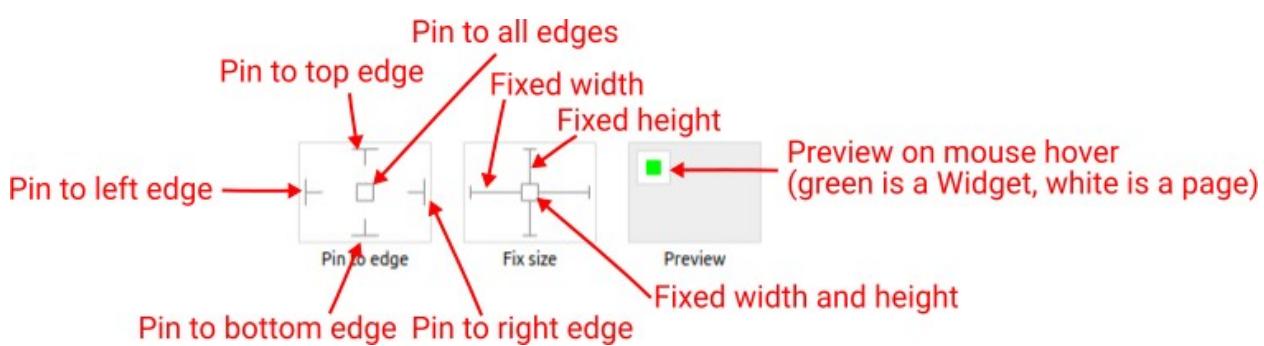
W32.2.8. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W32.2.9. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the *Top* position does not change the value. If *Pin to top edge* is not selected, then the *Top* position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

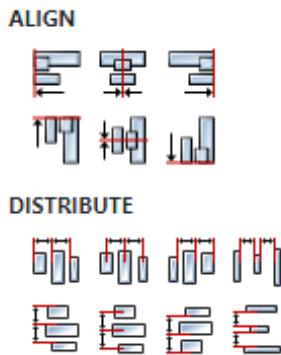
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W32.2.10. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W32.2.11. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W32.2.12. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W32.2.13. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the *Top*, *Width* and *Height* properties), simple mathematical expressions can be used. When we enter an expression and press enter, the ex-

pression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W32.2.14. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W32.2.15. Width Number

The width of the component. It is set in pixels.

W32.2.16. Height Number

The height of the component. It is set in pixels.

W32.2.17. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W32.2.18. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W32.2.19. Default Object

Style used when rendering of the Widget.

Events

W32.2.20. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W32.2.21. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W32.2.22. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W32.2.23. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W32.3. Examples

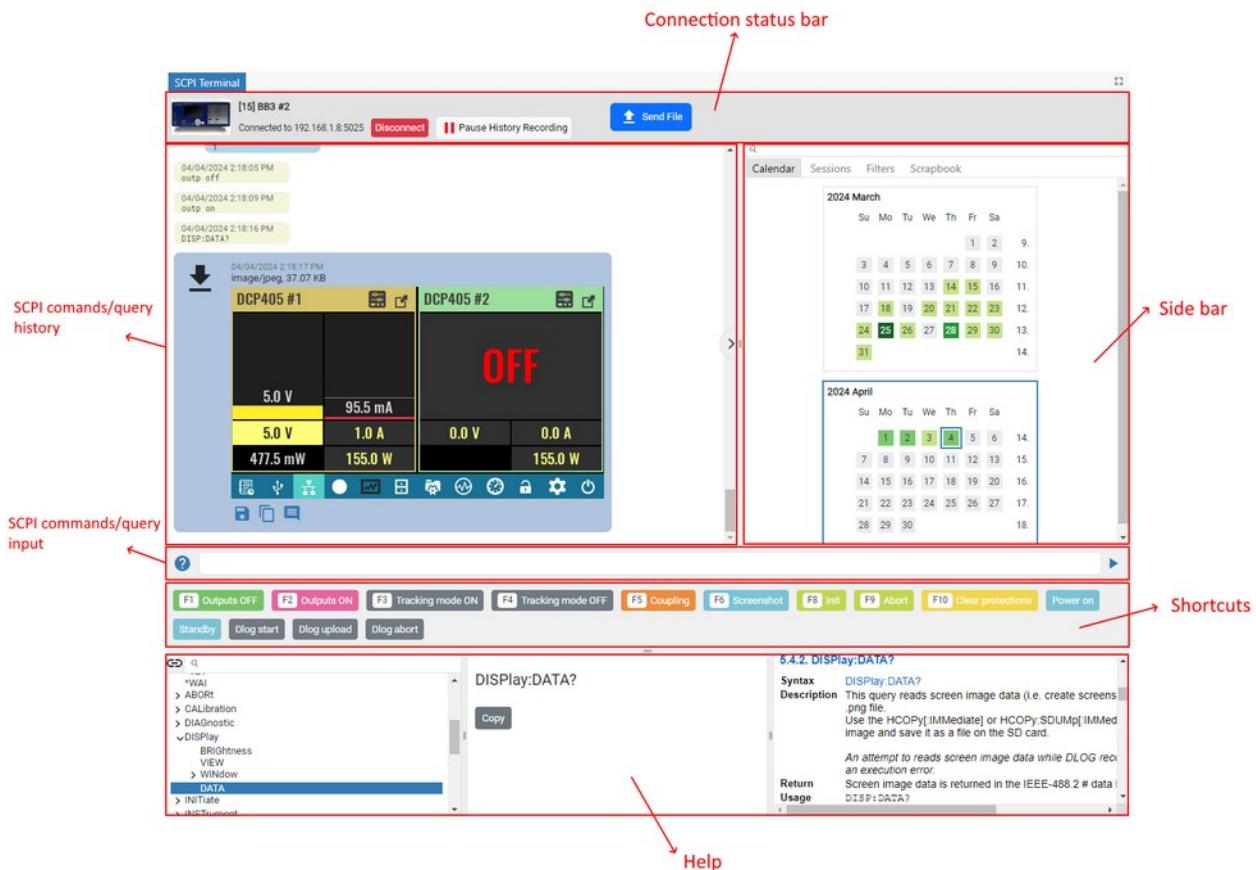
- *eez-gui-widgets-demo*
- *stm32f469i-disco-eez-flow-demo*

W33. InstrumentTerminal



W33.1. Description

This widget allows interaction with the instrument. It consists of several parts, some of them can be hidden with associated properties.



W33.2. Properties

Specific

W33.2.1. Instrument *EXPRESSION (object:Instrument)*

Selected instrument object.

W33.2.2. Show connection status bar *EXPRESSION (boolean)*

Show/Hide instrument connection status bar.

W33.2.3. Show shortcuts *EXPRESSION (boolean)*

Show/Hide shortctus panel.

W33.2.4. Show help *EXPRESSION (boolean)*

Show/Hide commands help panel.

W33.2.5. Show side bar *EXPRESSION (boolean)*

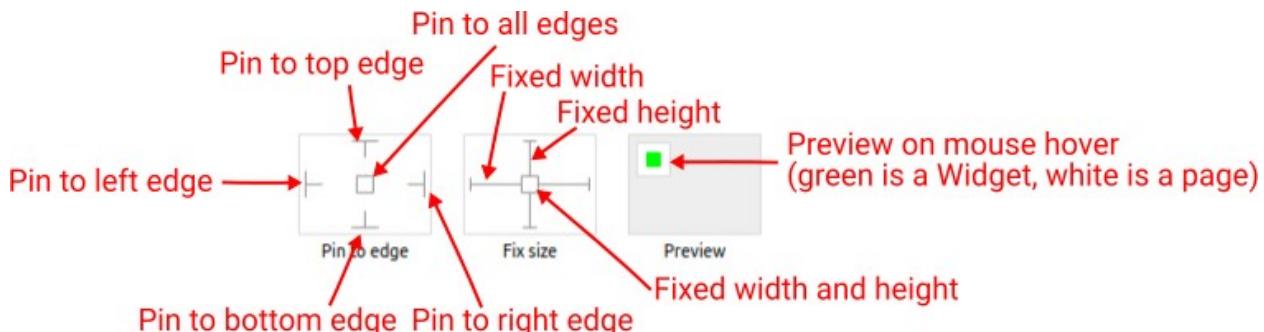
Show/Hide side bar with history search options.

W33.2.6. Visible EXPRESSION (boolean)

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size**W33.2.7. Resizing Any**

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



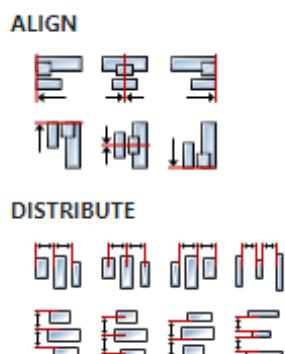
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W33.2.8. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W33.2.9. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W33.2.10. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2`.

W33.2.11. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W33.2.12. Width Number

The width of the component. It is set in pixels.

W33.2.13. Height Number

The height of the component. It is set in pixels.

W33.2.14. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W33.2.15. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Events

W33.2.16. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W33.2.17. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a

value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W33.2.18. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W33.2.19. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W33.2.20. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W33.3. Examples

- BB3 SCPI Terminal and Dashboard

W34. Keyboard



W34.1. Description

The virtual (on screen) keyboard to write texts into a Text area.
More info ([link](#))

W34.2. Properties

Specific

W34.2.1. Textarea *Enum*

The name of `Textarea` Widget attached to this Widget.

W34.2.2. Mode *Enum*

The following modes are available:

- `TEXT_LOWER` – Display lower case letters.
- `TEXT_UPPER` – Display upper case letters.
- `SPECIAL` – Display special characters.
- `NUMBER` – Display numbers, +/- sign, and decimal dot.
- `USER_1` to `USER_4` – User-defined modes.

General

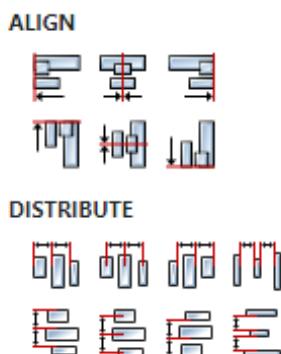
W34.2.3. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W34.2.4. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W34.2.5. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W34.2.6. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2`.

W34.2.7. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W34.2.8. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W34.2.9. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W34.2.10. Width *Number*

The width of the component. It is set in pixels.

W34.2.11. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W34.2.12. Height *Number*

The height of the component. It is set in pixels.

W34.2.13. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W34.2.14. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set

the title of the tab that contains this widget.

Style

W34.2.15. Use style ObjectReference

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W34.2.16. Hidden EXPRESSION (boolean)

Make the object hidden.

W34.2.17. Hidden flag type Enum

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W34.2.18. Clickable EXPRESSION (boolean)

Make the object clickable by input devices.

W34.2.19. Clickable flag type Enum

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W34.2.20. Click focusable Boolean

Add focused state to the object when clicked.

W34.2.21. Checkable Boolean

Toggle checked state when the object is clicked.

W34.2.22. Scrollable Boolean

Make the object scrollable.

W34.2.23. Scroll elastic Boolean

Allow scrolling inside but with slower speed.

W34.2.24. Scroll momentum Boolean

Make the object scroll further when "thrown".

W34.2.25. Scroll one Boolean

Allow scrolling only one snappable children.

W34.2.26. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W34.2.27. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W34.2.28. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W34.2.29. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W34.2.30. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W34.2.31. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W34.2.32. Event bubble Boolean

Propagate the events to the parent too.

W34.2.33. Gesture bubble Boolean

Propagate the gestures to the parent.

W34.2.34. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W34.2.35. Ignore layout Boolean

Make the object positionable by the layouts.

W34.2.36. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W34.2.37. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W34.2.38. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W34.2.39. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W34.2.40. Checked EXPRESSION (boolean)**

Toggled or checked state.

W34.2.41. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W34.2.42. Disabled *EXPRESSION (boolean)*

Disabled state

W34.2.43. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W34.2.44. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W34.2.45. Pressed *Boolean*

Being pressed.

Events**W34.2.46. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W34.2.47. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W34.2.48. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W34.2.49. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W34.3. Examples

- *LVGL Widgets Demo*

W35. Label



W35.1. Description

A Widget used to display text.
More info ([link](#))

W35.2. Properties

Specific

W35.2.1. Text *EXPRESSION (string)*

Text to be displayed.

W35.2.2. Text type *Enum*

Here we can choose whether the `Text` property will be calculated from the Expression.

W35.2.3. Preview value *String*

This is optional property. If specified then the content of the Label in the project editor will be this value not the expression entered in Text property.

W35.2.4. Long mode *Enum*

If `content` is selected for `Width` and `Height` then this item has no effect because the size of the Widget will be automatically set to fit the entire text, but if the size of the Widget is set manually (px or %) then using of this item defines one of the following ways in which the text will be split if it does not fit within the limits of the Widget:

- `WRAP` – Wrap too long lines. If the `Height` is set to `content` it will be expanded, otherwise the text will be clipped (Default).
- `DOT` – Replaces the last 3 characters from bottom right corner of the label with dots.
- `SCROLL` – If the text is wider than the label scroll it horizontally back and forth. If it's higher, scroll vertically. Only one direction is scrolled and horizontal scrolling has higher precedence.
- `SCROLL_CIRCULAR` – If the text is wider than the label scroll it horizontally continuously. If it's higher, scroll vertically. Only one direction is scrolled and horizontal scrolling has higher precedence.
- `CLIP` – Simply clip the parts of the text outside the label.

W35.2.5. Recolor *Boolean*

If this is enabled then, in the text, we can use commands to recolor parts of the text. For example: "Write a #ff0000 red# word".

General

W35.2.6. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

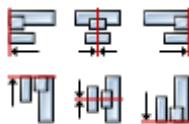
Position and size

W35.2.7. Align and distribute *Any*

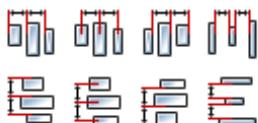
Alignment icons and component distribution. Alignment icons appear when two or more compo-

nents are selected, and distribution icons appear when three or more components are selected.

ALIGN

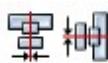


DISTRIBUTE



W35.2.8. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W35.2.9. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W35.2.10. Left unit Enum

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W35.2.11. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W35.2.12. Top unit Enum

The following options are available:

- px – Top is set in pixels.
- % – The top is set as a percentage in relation to the parent height.

W35.2.13. Width Number

The width of the component. It is set in pixels.

W35.2.14. Width unit Enum

The following options are available:

- px – Width is given in pixels.
- % – Width is given as a percentage in relation to the parent width.
- content – Width is automatically set to fit the entire content in width.

W35.2.15. Height *Number*

The height of the component. It is set in pixels.

W35.2.16. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W35.2.17. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W35.2.18. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W35.2.19. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W35.2.20. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W35.2.21. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W35.2.22. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W35.2.23. Click focusable *Boolean*

Add focused state to the object when clicked.

W35.2.24. Checkable *Boolean*

Toggle checked state when the object is clicked.

W35.2.25. Scrollable *Boolean*

Make the object scrollable.

W35.2.26. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W35.2.27. Scroll momentum Boolean

Make the object scroll further when "thrown".

W35.2.28. Scroll one Boolean

Allow scrolling only one snappable children.

W35.2.29. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W35.2.30. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W35.2.31. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W35.2.32. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W35.2.33. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W35.2.34. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W35.2.35. Event bubble Boolean

Propagate the events to the parent too.

W35.2.36. Gesture bubble Boolean

Propagate the gestures to the parent.

W35.2.37. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W35.2.38. Ignore layout Boolean

Make the object positionable by the layouts.

W35.2.39. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W35.2.40. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W35.2.41. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W35.2.42. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W35.2.43. Checked** *EXPRESSION (boolean)*

Toggled or checked state.

W35.2.44. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W35.2.45. Disabled *EXPRESSION (boolean)*

Disabled state

W35.2.46. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W35.2.47. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W35.2.48. Pressed *Boolean*

Being pressed.

Events**W35.2.49. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W35.2.50. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W35.2.51. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W35.2.52. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W36. Led



W36.1. Description

The LEDs are rectangle-like (or circle) object whose brightness can be adjusted. With lower brightness the colors of the LED become darker.

More info ([link](#))

W36.2. Properties

Specific

W36.2.1. Color *EXPRESSION (integer)*

You can set the color of the LED. This will be used as background color, border color, and shadow color.

W36.2.2. Color type *Enum*

Here we can choose whether the `Color` property will be calculated from the Expression.

W36.2.3. Brightness *EXPRESSION (integer)*

Brightness of the LED. The brightness should be between 0 (darkest) and 255 (lightest).

W36.2.4. Brightness type *Enum*

Here we can choose whether the `Brightness` property will be calculated from the Expression.

General

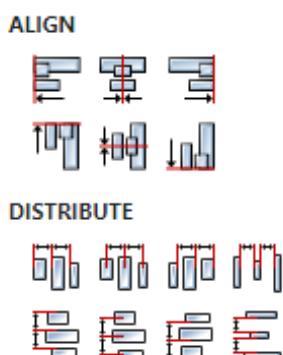
W36.2.5. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W36.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W36.2.7. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W36.2.8. Left** *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W36.2.9. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W36.2.10. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W36.2.11. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W36.2.12. Width *Number*

The width of the component. It is set in pixels.

W36.2.13. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W36.2.14. Height *Number*

The height of the component. It is set in pixels.

W36.2.15. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

W36.2.16. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W36.2.17. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W36.2.18. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W36.2.19. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W36.2.20. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W36.2.21. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W36.2.22. Click focusable *Boolean*

Add focused state to the object when clicked.

W36.2.23. Checkable *Boolean*

Toggle checked state when the object is clicked.

W36.2.24. Scrollable *Boolean*

Make the object scrollable.

W36.2.25. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W36.2.26. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W36.2.27. Scroll one *Boolean*

Allow scrolling only one snappable children.

W36.2.28. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W36.2.29. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W36.2.30. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W36.2.31. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W36.2.32. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W36.2.33. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W36.2.34. Event bubble *Boolean*

Propagate the events to the parent too.

W36.2.35. Gesture bubble *Boolean*

Propagate the gestures to the parent.

W36.2.36. Adv hitest *Boolean*

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W36.2.37. Ignore layout *Boolean*

Make the object positionable by the layouts.

W36.2.38. Floating *Boolean*

Do not scroll the object when the parent scrolls and ignore layout.

W36.2.39. Overflow visible *Boolean*

Do not clip the children's content to the parent's boundary.

W36.2.40. Scrollbar mode *Enum*

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W36.2.41. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States

W36.2.42. Checked *EXPRESSION (boolean)*

Toggled or checked state.

W36.2.43. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W36.2.44. Disabled *EXPRESSION (boolean)*

Disabled state

W36.2.45. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W36.2.46. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W36.2.47. Pressed *Boolean*

Being pressed.

Events**W36.2.48. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W36.2.49. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W36.2.50. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W36.2.51. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W37. Line



W37.1. Description

The Line object is capable of drawing straight lines between a set of points.
More info ([link](#))

W37.2. Properties

Specific

W37.2.1. Points *String*

List of points given as: `x1,y1 x2, y2 x3, y3 ...`, for example: `0,0 50,50 100,0 150,50 200,0`

W37.2.2. Invert Y *Boolean*

By default, the `y == 0` point is in the top of the object. It might be counter-intuitive in some cases so the y coordinates can be inverted with this property. In this case, `y == 0` will be the bottom of the object. `y invert` is disabled by default.

General

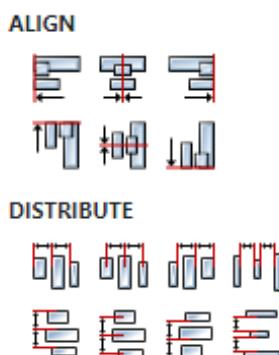
W37.2.3. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W37.2.4. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W37.2.5. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W37.2.6. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W37.2.7. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W37.2.8. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W37.2.9. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W37.2.10. Width *Number*

The width of the component. It is set in pixels.

W37.2.11. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W37.2.12. Height *Number*

The height of the component. It is set in pixels.

W37.2.13. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W37.2.14. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W37.2.15. Use style ObjectReference

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W37.2.16. Hidden EXPRESSION (boolean)**

Make the object hidden.

W37.2.17. Hidden flag type Enum

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W37.2.18. Clickable EXPRESSION (boolean)

Make the object clickable by input devices.

W37.2.19. Clickable flag type Enum

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W37.2.20. Click focusable Boolean

Add focused state to the object when clicked.

W37.2.21. Checkable Boolean

Toggle checked state when the object is clicked.

W37.2.22. Scrollable Boolean

Make the object scrollable.

W37.2.23. Scroll elastic Boolean

Allow scrolling inside but with slower speed.

W37.2.24. Scroll momentum Boolean

Make the object scroll further when "thrown".

W37.2.25. Scroll one Boolean

Allow scrolling only one snappable children.

W37.2.26. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W37.2.27. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W37.2.28. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W37.2.29. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W37.2.30. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W37.2.31. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W37.2.32. Event bubble Boolean

Propagate the events to the parent too.

W37.2.33. Gesture bubble Boolean

Propagate the gestures to the parent.

W37.2.34. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W37.2.35. Ignore layout Boolean

Make the object positionable by the layouts.

W37.2.36. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W37.2.37. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W37.2.38. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W37.2.39. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W37.2.40. Checked EXPRESSION (boolean)**

Toggled or checked state.

W37.2.41. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W37.2.42. Disabled *EXPRESSION (boolean)*

Disabled state

W37.2.43. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W37.2.44. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W37.2.45. Pressed *Boolean*

Being pressed.

Events**W37.2.46. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

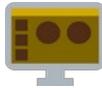
Flow**W37.2.47. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W37.2.48. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through

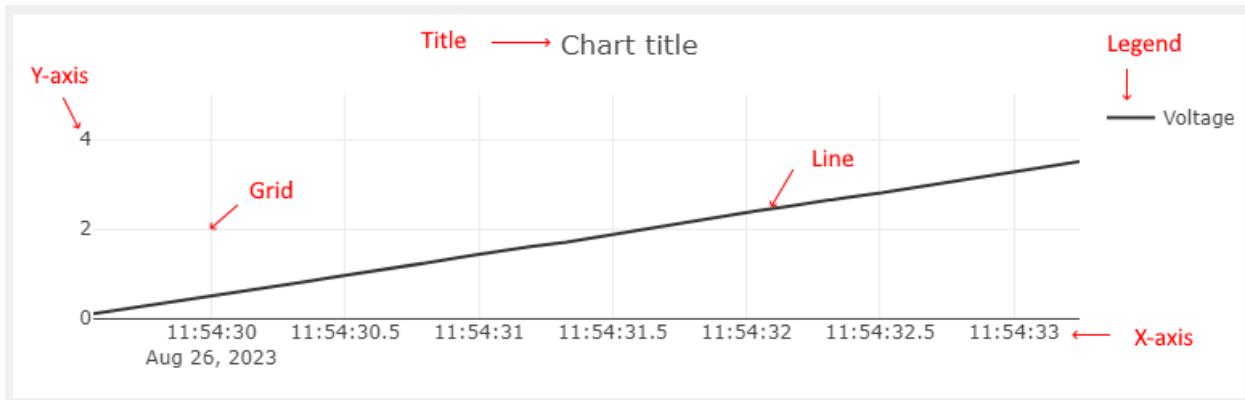
W38. LineChart (Dashboard)



W38.1. Description

Displays a Line chart consisting of the following parts:

- Title
- X Axis
- Y axis
- A legend
- Grid
- One or more lines



At the beginning of the chart there is not a single point on the lines. In order to add a point, it is necessary to pass the data through `value` input. One point is added for each applied data on that input. The X and Y values of that point on all lines should then be calculated from the received data. For example the received data can be a structure that has an X value and a Y value for each line.

W38.2. Properties

Specific

W38.2.1. X value *EXPRESSION (any)*

Defines the value on the X-axis for the added point. It can be set to the current time with `Date.now()` or some other value, but care must be taken to increase the value with each newly added point.

W38.2.2. Lines *Array*

Defines one or more lines on the Y-axis. The following must be specified for each line:

- `Label` – The name of the line that is displayed in the Legend.
- `Color` – Color of the line.
- `Value` – The value on the Y axis for the added point.

W38.2.3. Title *String*

Name of the chart.

W38.2.4. Display mode bar *Enum*

When the mode bar with buttons will be displayed in the top right corner of the chart, possible options are: `Hover`, `Always` and `Never`.

W38.2.5. Show legend Boolean

It should be set if we want to display the legend.

W38.2.6. Show grid Boolean

It should be set if we want to display the grid.

W38.2.7. Show zero lines Boolean

It should be set if we want to display zero lines.

W38.2.8. Show X axis Boolean

It should be set if we want to display the X-axis.

W38.2.9. X axis tick suffix String

If specified, this string value will be appended to the x axis values. Use this to set the unit of X Axis values.

W38.2.10. X axis range option Enum

Here we have two options:

- `Floating` – X-axis range will be automatically selected based on the X value at all points.
- `Fixed` – X-axis range is set via `X axis range from` and `X axis range to` items.

W38.2.11. X axis range from EXPRESSION (double)

If `Fixed` is selected for `X axis range option`, then the lower limit of the X-axis range is set with this item.

W38.2.12. X axis range to EXPRESSION (double)

If `Fixed` is selected for `X axis range option`, then the upper limit of the X-axis range is set with this item.

W38.2.13. Show Y axis Boolean

It should be set if we want to display the Y-axis.

W38.2.14. Y axis tick suffix String

If specified, this string value will be appended to the Y axis values. Use this to set the unit of Y Axis values.

W38.2.15. Y axis range option Enum

Here we have two options:

- `Floating` – Y-axis range will be automatically selected based on the Y value at all points.
- `Fixed` – Y-axis range is set via `Y axis range from` and `Y axis range to` items.

W38.2.16. Y axis range from EXPRESSION (double)

If `Fixed` is selected for `Y axis range option`, then the lower limit of the Y-axis range is set with this item.

W38.2.17. Y axis range to EXPRESSION (double)

If `Fixed` is selected for `Y axis range option`, then the upper limit of the Y-axis range is set with this item.

W38.2.18. Max points *EXPRESSION (integer)*

The maximum number of points that will be displayed.

W38.2.19. Margin *Object*

Manually selected margin values between the Widget borders and the chart itself within the Widget. It is necessary to leave an empty space for Title (displayed above the chart, so the appropriate Top margin should be selected), X-axis (displayed below the chart, Bottom margin), Y-axis (displayed to the left of the chart, Left margin) and Legend (displayed to the right of the chart, Right margin).

W38.2.20. Marker *EXPRESSION (float)*

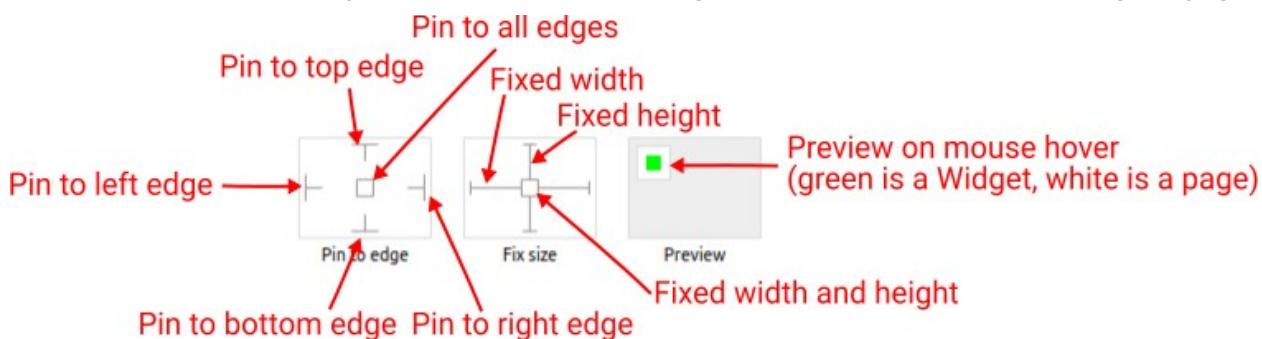
At this position, a vertical line will be displayed inside the chart using **Marker** style.

W38.2.21. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size**W38.2.22. Resizing** *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



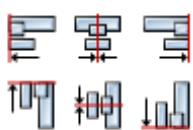
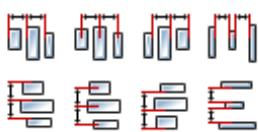
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W38.2.23. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W38.2.24. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W38.2.25. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W38.2.26. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W38.2.27. Width Number

The width of the component. It is set in pixels.

W38.2.28. Height Number

The height of the component. It is set in pixels.

W38.2.29. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W38.2.30. Tab title EXPRESSION (string)**

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W38.2.31. Default Object**

Style used when rendering of the Widget.

W38.2.32. Marker Object

Style used to render the marker.

Events**W38.2.33. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W38.2.34. Output widget handle Boolean**

If enabled then a new output named @Widget will be added. In runtime, upon a widget creation, a value of type widget will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is AddToInstrumentHistory action component when Plotly is selected for the Item type property. Then it is necessary to set the property Plotly widget to the reference to LineChart widget.

W38.2.35. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W38.2.36. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W38.2.37. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W38.3. Inputs**W38.3.1. reset DATA(any) / OPTIONAL**

If we want to erase all the points on the chart, it is necessary to send a signal to this input.

W38.3.2. value DATA(any) / MANDATORY

The input to which the value of the point that we want to add to the chart is sent. When the maximum number of points, which is set through the `Max points` item, is reached, then the oldest added point will be deleted.

W38.4. Examples

- *Dashboard Widgets Demo*

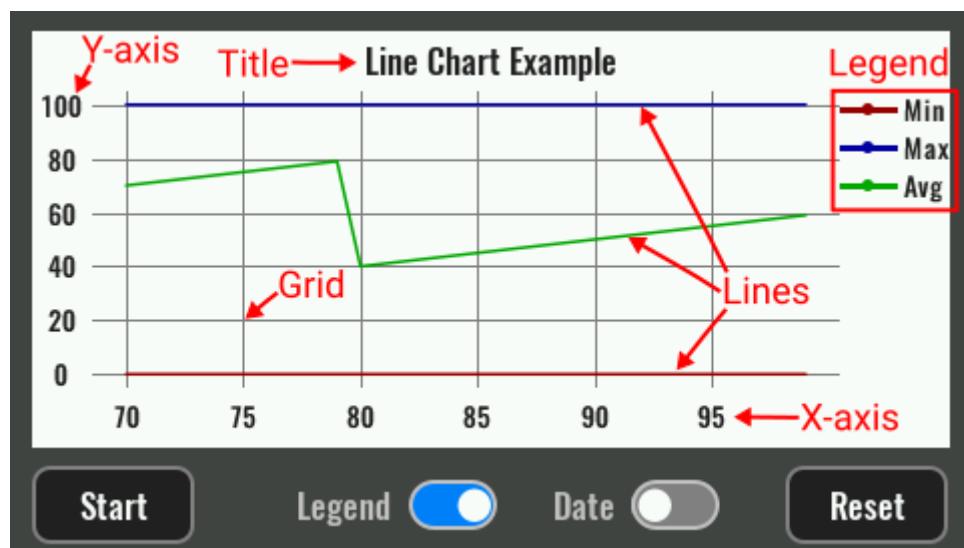
W39. LineChart (EEZ-GUI)



W39.1. Description

Displays a Line chart consisting of the following parts:

- Title
- X Axis
- Y axis
- A legend
- Grid
- One or more lines



At the beginning of the chart there is not a single point on the lines. In order to add a point, it is necessary to pass the data through `value` input. One point is added for each applied data on that input. The X and Y values of that point on all lines should then be calculated from the received data. For example the received data can be a structure that has an X value and a Y value for each line.

W39.2. Properties

Specific

W39.2.1. X value *EXPRESSION (any)*

Defines the value on the X-axis for the added point. It can be set to the current time with `Date.now()` or some other value, but care must be taken to increase the value with each newly added point.

W39.2.2. Lines *Array*

Defines one or more lines on the Y-axis. The following must be specified for each line:

- `Label` – The name of the line that is displayed in the Legend.
- `Color` – Color of the line.
- `Line width` – The thickness of the line in pixels.
- `Value` – The value on the Y axis for the added point.

W39.2.3. Show title *EXPRESSION (boolean)*

It should be set if we want to render the title.

W39.2.4. Show legend *EXPRESSION (boolean)*

It should be set if we want to render the legend.

W39.2.5. Show X axis *EXPRESSION (boolean)*

It should be set if we want to render the X-axis.

W39.2.6. Show Y axis *EXPRESSION (boolean)*

It should be set if we want to render the Y-axis.

W39.2.7. Show grid *EXPRESSION (boolean)*

It should be set if we want to render the grid.

W39.2.8. Title *EXPRESSION (string)*

Name of the chart.

W39.2.9. Y axis range option *Enum*

Here we have two options:

- `Floating` – Y-axis range will be automatically selected based on the Y value at all points.
- `Fixed` – Y-axis range is set via `Y axis range from` and `Y axis range to` items.

W39.2.10. Y axis range from *EXPRESSION (double)*

If `Fixed` is selected for `Y axis range option`, then the lower limit of the Y-axis range is set with this item.

W39.2.11. Y axis range to *EXPRESSION (double)*

If `Fixed` is selected for `Y axis range option`, then the upper limit of the Y-axis range is set with this item.

W39.2.12. Max points *Number*

The maximum number of points that will be displayed.

W39.2.13. Margin *Object*

Manually selected margin values between the Widget borders and the chart itself within the Widget. It is necessary to leave an empty space for Title (displayed above the chart, so the appropriate `Top` margin should be selected), X-axis (displayed below the chart, `Bottom` margin), Y-axis (displayed to the left of the chart, `Left` margin) and Legend (displayed to the right of the chart, `Right` margin).

W39.2.14. Marker *EXPRESSION (float)*

At this position, a vertical line will be displayed inside the chart using `Marker` style.

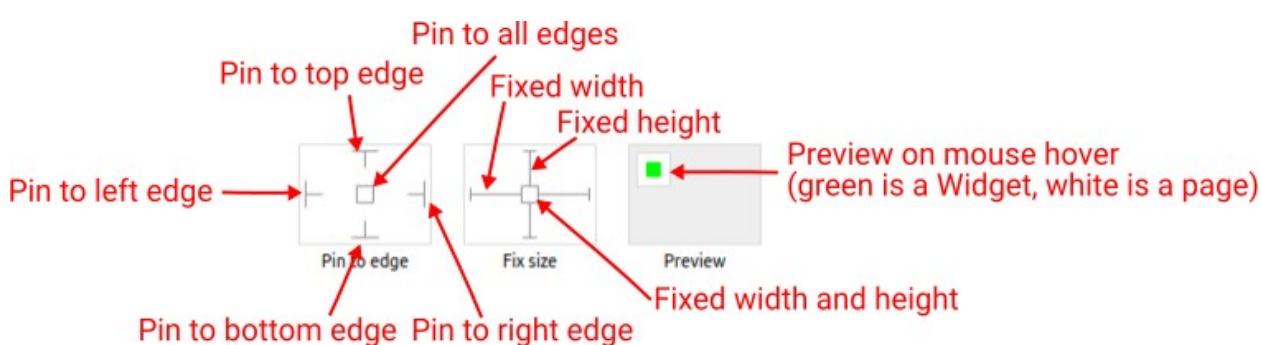
W39.2.15. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W39.2.16. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

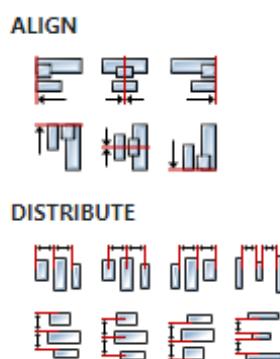
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W39.2.17. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W39.2.18. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W39.2.19. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W39.2.20. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the ex-

pression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W39.2.21. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W39.2.22. Width Number

The width of the component. It is set in pixels.

W39.2.23. Height Number

The height of the component. It is set in pixels.

W39.2.24. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W39.2.25. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W39.2.26. Default Object

Style used when rendering of the Widget.

W39.2.27. Title Object

Style used to render the title.

W39.2.28. Legend Object

Style used to render the legend.

W39.2.29. X axis Object

Style used to render the X-axis.

W39.2.30. Y axis Object

Style used to render the Y-axis.

W39.2.31. Marker Object

Style used to render the marker.

Events

W39.2.32. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W39.2.33. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W39.2.34. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W39.2.35. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W39.3. Inputs

W39.3.1. reset DATA(any) / OPTIONAL

If we want to erase all the points on the chart, it is necessary to send a signal to this input.

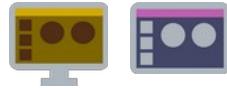
W39.3.2. value DATA(any) / MANDATORY

The input to which the value of the point that we want to add to the chart is sent. When the maximum number of points, which is set through the Max points item, is reached, then the oldest added point will be deleted.

W39.4. Examples

- *Line Chart*

W40. List



W40.1. Description

Use this Widget when you want to display the same Widget multiple times. This Widget has one Child widget under it, and the number of times it will be displayed depends on the `Data` property. Multiplied Widgets can be displayed by first filling rows or columns:

It wouldn't be very useful if multiplied Widgets always had the same content, that's why there is a system variable `$index` that tells us in which order the Widget is rendered. That variable is zero based, that means when its value is 0 then the first Widget is rendered, when its value is 1 then the second Widget is rendered and so on. That `$index` can then be used within the expression of the property of the Child widget, and in this way it is achieved that each rendered Widget has different content (e.g. the `Text` Widget can display a string that is taken from an array variable: `country_cities[$index].country`).

W40.2. Properties

Specific

W40.2.1. Data *EXPRESSION (any)*

Determines how many times the Child widget will be multiplied, i.e. the number of elements in the list. The value of this property can be an integer and then it is the number of elements, and if the value of this property is an array, then the number of elements in the list is equal to the number of elements in that array.

In the case of *EEZ-GUI* projects, the value of this property can also be `struct:$ScrollbarState`. The same structure is used for the `ScrollBar` Widget, which can then be connected to the `List` Widget via the `struct:$ScrollbarState` variable and thus enable scrolling of the list in case the total number of list elements is greater than the number of elements that fit within the `List` Widget.

More about the `struct:$ScrollbarState` system structure can be found in the `ScrollBar` Widget documentation.

W40.2.2. List type *Enum*

Defines vertical or horizontal orientation.

W40.2.3. Gap *Number*

The distance in pixels between two grid elements.

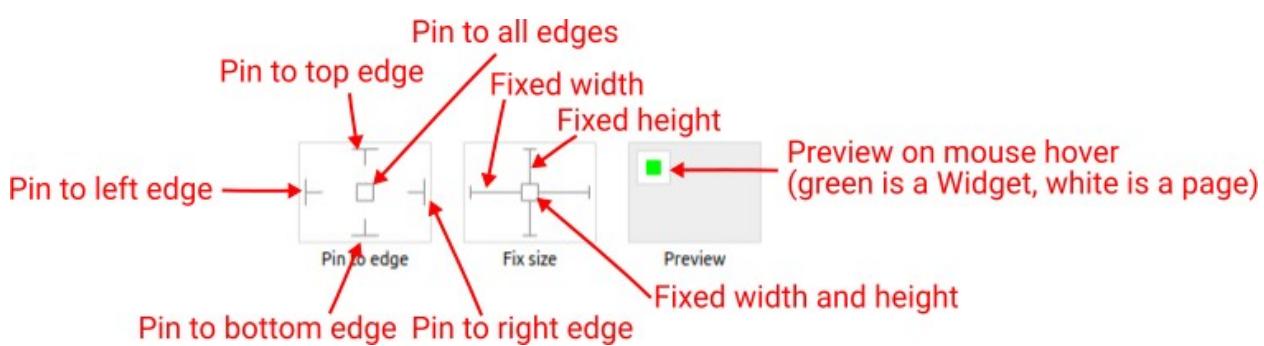
W40.2.4. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W40.2.5. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

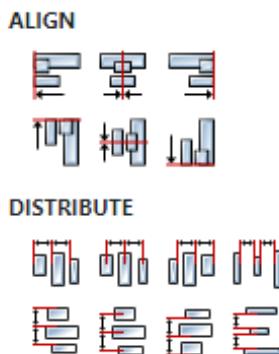
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W40.2.6. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W40.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W40.2.8. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W40.2.9. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the *Top*, *Width* and *Height* properties), simple mathematical expressions can be used. When we enter an expression and press enter, the ex-

pression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W40.2.10. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W40.2.11. Width Number

The width of the component. It is set in pixels.

W40.2.12. Height Number

The height of the component. It is set in pixels.

W40.2.13. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W40.2.14. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W40.2.15. Default Object

Style used when rendering the background of the Widget.

Events

W40.2.16. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W40.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W40.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W40.2.19. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W40.3. Examples

- *eez-gui-widgets-demo*
- *CSV*
- *JSON*
- *MQTT*
- *Simple HTTP*
- *Charts*
- *Regexp String*
- *Multi-Language*

W41. List (Dashboard)



W41.1. Description

Use this Widget when you want to display the same Widget multiple times. This Widget has one Child widget under it, and the number of times it will be displayed depends on the `Data` property. Multiplied Widgets can be displayed by first filling rows or columns:

It wouldn't be very useful if multiplied Widgets always had the same content, that's why there is a system variable `$index` that tells us in which order the Widget is rendered. That variable is zero based, that means when its value is 0 then the first Widget is rendered, when its value is 1 then the second Widget is rendered and so on. That `$index` can then be used within the expression of the property of the Child widget, and in this way it is achieved that each rendered Widget has different content (e.g. the `Text` Widget can display a string that is taken from an array variable: `country_cities[$index].country`).

W41.2. Properties

Specific

W41.2.1. Data *EXPRESSION (any)*

Determines how many times the Child widget will be multiplied, i.e. the number of elements in the list. The value of this property can be an integer and then it is the number of elements, and if the value of this property is an array, then the number of elements in the list is equal to the number of elements in that array.

In the case of *EEZ-GUI* projects, the value of this property can also be `struct:$ScrollbarState`. The same structure is used for the `ScrollBar` Widget, which can then be connected to the `List` Widget via the `struct:$ScrollbarState` variable and thus enable scrolling of the list in case the total number of list elements is greater than the number of elements that fit within the `List` Widget.

More about the `struct:$ScrollbarState` system structure can be found in the `ScrollBar` Widget documentation.

W41.2.2. List type *Enum*

Defines vertical or horizontal orientation.

W41.2.3. Gap *Number*

The distance in pixels between two grid elements.

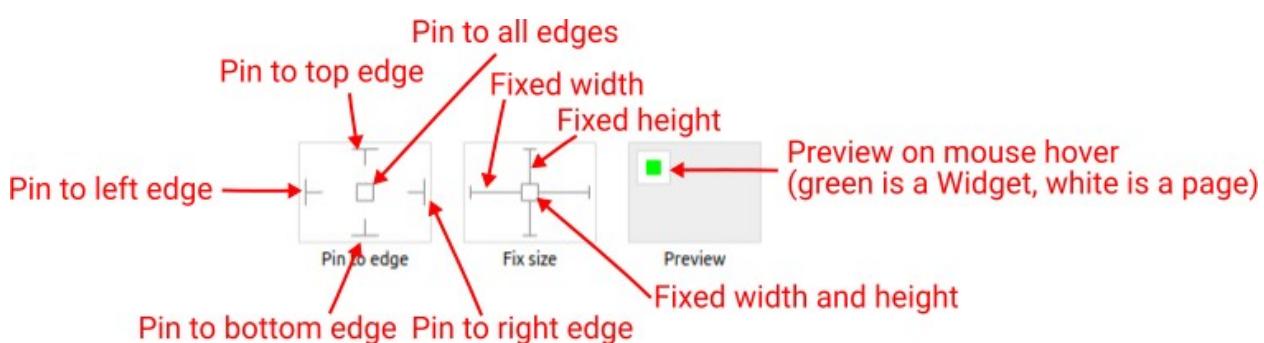
W41.2.4. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W41.2.5. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

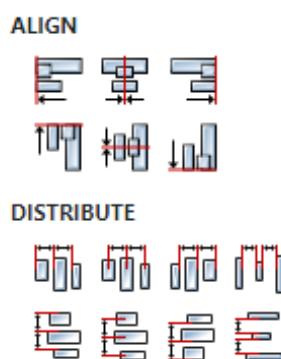
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W41.2.6. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W41.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W41.2.8. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W41.2.9. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the **Top**, **Width** and **Height** properties), simple mathematical expressions can be used. When we enter an expression and press enter, the ex-

pression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W41.2.10. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W41.2.11. Width Number

The width of the component. It is set in pixels.

W41.2.12. Height Number

The height of the component. It is set in pixels.

W41.2.13. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W41.2.14. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W41.2.15. Default Object

Style used when rendering the background of the Widget.

Events

W41.2.16. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W41.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W41.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W41.2.19. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W41.3. Examples

- *eez-gui-widgets-demo*
- *CSV*
- *JSON*
- *MQTT*
- *Simple HTTP*
- *Charts*
- *Regexp String*
- *Multi-Language*

W42. List (LVGL)



W42.1. Description

This widget is work in progress, it means that you can add it to your project and Studio will generate all the code for its creation, but for anything more than that you should do it in your custom code, for example after `ui_init()` has been called.

More info ([link](#))

W42.2. Properties

General

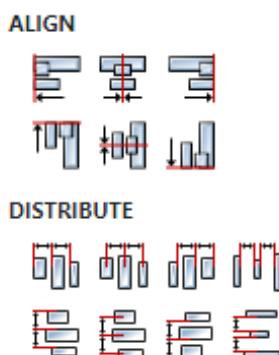
W42.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W42.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W42.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W42.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W42.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W42.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W42.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W42.2.8. Width *Number*

The width of the component. It is set in pixels.

W42.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W42.2.10. Height *Number*

The height of the component. It is set in pixels.

W42.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W42.2.12. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W42.2.13. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W42.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W42.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W42.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W42.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W42.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W42.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W42.2.20. Scrollable *Boolean*

Make the object scrollable.

W42.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W42.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W42.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W42.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W42.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W42.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W42.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W42.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W42.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W42.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W42.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W42.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W42.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W42.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W42.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W42.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W42.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W42.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W42.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W42.2.40. Disabled EXPRESSION (boolean)

Disabled state

W42.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W42.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W42.2.43. Pressed Boolean

Being pressed.

Events**W42.2.44. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W42.2.45. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W42.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W42.2.47. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W43. Lottie



W43.1. Description

This widget is work in progress, it means that you can add it to your project and Studio will generate all the code for its creation, but for anything more than that you should do it in your custom code, for example after `ui_init()` has been called.

More info ([link](#))

W43.2. Properties

General

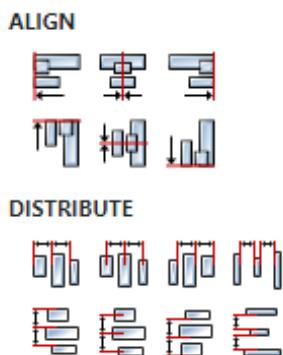
W43.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W43.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W43.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W43.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W43.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W43.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W43.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W43.2.8. Width *Number*

The width of the component. It is set in pixels.

W43.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W43.2.10. Height *Number*

The height of the component. It is set in pixels.

W43.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W43.2.12. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W43.2.13. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W43.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W43.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W43.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W43.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W43.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W43.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W43.2.20. Scrollable *Boolean*

Make the object scrollable.

W43.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W43.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W43.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W43.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W43.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W43.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W43.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W43.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W43.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W43.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W43.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W43.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W43.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W43.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W43.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W43.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W43.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W43.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W43.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W43.2.40. Disabled EXPRESSION (boolean)

Disabled state

W43.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W43.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W43.2.43. Pressed Boolean

Being pressed.

Events**W43.2.44. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W43.2.45. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W43.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W43.2.47. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W44. Markdown



W44.1. Description

Widget for displaying Markdown text.

W44.2. Properties

Specific

W44.2.1. Text *MultilineText*

Markdown text to be displayed.

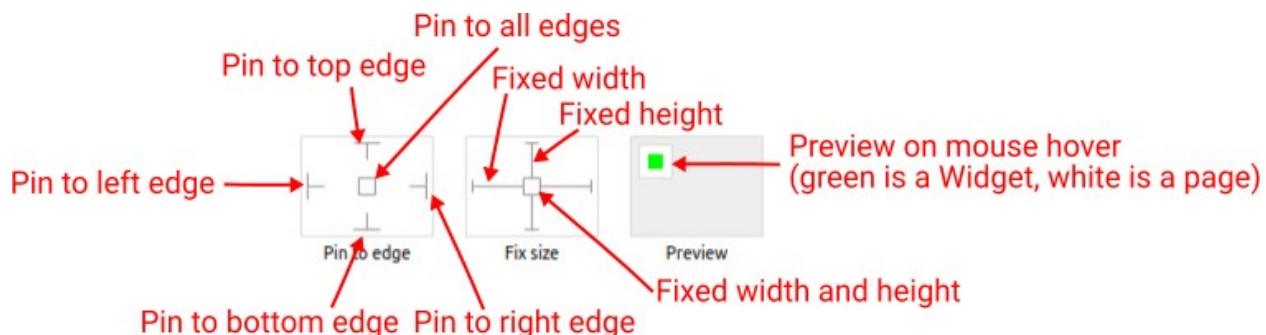
W44.2.2. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W44.2.3. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



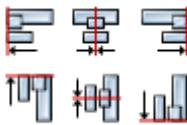
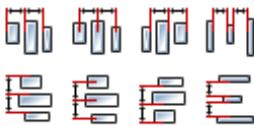
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W44.2.4. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W44.2.5. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W44.2.6. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W44.2.7. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W44.2.8. Width Number

The width of the component. It is set in pixels.

W44.2.9. Height Number

The height of the component. It is set in pixels.

W44.2.10. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W44.2.11. Tab title EXPRESSION (string)**

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W44.2.12. Default Object**

Style used when rendering the background of the Widget.

Events

W44.2.13. Event handlers *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W44.2.14. Output widget handle *Boolean*

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W44.2.15. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W44.2.16. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W44.2.17. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W44.3. Examples

- *Dashboard Widgets Demo*

W45. Menu



W45.1. Description

This widget is work in progress, it means that you can add it to your project and Studio will generate all the code for its creation, but for anything more than that you should do it in your custom code, for example after `ui_init()` has been called.

More info ([link](#))

W45.2. Properties

General

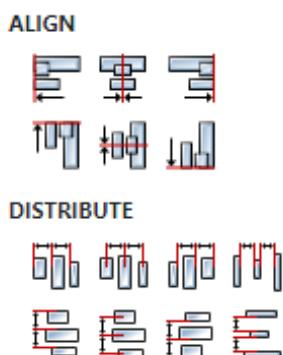
W45.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W45.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W45.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W45.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W45.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W45.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W45.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W45.2.8. Width *Number*

The width of the component. It is set in pixels.

W45.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W45.2.10. Height *Number*

The height of the component. It is set in pixels.

W45.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W45.2.12. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W45.2.13. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W45.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W45.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W45.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W45.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W45.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W45.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W45.2.20. Scrollable *Boolean*

Make the object scrollable.

W45.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W45.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W45.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W45.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W45.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W45.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W45.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W45.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W45.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W45.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W45.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W45.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W45.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W45.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W45.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W45.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W45.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W45.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W45.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W45.2.40. Disabled EXPRESSION (boolean)

Disabled state

W45.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W45.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W45.2.43. Pressed Boolean

Being pressed.

Events**W45.2.44. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W45.2.45. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W45.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W45.2.47. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W46. MessageBox



W46.1. Description

This widget is work in progress, it means that you can add it to your project and Studio will generate all the code for its creation, but for anything more than that you should do it in your custom code, for example after `ui_init()` has been called.

More info ([link](#))

W46.2. Properties

General

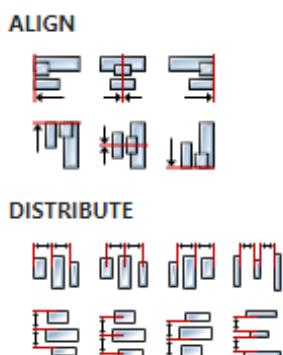
W46.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W46.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W46.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W46.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W46.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W46.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W46.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W46.2.8. Width *Number*

The width of the component. It is set in pixels.

W46.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W46.2.10. Height *Number*

The height of the component. It is set in pixels.

W46.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W46.2.12. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W46.2.13. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W46.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W46.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W46.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W46.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W46.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W46.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W46.2.20. Scrollable *Boolean*

Make the object scrollable.

W46.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W46.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W46.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W46.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W46.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W46.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W46.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W46.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W46.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W46.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W46.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W46.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W46.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W46.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W46.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W46.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W46.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W46.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W46.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W46.2.40. Disabled EXPRESSION (boolean)

Disabled state

W46.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W46.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W46.2.43. Pressed Boolean

Being pressed.

Events**W46.2.44. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W46.2.45. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W46.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W46.2.47. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W47. Meter



W47.1. Description

The `Meter` Widget can visualize data in very flexible ways. It can show arcs, needles, ticks lines and labels.

More info ([link](#))

W47.2. Properties

Specific

W47.2.1. Scales *Array*

List of scale definitions. The Scale has minor and major ticks and labels on the major ticks. One or more indicator can be added to each scales. There are four different type of indicators:

- Needle image
- Needle line
- Scale lines
- Arc

General

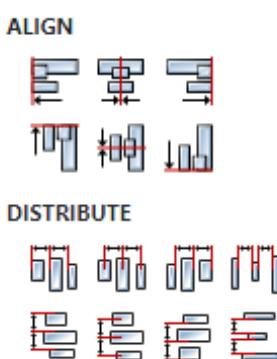
W47.2.2. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W47.2.3. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W47.2.4. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W47.2.5. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W47.2.6. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W47.2.7. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W47.2.8. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W47.2.9. Width *Number*

The width of the component. It is set in pixels.

W47.2.10. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W47.2.11. Height *Number*

The height of the component. It is set in pixels.

W47.2.12. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W47.2.13. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W47.2.14. Use style ObjectReference

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W47.2.15. Hidden EXPRESSION (boolean)**

Make the object hidden.

W47.2.16. Hidden flag type Enum

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W47.2.17. Clickable EXPRESSION (boolean)

Make the object clickable by input devices.

W47.2.18. Clickable flag type Enum

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W47.2.19. Click focusable Boolean

Add focused state to the object when clicked.

W47.2.20. Checkable Boolean

Toggle checked state when the object is clicked.

W47.2.21. Scrollable Boolean

Make the object scrollable.

W47.2.22. Scroll elastic Boolean

Allow scrolling inside but with slower speed.

W47.2.23. Scroll momentum Boolean

Make the object scroll further when "thrown".

W47.2.24. Scroll one Boolean

Allow scrolling only one snappable children.

W47.2.25. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W47.2.26. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W47.2.27. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W47.2.28. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W47.2.29. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W47.2.30. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W47.2.31. Event bubble Boolean

Propagate the events to the parent too.

W47.2.32. Gesture bubble Boolean

Propagate the gestures to the parent.

W47.2.33. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W47.2.34. Ignore layout Boolean

Make the object positionable by the layouts.

W47.2.35. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W47.2.36. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W47.2.37. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W47.2.38. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W47.2.39. Checked EXPRESSION (boolean)**

Toggled or checked state.

W47.2.40. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W47.2.41. Disabled *EXPRESSION (boolean)*

Disabled state

W47.2.42. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W47.2.43. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W47.2.44. Pressed *Boolean*

Being pressed.

Events**W47.2.45. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W47.2.46. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W47.2.47. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W47.2.48. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W47.3. Examples

W48. MultilineText



W48.1. Description

A widget used to display multiple lines text.

W48.2. Properties

Specific

W48.2.1. Text *EXPRESSION (any)*

Text to be displayed. This is an expression and if you only want to display some static text, then that text should be entered in quotation marks. If the expression uses variables, then that expression cannot be calculated in the editor, so the expression will be displayed instead of the text itself.

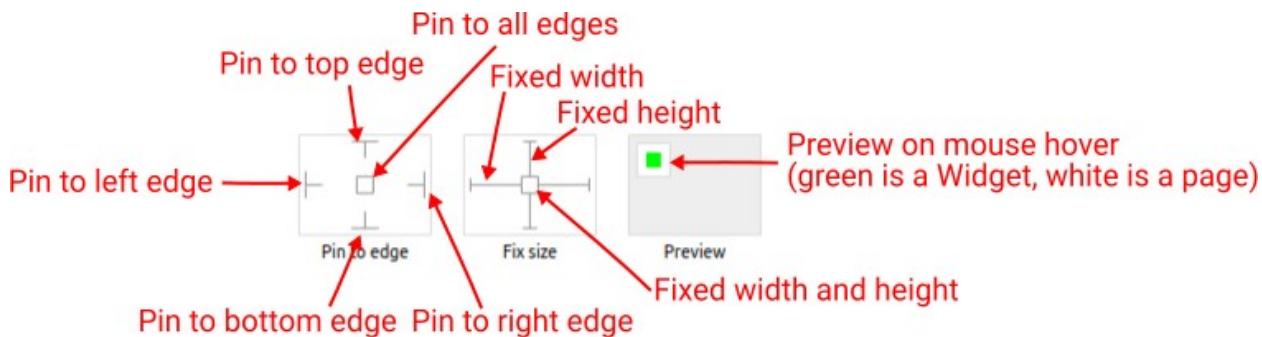
W48.2.2. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W48.2.3. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

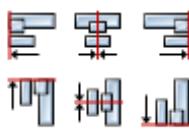
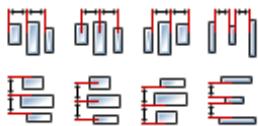
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W48.2.4. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

W48.2.5. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W48.2.6. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W48.2.7. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W48.2.8. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W48.2.9. Width Number

The width of the component. It is set in pixels.

W48.2.10. Height Number

The height of the component. It is set in pixels.

W48.2.11. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W48.2.12. Tab title EXPRESSION (string)**

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

General**W48.2.13. Name String**

If an expression is used for the `Text` property that cannot be calculated during editing, then the text displayed in the editor can be set here. Also, this text will be displayed in the *Widgets Structure* panel.

Indentation

W48.2.14. First line Number

First line indentation.

W48.2.15. Hanging Number

Indentation of other lines.

Style

W48.2.16. Default Object

Style that will be used to render the Widget.

Events

W48.2.17. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W48.2.18. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W48.2.19. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W48.2.20. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that out-

put. The data that will be passed through that output is the textual description of the error.

W48.3. Examples

- *eez-gui-widgets-demo*

W49. NumberInput



W49.1. Description

This Widget is used when we want to enter a number.

W49.2. Properties

Specific

W49.2.1. Value *EXPRESSION (double)*

The variable in which the entered number will be stored.

W49.2.2. Min *EXPRESSION (double)*

The minimum value that can be entered.

W49.2.3. Max *EXPRESSION (double)*

The maximum value that can be entered.

W49.2.4. Step *EXPRESSION (double)*

The step is a number that specifies the granularity that the value must adhere to.

W49.2.5. Disable default tab handling *Boolean*

Set this to false if you want to disable default TAB key handling when this widget is in focus.

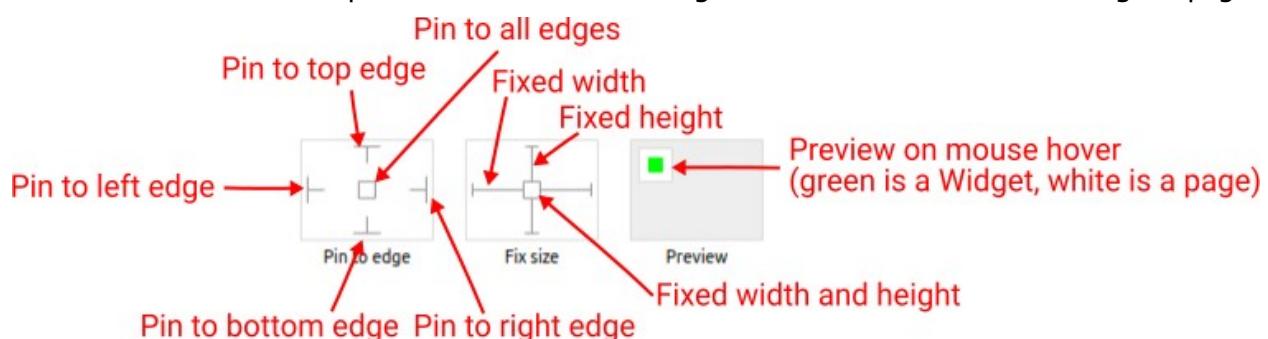
W49.2.6. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W49.2.7. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

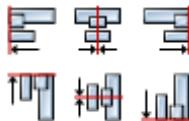
Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

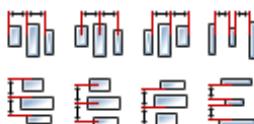
W49.2.8. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE



W49.2.9. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W49.2.10. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W49.2.11. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W49.2.12. Width Number

The width of the component. It is set in pixels.

W49.2.13. Height Number

The height of the component. It is set in pixels.

W49.2.14. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W49.2.15. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W49.2.16. Default** *Object*

Style used when rendering of the Widget.

Events**W49.2.17. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W49.2.18. Output widget handle** *Boolean*

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W49.2.19. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W49.2.20. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W49.2.21. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W50. Panel



W50.1. Description

It is used to group several Widgets, and it is used when we want to additionally organize a page that contains a large number of Widgets or if we want to perform some operation on several Widgets at once, e.g. hide using the `Hidden` flag of the Panel. When the Widget is inside the Panel, then its left and top coordinates are relative to the left and top of the Panel, which means that when the Panel is moved, all the Widgets inside it are also moved. Widgets are added to the Panel via the *Widgets Structure* panel using drag and drop.

W50.2. Properties

General

W50.2.1. Name *String*

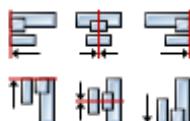
Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

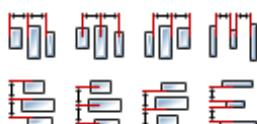
W50.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN

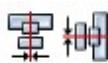


DISTRIBUTE



W50.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W50.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W50.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W50.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W50.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W50.2.8. Width *Number*

The width of the component. It is set in pixels.

W50.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W50.2.10. Height *Number*

The height of the component. It is set in pixels.

W50.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W50.2.12. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W50.2.13. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W50.2.14. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W50.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W50.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W50.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W50.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W50.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W50.2.20. Scrollable *Boolean*

Make the object scrollable.

W50.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W50.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W50.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W50.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W50.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W50.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W50.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W50.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W50.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W50.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W50.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W50.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W50.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W50.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W50.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W50.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W50.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States

W50.2.38. Checked EXPRESSION (boolean)

Toggled or checked state.

W50.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W50.2.40. Disabled EXPRESSION (boolean)

Disabled state

W50.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W50.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W50.2.43. Pressed Boolean

Being pressed.

Events**W50.2.44. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W50.2.45. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W50.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W50.2.47. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W51. Plotly



W51.1. Description

A widget used to display Plotly charts by specifying chart data, layout and configuration options through JSON values.

W51.2. Properties

Specific

W51.2.1. Chart data *EXPRESSION (json)*

Check Plotly documentation ([link](#)) for more informations.

W51.2.2. Layout options *EXPRESSION (json)*

Check Plotly documentation ([link](#)) for more informations.

W51.2.3. Configuration options *EXPRESSION (json)*

Check Plotly documentation ([link](#)) for more informations.

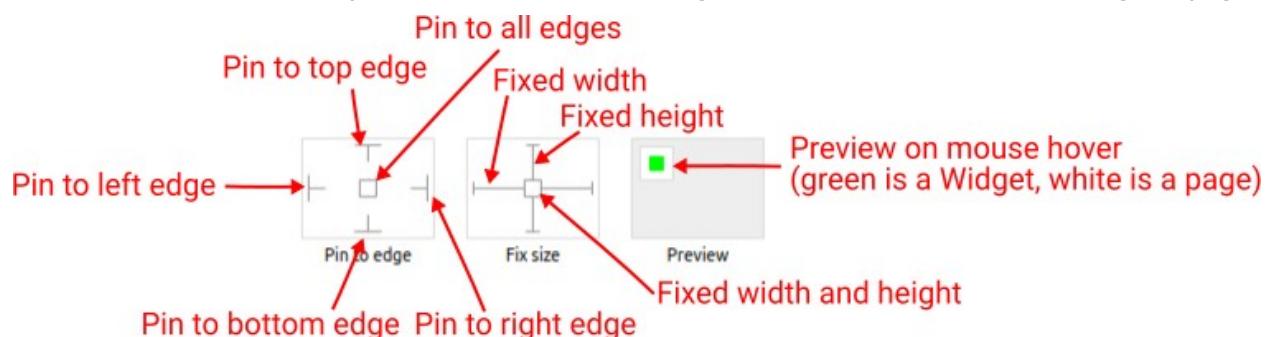
W51.2.4. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W51.2.5. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



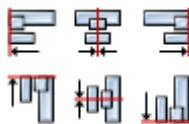
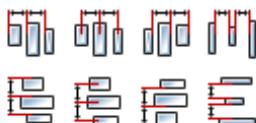
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

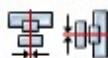
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W51.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W51.2.7. Center widget** *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W51.2.8. Left** *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W51.2.9. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W51.2.10. Width *Number*

The width of the component. It is set in pixels.

W51.2.11. Height *Number*

The height of the component. It is set in pixels.

W51.2.12. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W51.2.13. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Events

W51.2.14. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W51.2.15. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W51.2.16. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W51.2.17. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W51.2.18. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W51.3. Inputs

W51.4. Outputs

W51.5. Examples

- *Plotly Examples*

W52. Progress (Dashboard)



W52.1. Description

We can use this Widget, for example, when we want to display the execution progress of an operation.

W52.2. Properties

Specific

W52.2.1. Data *EXPRESSION (integer)*

A value that goes from Min (progress is at 0%) to Max (progress is at 100%).

W52.2.2. Min *EXPRESSION (any)*

The minimum value that **Data** can contain.

W52.2.3. Max *EXPRESSION (any)*

The maximum value that **Data** can contain.

W52.2.4. Orientation *Enum*

There are two orientations that the **Progress** widget can have:

- Horizontal
- Vertical

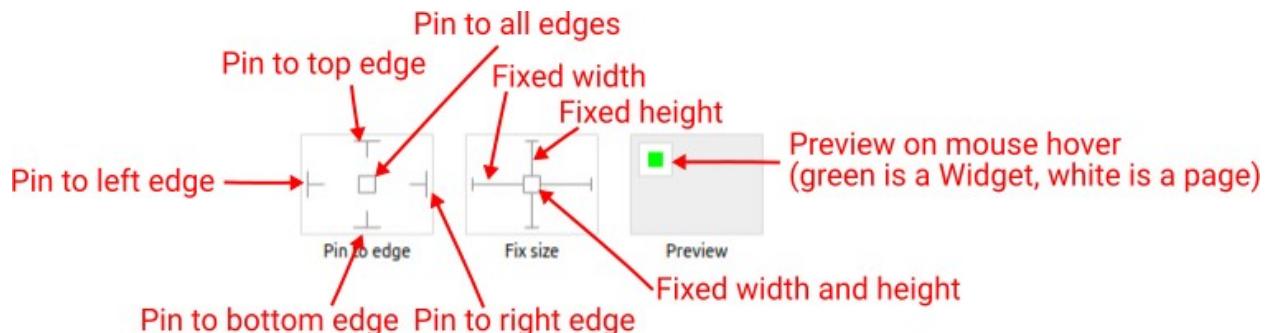
W52.2.5. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W52.2.6. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

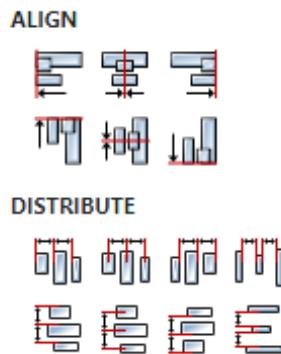
Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the

width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W52.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W52.2.8. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W52.2.9. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the *Top*, *Width* and *Height* properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use *+*, *-*, *** and */* operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2`.

W52.2.10. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W52.2.11. Width Number

The width of the component. It is set in pixels.

W52.2.12. Height Number

The height of the component. It is set in pixels.

W52.2.13. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W52.2.14. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W52.2.15. Default Object

Style used when rendering of the Widget.

Events

W52.2.16. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W52.2.17. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W52.2.18. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W52.2.19. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W52.2.20. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W52.3. Examples

- *Dashboard Widgets Demo*



W53.1. Description

We can use this Widget, for example, when we want to display the execution progress of an operation.

W53.2. Properties

Specific

W53.2.1. Data *EXPRESSION (integer)*

A value that goes from Min (progress is at 0%) to Max (progress is at 100%).

W53.2.2. Min *EXPRESSION (any)*

The minimum value that `Data` can contain.

W53.2.3. Max *EXPRESSION (any)*

The maximum value that `Data` can contain.

W53.2.4. Orientation *Enum*

There are two orientations that the `Progress` widget can have:

- Horizontal



- Vertical



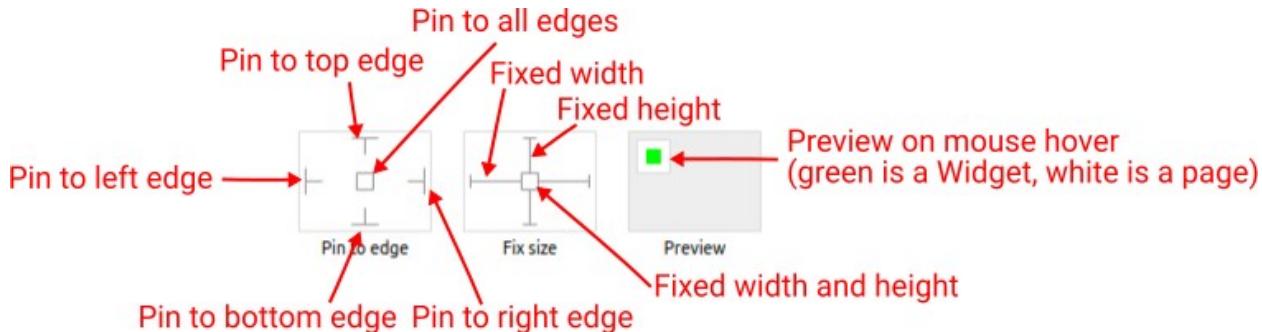
W53.2.5. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W53.2.6. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

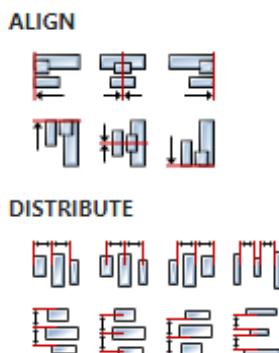
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W53.2.7. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W53.2.8. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W53.2.9. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W53.2.10. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W53.2.11. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W53.2.12. Width *Number*

The width of the component. It is set in pixels.

W53.2.13. Height *Number*

The height of the component. It is set in pixels.

W53.2.14. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W53.2.15. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W53.2.16. Default** *Object*

Style used when rendering of the Widget.

Events**W53.2.17. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W53.2.18. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W53.2.19. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

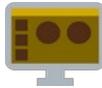
W53.2.20. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W53.3. Examples

- *eez-gui-widgets-demo*

W54. QRCode (Dashboard)



W54.1. Description

Renders a QR Code representing the given text string at the given error correction level.

W54.2. Properties

Specific

W54.2.1. Text *EXPRESSION (any)*

Text for which the QR Code will be generated.

W54.2.2. Error correction *Enum*

The error correction level in a QR Code symbol, possible options:

- Low – The QR Code can tolerate about 7% erroneous codewords.
- Medium – The QR Code can tolerate about 15% erroneous codewords.
- Quartile – The QR Code can tolerate about 25% erroneous codewords.
- High – The QR Code can tolerate about 30% erroneous codewords.

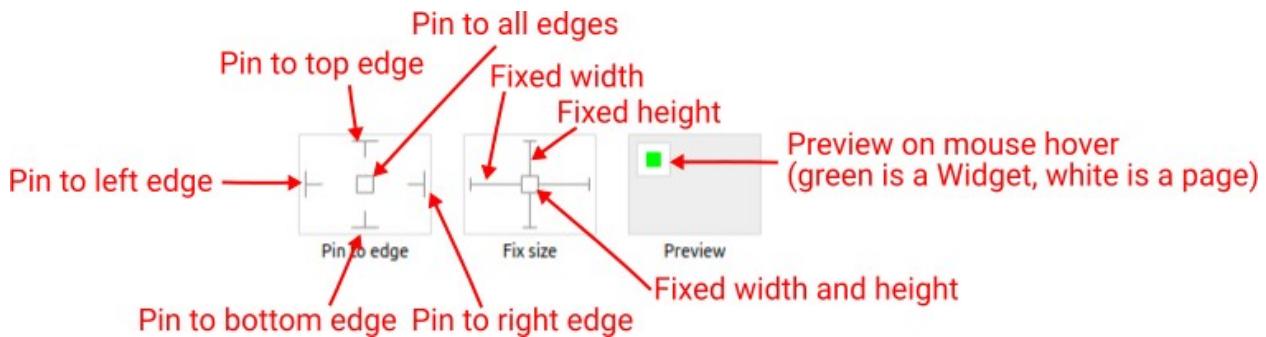
W54.2.3. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W54.2.4. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



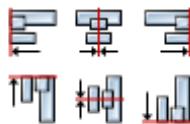
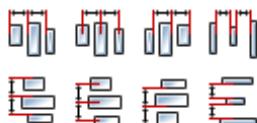
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W54.2.5. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W54.2.6. Center widget** *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W54.2.7. Left** *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W54.2.8. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W54.2.9. Width *Number*

The width of the component. It is set in pixels.

W54.2.10. Height *Number*

The height of the component. It is set in pixels.

W54.2.11. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W54.2.12. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W54.2.13. Default Object

Style used when rendering of the Widget.

Events**W54.2.14. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W54.2.15. Output widget handle Boolean**

If enabled then a new output named @Widget will be added. In runtime, upon a widget creation, a value of type widget will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is AddToInstrumentHistory action component when Plotly is selected for the Item type property. Then it is necessary to set the property Plotly widget to the reference to LineChart widget.

W54.2.16. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W54.2.17. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W54.2.18. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W54.3. Examples

- Dashboard Widgets Demo



W55.1. Description

Renders a QR Code representing the given text string at the given error correction level.

W55.2. Properties

Specific

W55.2.1. Text *EXPRESSION (any)*

Text for which the QR Code will be generated.

W55.2.2. Error correction *Enum*

The error correction level in a QR Code symbol, possible options:

- Low – The QR Code can tolerate about 7% erroneous codewords.
- Medium – The QR Code can tolerate about 15% erroneous codewords.
- Quartile – The QR Code can tolerate about 25% erroneous codewords.
- High – The QR Code can tolerate about 30% erroneous codewords.

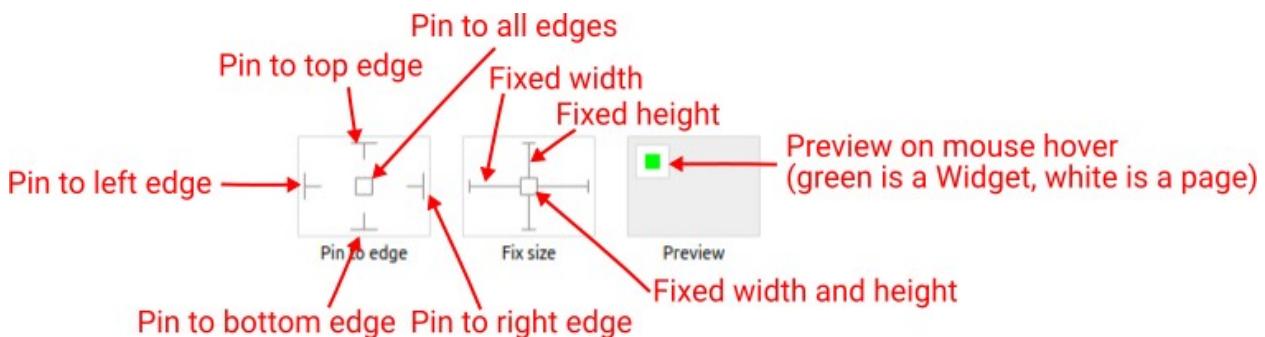
W55.2.3. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W55.2.4. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

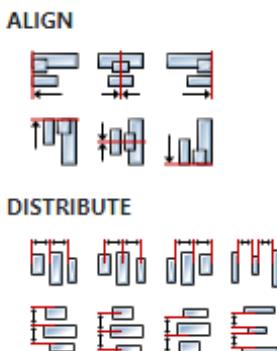
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W55.2.5. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

W55.2.6. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W55.2.7. Center widget** *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W55.2.8. Left** *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W55.2.9. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W55.2.10. Width *Number*

The width of the component. It is set in pixels.

W55.2.11. Height *Number*

The height of the component. It is set in pixels.

W55.2.12. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W55.2.13. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set

the title of the tab that contains this widget.

Style

W55.2.14. Default Object

Style used when rendering of the Widget.

Events

W55.2.15. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W55.2.16. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W55.2.17. Outputs Array

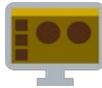
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W55.2.18. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W55.3. Examples

- [eez-gui-widgets-demo](#)



W56.1. Description

Radio Widget is used in radio groups — collections of radio buttons describing a set of related options. Only one radio in a given group can be selected at the same time. In group, each radio has different `Value` property and the same `Group variable` property.

W56.2. Properties

Specific

W56.2.1. Label *EXPRESSION (string)*

Label displayed next to the radio widget.

W56.2.2. Group variable *ASSIGNABLE EXPRESSION (any)*

Variable in which value from `Value` property is stored when this radio widget is selected.

W56.2.3. Value *EXPRESSION (any)*

Value which will be stored in `Group variable` when this radio widget is selected. Also, by comparing this value with the `Group variable` it is decided if this radio widget is selected or not.

W56.2.4. Enabled *EXPRESSION (any)*

If it is true, then the radio is enabled, otherwise it will be disabled.

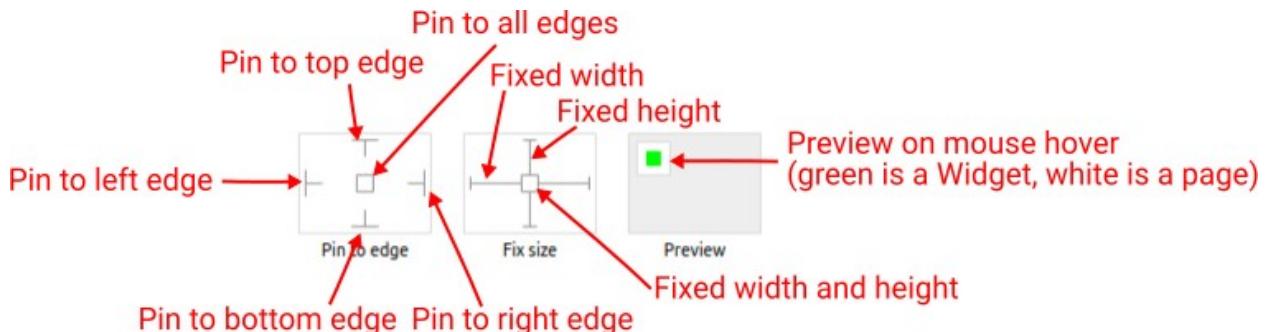
W56.2.5. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W56.2.6. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the `Pin to edge` option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the `Scale to fit` option is selected. E.g. if we selected `Pin to top edge` then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If `Pin to top edge` is not selected, then the Top position will scale proportionally as the page height scales.

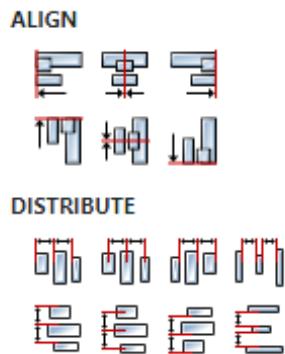
Using the `Fix size` option, we can fix the width/height of the Widget, i.e. if this option is selected the

width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W56.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W56.2.8. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W56.2.9. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the *Top*, *Width* and *Height* properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use *+*, *-*, *** and */* operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2`.

W56.2.10. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W56.2.11. Width Number

The width of the component. It is set in pixels.

W56.2.12. Height Number

The height of the component. It is set in pixels.

W56.2.13. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W56.2.14. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W56.2.15. Default Object

Style used when rendering of the Widget.

Events

W56.2.16. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W56.2.17. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W56.2.18. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W56.2.19. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

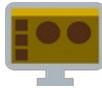
W56.2.20. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W56.3. Examples

- *Dashboard Widgets Demo*

W57. Rectangle (Dashboard)



W57.1. Description

This Widget renders a solid rectangle using the background color from the `Default` style. It can also be used to render horizontal (if height is 1 pixel) and vertical lines (if width is 1 pixel).

W57.2. Properties

Specific

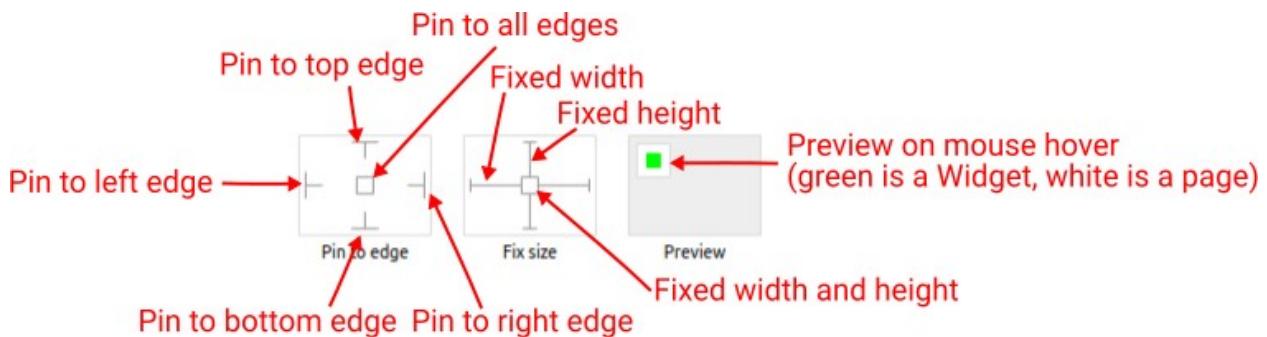
W57.2.1. Visible EXPRESSION (boolean)

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W57.2.2. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



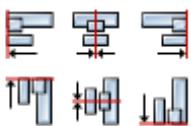
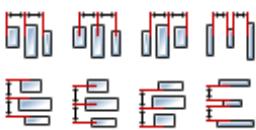
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W57.2.3. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W57.2.4. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W57.2.5. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W57.2.6. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W57.2.7. Width Number

The width of the component. It is set in pixels.

W57.2.8. Height Number

The height of the component. It is set in pixels.

W57.2.9. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W57.2.10. Tab title EXPRESSION (string)**

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W57.2.11. Default Object**

Style used when rendering the background of the Widget.

Events

W57.2.12. Event handlers *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W57.2.13. Output widget handle *Boolean*

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W57.2.14. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W57.2.15. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W57.2.16. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W57.3. Examples

- *Dashboard Widgets Demo*

W58. Rectangle (EEZ-GUI)



W58.1. Description

This Widget renders a solid rectangle using the background color from the `Default` style. It can also be used to render horizontal (if height is 1 pixel) and vertical lines (if width is 1 pixel).

W58.2. Properties

Specific

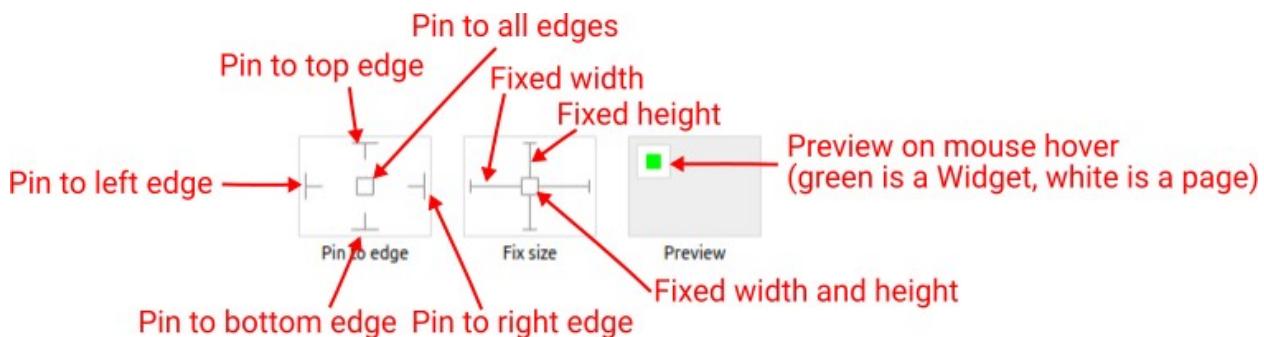
W58.2.1. Visible EXPRESSION (boolean)

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W58.2.2. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

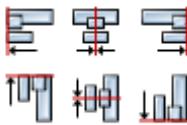
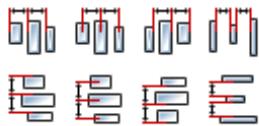
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W58.2.3. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W58.2.4. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W58.2.5. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W58.2.6. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W58.2.7. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W58.2.8. Width Number

The width of the component. It is set in pixels.

W58.2.9. Height Number

The height of the component. It is set in pixels.

W58.2.10. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W58.2.11. Tab title EXPRESSION (string)**

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W58.2.12. Default Object**

Style used when rendering the background of the Widget.

Events

W58.2.13. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W58.2.14. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W58.2.15. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W58.2.16. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W59. Roller (EEZ-GUI)



W59.1. Description

This Widget allows us to select one option from a list using touch based scrolling.

W59.2. Properties

Specific

W59.2.1. Data *EXPRESSION (integer)*

The variable in which the selected value in the range of [Min, Max] is saved.

W59.2.2. Min *EXPRESSION (any)*

The minimum value that can be selected.

W59.2.3. Max *EXPRESSION (any)*

The maximum value that can be selected.

W59.2.4. Text *EXPRESSION (any)*

The text that is displayed in the widget for each possible value that is selected.

Example: set Data to selected_option (of type integer), set Min to 0, and Max to Array.length(TEXTS) - 1, where TEXTS is a variable of type array:string with Default value set to: ["Option 1", "Option 2", "Option 3", ...] and then we can set this property to TEXTS[selected_option].

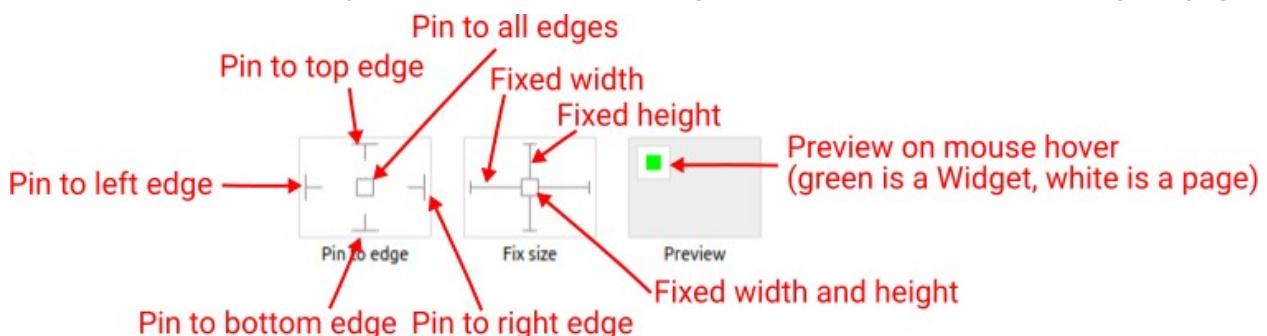
W59.2.5. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W59.2.6. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

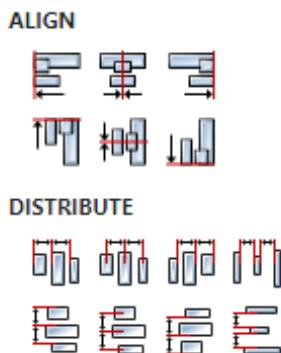
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W59.2.7. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W59.2.8. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W59.2.9. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W59.2.10. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W59.2.11. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W59.2.12. Width Number

The width of the component. It is set in pixels.

W59.2.13. Height Number

The height of the component. It is set in pixels.

W59.2.14. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W59.2.15. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W59.2.16. Default *Object*

Style used when rendering the background of the Widget.

W59.2.17. Selected value *Object*

Style used to render selected value.

W59.2.18. Unselected value *Object*

Style used to render other (unselected) values.

Events

W59.2.19. Event handlers *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W59.2.20. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W59.2.21. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W59.2.22. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error

occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W59.3. Inputs

W59.3.1. clear SEQ / OPTIONAL

We need to send a signal to this input if we want to reset the selection, i.e. choose the first option.

W59.4. Examples

- *eez-gui-widgets-demo*

W60. Roller (LVGL)



W60.1. Description

This Widget allows us to select one option from a list using touch based scrolling.
More info ([link](#))

W60.2. Properties

Specific

W60.2.1. Options *EXPRESSION (array:string)*

List of options.

W60.2.2. Options type *Enum*

Select between Literal and Expression. If Literal is selected then Options are entered one option per line. If Expression is selected then options are evaluated from Options expression which must be of type array:string.

W60.2.3. Selected *EXPRESSION (integer)*

The zero-based index of the selected option.

W60.2.4. Selected type *Enum*

Select between Literal and Assignable. If Assignable is selected then Options can be variable in which the zero-based index of the selected option will be stored.

W60.2.5. Mode *Enum*

Roller mode options:

- NORMAL – normal roller.
- INFINITE – makes the roller circular.

General

W60.2.6. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W60.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

W60.2.15. Height *Number*

The height of the component. It is set in pixels.

W60.2.16. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W60.2.17. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W60.2.18. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W60.2.19. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W60.2.20. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W60.2.21. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W60.2.22. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W60.2.23. Click focusable *Boolean*

Add focused state to the object when clicked.

W60.2.24. Checkable *Boolean*

Toggle checked state when the object is clicked.

W60.2.25. Scrollable *Boolean*

Make the object scrollable.

W60.2.26. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W60.2.27. Scroll momentum Boolean

Make the object scroll further when "thrown".

W60.2.28. Scroll one Boolean

Allow scrolling only one snappable children.

W60.2.29. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W60.2.30. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W60.2.31. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W60.2.32. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W60.2.33. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W60.2.34. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W60.2.35. Event bubble Boolean

Propagate the events to the parent too.

W60.2.36. Gesture bubble Boolean

Propagate the gestures to the parent.

W60.2.37. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W60.2.38. Ignore layout Boolean

Make the object positionable by the layouts.

W60.2.39. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W60.2.40. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W60.2.41. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W60.2.42. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W60.2.43. Checked** *EXPRESSION (boolean)*

Toggled or checked state.

W60.2.44. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W60.2.45. Disabled *EXPRESSION (boolean)*

Disabled state

W60.2.46. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W60.2.47. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W60.2.48. Pressed *Boolean*

Being pressed.

Events**W60.2.49. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W60.2.50. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W60.2.51. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W60.2.52. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W60.3. Examples

- *LVGL Widgets Demo*

W61. Scale



W61.1. Description

Scale allows you to have a linear scale with ranges and sections with custom styling.
More info ([link](#))

W61.2. Properties

Specific

W61.2.1. Scale mode *Enum*

Defines position and orientation of the scale.

W61.2.2. Minor range *Number*

Set the smallest tick value.

W61.2.3. Major range *Number*

Set the largest tick value.

W61.2.4. Total tick count *Number*

Set the number of total ticks.

W61.2.5. Major tick every *Number*

Configure the major tick being every Nth ticks.

W61.2.6. Show labels *Boolean*

Set to true if labels should be drawn.

General

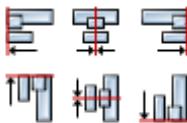
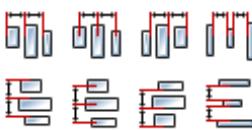
W61.2.7. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W61.2.8. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W61.2.9. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

W61.2.10. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W61.2.11. Left unit Enum

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W61.2.12. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W61.2.13. Top unit Enum

The following options are available:

- px – Top is set in pixels.
- % – The top is set as a percentage in relation to the parent height.

W61.2.14. Width Number

The width of the component. It is set in pixels.

W61.2.15. Width unit Enum

The following options are available:

- px – Width is given in pixels.
- % – Width is given as a percentage in relation to the parent width.
- content – Width is automatically set to fit the entire content in width.

W61.2.16. Height *Number*

The height of the component. It is set in pixels.

W61.2.17. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W61.2.18. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W61.2.19. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W61.2.20. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W61.2.21. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W61.2.22. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W61.2.23. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W61.2.24. Click focusable *Boolean*

Add focused state to the object when clicked.

W61.2.25. Checkable *Boolean*

Toggle checked state when the object is clicked.

W61.2.26. Scrollable *Boolean*

Make the object scrollable.

W61.2.27. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W61.2.28. Scroll momentum Boolean

Make the object scroll further when "thrown".

W61.2.29. Scroll one Boolean

Allow scrolling only one snappable children.

W61.2.30. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W61.2.31. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W61.2.32. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W61.2.33. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W61.2.34. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W61.2.35. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W61.2.36. Event bubble Boolean

Propagate the events to the parent too.

W61.2.37. Gesture bubble Boolean

Propagate the gestures to the parent.

W61.2.38. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W61.2.39. Ignore layout Boolean

Make the object positionable by the layouts.

W61.2.40. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W61.2.41. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W61.2.42. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W61.2.43. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W61.2.44. Checked** *EXPRESSION (boolean)*

Toggled or checked state.

W61.2.45. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W61.2.46. Disabled *EXPRESSION (boolean)*

Disabled state

W61.2.47. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W61.2.48. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W61.2.49. Pressed *Boolean*

Being pressed.

Events**W61.2.50. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W61.2.51. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W61.2.52. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W61.2.53. Catch error Boolean

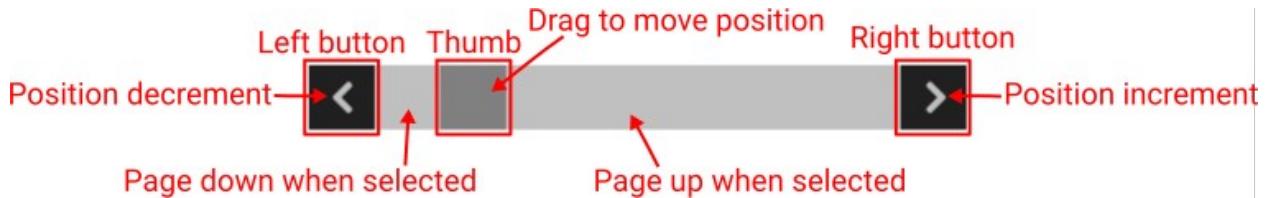
If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W62. ScrollBar



W62.1. Description

This Widget can be used with `List` and `Grid` Widget for scrolling within large lists that do not fit entirely within said Widgets. If `width > height` then a horizontal `ScrollBar` is displayed:



... and if `width <= height` then a vertical `ScrollBar` is displayed.



The horizontal `ScrollBar` has left and right buttons, and the vertical top and bottom buttons.

This Widget connects to the `List` or `Grid` Widget via a variable of type `struct:$ScrollbarState` which is set in the `Data` property. The structure `struct:$ScrollbarState` has these fields:

- `numItems` – how many items/elements are in the list
- `itemsPerPage` – how many items fit inside the `List` or `Grid` Widget.
- `positionIncrement` – determines how many items we will move within the list when the left/top button (shift to the left/up) or the right/bottom button (shift to the right/down) is selected.
- `position` – the position of the first item/element that is rendered in the list. So within the `List` or `Grid` Widget, items from `position` to `position + itemsPerPage` will be rendered. `position` can be in the interval from `0` to `numItems - itemsPerPage`.

The scrollbar can change its 'position' in the following ways:

- By selecting the Left/Top button `position` is decreased by the `positionIncrement` value.
- By selecting Right/Bottom button `position` is increased by `positionIncrement` value.
- By moving the thumb `position` is set to a value in the interval from `0` to `numItems - itemsPerPage`.
- If the region between the Left/Top button and the thumb is selected, then the position is

- reduced by `itemsPerPage` (AKA "page up").
- If the region between the thumb and the Right/Bottom button is selected, then the position is increased by `itemsPerPage` (AKA "page down").

W62.2. Properties

Specific

W62.2.1. Data *EXPRESSION (struct:\$ScrollbarState)*

Set here the name of the `struct:$ScrollbarState` type variable.

W62.2.2. Left button text *String*

The text that will be displayed inside the left/top button. Usually a single character from an icons font is used.

W62.2.3. Right button text *String*

The text that will be displayed inside the right/bottom button. Usually a single character from an icons font is used.

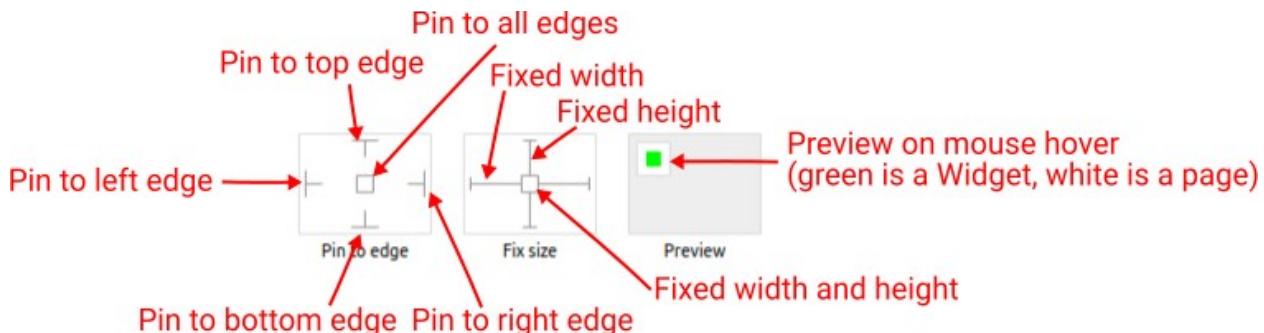
W62.2.4. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W62.2.5. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

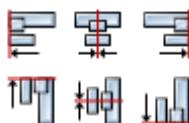
W62.2.6. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

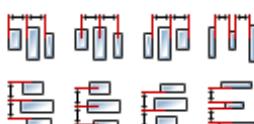
W62.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE

**W62.2.8. Center widget** Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W62.2.9. Left** Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W62.2.10. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W62.2.11. Width Number

The width of the component. It is set in pixels.

W62.2.12. Height Number

The height of the component. It is set in pixels.

W62.2.13. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W62.2.14. Tab title** EXPRESSION (string)

If this widget is a child of a container with layout set to Docking Manager, use this property to set

the title of the tab that contains this widget.

Style

W62.2.15. Default Object

Style used when rendering the background of the Widget.

W62.2.16. Thumb Object

Style that will be used to render the scrollbar thumb.

W62.2.17. Buttons Object

Style used to render the left and right buttons.

Events

W62.2.18. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W62.2.19. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W62.2.20. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W62.2.21. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W62.3. Examples

- *eez-gui-widgets-demo*

W63. Select



W63.1. Description

This Widget, similar to `Container`, has multiple Child widgets under it. But unlike `Container`, which will always display all Child widgets, this Widget displays only one Child widget, and that is the one we selected via the `Data` property. Therefore, use this Widget when you want depending on e.g. the value of some variable to change the structure of the page. Widgets are added to `Select` via the *Widgets Structure* panel using drag and drop.

W63.2. Properties

Specific

W63.2.1. Data EXPRESSION (boolean)

The result of the evaluation of this expression must be the zero based index of the Widget that is to be displayed. So if the result is 0 then the first Widget will be displayed, if the result is 1 then the second Widget will be displayed, etc. The order of Widgets can be selected using drag and drop within the *Widgets Structure* panel.

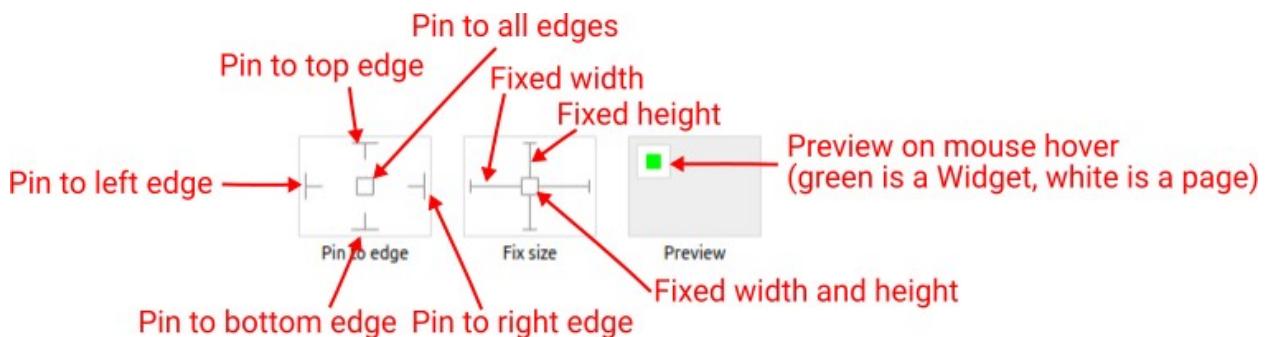
W63.2.2. Visible EXPRESSION (boolean)

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W63.2.3. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

the title of the tab that contains this widget.

Style

W63.2.13. Default Object

Style used when rendering the background of the Widget.

Events

W63.2.14. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W63.2.15. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W63.2.16. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

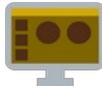
W63.2.17. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W63.3. Examples

- [eez-gui-widgets-demo](#)

W64. Slider (Dashboard)



W64.1. Description

This Widget allows us to select one value from the list by moving the knob on the slider.

W64.2. Properties

Specific

W64.2.1. Value *EXPRESSION (double)*

The variable in which the selected value in the range of [Min, Max] is saved.

W64.2.2. Min *EXPRESSION (double)*

The minimum value that can be selected.

W64.2.3. Max *EXPRESSION (double)*

The maximum value that can be selected.

W64.2.4. Step *EXPRESSION (double)*

The step is a number that specifies the granularity that the value must adhere to.

W64.2.5. View min *EXPRESSION (double)*

The minimum value that is displayed.

W64.2.6. View max *EXPRESSION (double)*

The maximum value that is displayed.

W64.2.7. Enabled *EXPRESSION (any)*

If it is true, then the slider is enabled, otherwise it will be disabled.

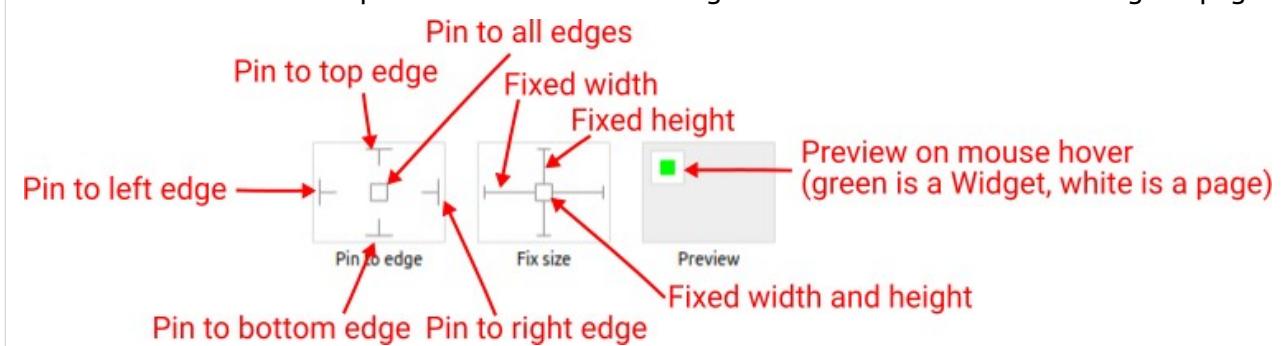
W64.2.8. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W64.2.9. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



Style used when rendering of the Widget.

Events

W64.2.19. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W64.2.20. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W64.2.21. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W64.2.22. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W64.2.23. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W64.3. Examples

- *Dashboard Widgets Demo*

W65. Slider (EEZ-GUI)



W65.1. Description

This Widget allows us to select one value from the list by moving the knob on the slider.

W65.2. Properties

Specific

W65.2.1. Data *EXPRESSION (integer)*

The variable in which the selected value in the range of [Min, Max] is saved.

W65.2.2. Min *EXPRESSION (any)*

The minimum value that can be selected.

W65.2.3. Max *EXPRESSION (any)*

The maximum value that can be selected.

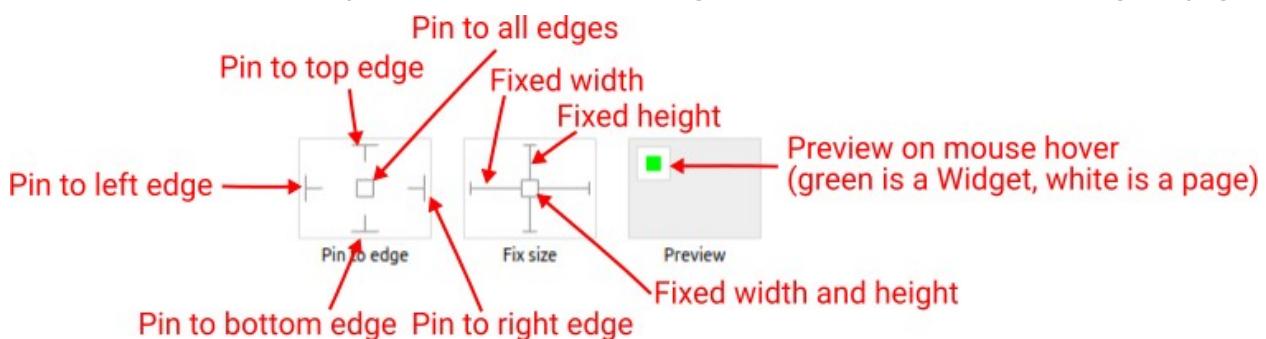
W65.2.4. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W65.2.5. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

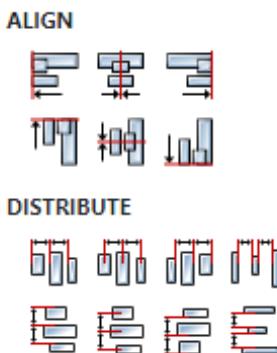
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W65.2.6. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

W65.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W65.2.8. Center widget** *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W65.2.9. Left** *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W65.2.10. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W65.2.11. Width *Number*

The width of the component. It is set in pixels.

W65.2.12. Height *Number*

The height of the component. It is set in pixels.

W65.2.13. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W65.2.14. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set

the title of the tab that contains this widget.

Style

W65.2.15. Default Object

Style used when rendering of the Widget.

Events

W65.2.16. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W65.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W65.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W65.2.19. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W65.3. Examples

- [eez-gui-widgets-demo](#)

W66. Slider (LVGL)



W66.1. Description

This Widget allows us to select one or two values from the list by moving the knob on the slider.
More info ([link](#))

W66.2. Properties

Specific

W66.2.1. Min *Number*

The minimum value that can be selected.

W66.2.2. Max *Number*

The maximum value that can be selected.

W66.2.3. Mode *Enum*

Slider mode options:

- `NORMAL` – A normal slider.
- `SYMMETRICAL` – Draw the indicator form the zero value to current value. Requires negative minimum range and positive maximum range.
- `RANGE` – Allows setting the start value (`Value left` property) and end value (`Value` property).

W66.2.4. Value *EXPRESSION (integer)*

The selected value on the slider. If `RANGE` mode is selected then this is selected end value on the slider.

W66.2.5. Value type *Enum*

Select between `Literal` and `Assignable`. If `Assignable` is selected then `Value` can be variable in which the selected value will be stored.

W66.2.6. Value left *EXPRESSION (integer)*

If `RANGE` mode is selected then this is selected start value on the slider.

W66.2.7. Value left type *Enum*

Select between `Literal` and `Assignable`. If `Assignable` is selected then `Value left` can be variable in which the selected start value will be stored.

W66.2.8. Enable animation *Boolean*

If enabled then value change will be animated. Duration of animation is controlled with the `style` property ("Miscellaneous" section) "Anim time" in LVGL 8.4 or "Anim duration" in LVGL 9.1.

General

W66.2.9. Name *String*

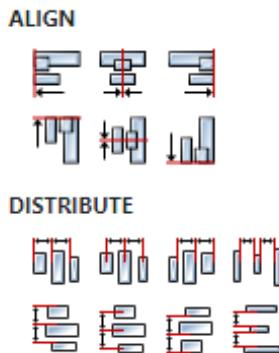
Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional

and does not need to be set if we do not need to reference the Widget.

Position and size

W66.2.10. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W66.2.11. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W66.2.12. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W66.2.13. Left unit Enum

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W66.2.14. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W66.2.15. Top unit Enum

The following options are available:

- px – Top is set in pixels.
- % – The top is set as a percentage in relation to the parent height.

W66.2.16. Width Number

The width of the component. It is set in pixels.

W66.2.17. Width unit Enum

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W66.2.18. Height Number

The height of the component. It is set in pixels.

W66.2.19. Height unit Enum

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W66.2.20. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W66.2.21. Use style ObjectReference

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W66.2.22. Hidden EXPRESSION (boolean)

Make the object hidden.

W66.2.23. Hidden flag type Enum

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W66.2.24. Clickable EXPRESSION (boolean)

Make the object clickable by input devices.

W66.2.25. Clickable flag type Enum

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W66.2.26. Click focusable Boolean

Add focused state to the object when clicked.

W66.2.27. Checkable Boolean

Toggle checked state when the object is clicked.

W66.2.44. Scrollbar mode *Enum*

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W66.2.45. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W66.2.46. Checked** *EXPRESSION (boolean)*

Toggled or checked state.

W66.2.47. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W66.2.48. Disabled *EXPRESSION (boolean)*

Disabled state

W66.2.49. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W66.2.50. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W66.2.51. Pressed *Boolean*

Being pressed.

Events**W66.2.52. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.

- Action - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W66.2.53. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W66.2.54. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W66.2.55. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W66.3. Examples

- Dashboard Widgets Demo

W67. Span



W67.1. Description

This widget is work in progress, it means that you can add it to your project and Studio will generate all the code for its creation, but for anything more than that you should do it in your custom code, for example after `ui_init()` has been called.

More info ([link](#))

W67.2. Properties

General

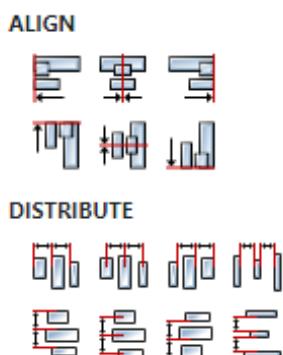
W67.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W67.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W67.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W67.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W67.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W67.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W67.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W67.2.8. Width *Number*

The width of the component. It is set in pixels.

W67.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W67.2.10. Height *Number*

The height of the component. It is set in pixels.

W67.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W67.2.12. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W67.2.13. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W67.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W67.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W67.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W67.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W67.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W67.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W67.2.20. Scrollable *Boolean*

Make the object scrollable.

W67.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W67.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W67.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W67.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W67.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W67.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W67.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W67.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W67.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W67.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W67.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W67.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W67.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W67.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W67.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W67.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W67.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W67.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W67.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W67.2.40. Disabled EXPRESSION (boolean)

Disabled state

W67.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W67.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W67.2.43. Pressed Boolean

Being pressed.

Events**W67.2.44. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W67.2.45. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W67.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W67.2.47. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W68. Spinbox



W68.1. Description

This widget is work in progress, it means that you can add it to your project and Studio will generate all the code for its creation, but for anything more than that you should do it in your custom code, for example after `ui_init()` has been called.

More info ([link](#))

W68.2. Properties

General

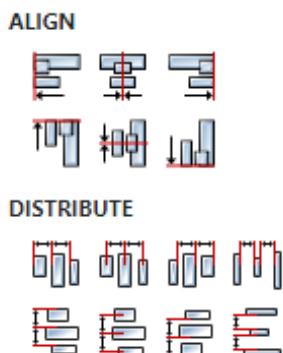
W68.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W68.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W68.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W68.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W68.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W68.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W68.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W68.2.8. Width *Number*

The width of the component. It is set in pixels.

W68.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W68.2.10. Height *Number*

The height of the component. It is set in pixels.

W68.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W68.2.12. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W68.2.13. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W68.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W68.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W68.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W68.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W68.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W68.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W68.2.20. Scrollable *Boolean*

Make the object scrollable.

W68.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W68.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W68.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W68.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W68.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W68.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W68.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W68.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W68.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W68.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W68.2.43. Pressed Boolean

Being pressed.

Events**W68.2.44. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W68.2.45. Inputs Array**

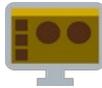
Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W68.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W68.2.47. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.



W69.1. Description

We use this Widget to show that some operation is in progress or something is loading, etc.

W69.2. Properties

Specific

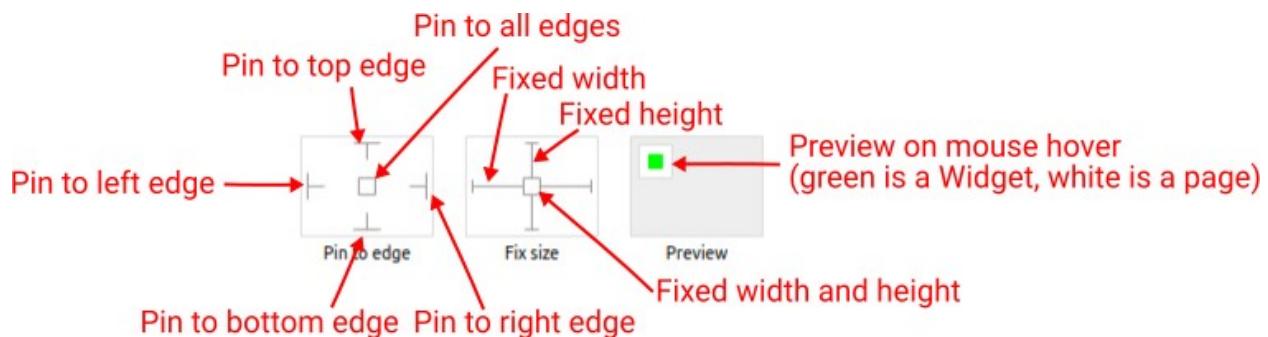
W69.2.1. Visible EXPRESSION (boolean)

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W69.2.2. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



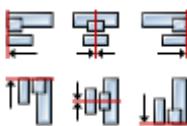
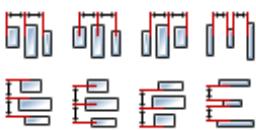
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W69.2.3. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W69.2.4. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W69.2.5. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W69.2.6. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W69.2.7. Width Number

The width of the component. It is set in pixels.

W69.2.8. Height Number

The height of the component. It is set in pixels.

W69.2.9. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W69.2.10. Tab title EXPRESSION (string)**

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W69.2.11. Default Object**

Style used when rendering of the Widget.

Events

W69.2.12. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W69.2.13. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W69.2.14. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W69.2.15. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W69.2.16. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W69.3. Examples

- *Dashboard Widgets Demo*

W70. Spinner (LVGL)



W70.1. Description

Use this Widget to show that some operation is in progress or something is loading, etc.

W70.2. Properties

General

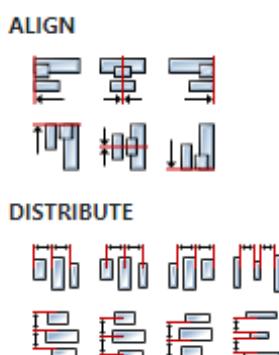
W70.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W70.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W70.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W70.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W70.2.5. Left unit *Enum*

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W70.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W70.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W70.2.8. Width *Number*

The width of the component. It is set in pixels.

W70.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W70.2.10. Height *Number*

The height of the component. It is set in pixels.

W70.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W70.2.12. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W70.2.13. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W70.2.14. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W70.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W70.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W70.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W70.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W70.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W70.2.20. Scrollable *Boolean*

Make the object scrollable.

W70.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W70.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W70.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W70.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W70.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W70.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W70.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W70.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W70.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W70.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W70.2.31. Gesture bubble *Boolean*

Propagate the gestures to the parent.

W70.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W70.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W70.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W70.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W70.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W70.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W70.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W70.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W70.2.40. Disabled EXPRESSION (boolean)

Disabled state

W70.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W70.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W70.2.43. Pressed Boolean

Being pressed.

Events

W70.2.44. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W70.2.45. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W70.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W70.2.47. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W70.3. Examples

- *Dashboard Widgets Demo*

W71. Switch (Dashboard)



W71.1. Description

Switch Widget is used when we want a turn ON or turn OFF option.

W71.2. Properties

Specific

W71.2.1. Value *EXPRESSION (any)*

Boolean variable in which `true` is stored when the switch is ON and `false` when the switch is OFF.

W71.2.2. Enabled *EXPRESSION (any)*

If it is true, then the switch is enabled, otherwise it will be disabled.

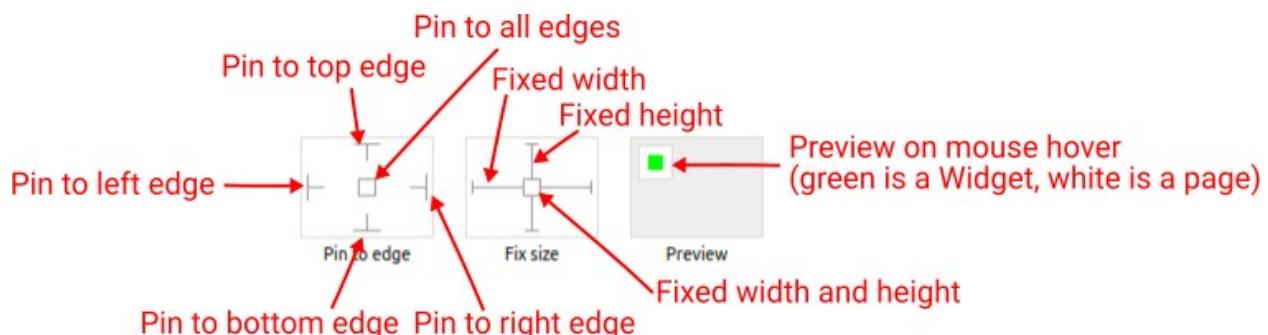
W71.2.3. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W71.2.4. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



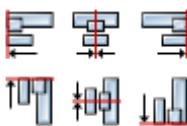
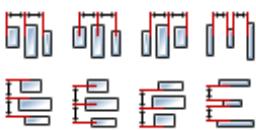
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W71.2.5. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W71.2.6. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W71.2.7. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W71.2.8. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W71.2.9. Width Number

The width of the component. It is set in pixels.

W71.2.10. Height Number

The height of the component. It is set in pixels.

W71.2.11. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W71.2.12. Tab title EXPRESSION (string)**

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W71.2.13. Default Object**

Style used when rendering of the Widget.

Events

W71.2.14. Event handlers *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W71.2.15. Output widget handle *Boolean*

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W71.2.16. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W71.2.17. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W71.2.18. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W71.3. Examples

- *Dashboard Widgets Demo*

W72. Switch (EEZ-GUI)



W72.1. Description

Checkbox Widget is used when we want a turn ON or turn OFF option.

W72.2. Properties

Specific

W72.2.1. Data EXPRESSION (boolean)

Boolean variable in which `true` is stored when the switch is ON and `false` when the switch is OFF.

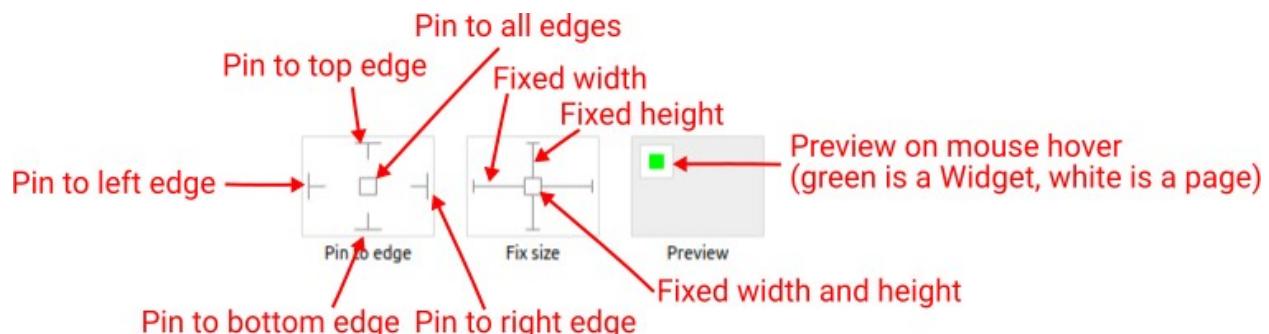
W72.2.2. Visible EXPRESSION (boolean)

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W72.2.3. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

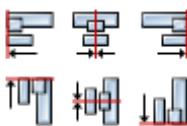
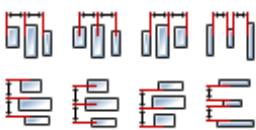
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W72.2.4. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W72.2.5. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W72.2.6. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W72.2.7. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W72.2.8. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W72.2.9. Width Number

The width of the component. It is set in pixels.

W72.2.10. Height Number

The height of the component. It is set in pixels.

W72.2.11. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W72.2.12. Tab title EXPRESSION (string)**

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W72.2.13. Default Object**

Style used when rendering of the Widget.

Events

W72.2.14. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W72.2.15. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W72.2.16. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W72.2.17. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W72.3. Examples

- [eez-gui-widgets-demo](#)

W73. Switch (LVGL)



W73.1. Description

Switch Widget is used when we want a turn ON or turn OFF option.
More info ([link](#))

W73.2. Properties

General

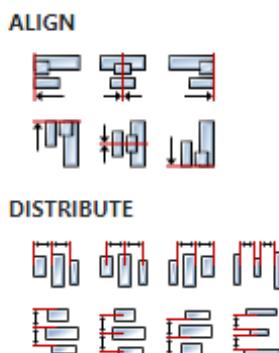
W73.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W73.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W73.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W73.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W73.2.5. Left unit *Enum*

The following options are available:

- px – Left is default in pixels.

- `%` – Left is set as a percentage in relation to the parent width.

W73.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W73.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W73.2.8. Width *Number*

The width of the component. It is set in pixels.

W73.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W73.2.10. Height *Number*

The height of the component. It is set in pixels.

W73.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W73.2.12. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W73.2.13. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W73.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W73.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W73.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W73.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W73.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W73.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W73.2.20. Scrollable *Boolean*

Make the object scrollable.

W73.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W73.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W73.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W73.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W73.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W73.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W73.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W73.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W73.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W73.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W73.2.31. Gesture bubble *Boolean*

Propagate the gestures to the parent.

Events

W73.2.44. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W73.2.45. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W73.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W73.2.47. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W73.3. Examples

- *Dashboard Widgets Demo*

W74. Tab



W74.1. Description

This widget should be used as a child of Tabview widget. See Tabview widget for the more details.

W74.2. Properties

Specific

W74.2.1. Tab name *EXPRESSION (string)*

The name of the tab.

W74.2.2. Tab name type *Enum*

Here we can choose whether the `Name` property will be calculated from the Expression.

General

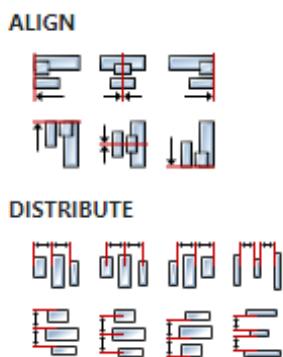
W74.2.3. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W74.2.4. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W74.2.5. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W74.2.6. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), sim-

ple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36, 50 + 32 * 6, (100 - 32) / 2.`

W74.2.7. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W74.2.8. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W74.2.9. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W74.2.10. Width *Number*

The width of the component. It is set in pixels.

W74.2.11. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W74.2.12. Height *Number*

The height of the component. It is set in pixels.

W74.2.13. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W74.2.14. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W74.2.15. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W74.2.16. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W74.2.17. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W74.2.18. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W74.2.19. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W74.2.20. Click focusable *Boolean*

Add focused state to the object when clicked.

W74.2.21. Checkable *Boolean*

Toggle checked state when the object is clicked.

W74.2.22. Scrollable *Boolean*

Make the object scrollable.

W74.2.23. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W74.2.24. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W74.2.25. Scroll one *Boolean*

Allow scrolling only one snappable children.

W74.2.26. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W74.2.27. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W74.2.28. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W74.2.29. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W74.2.30. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W74.2.31. Press lock *Boolean*

W74.2.44. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W74.2.45. Pressed Boolean

Being pressed.

Events**W74.2.46. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W74.2.47. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W74.2.48. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W74.2.49. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W74.3. Examples

- `Tabview`
- `Styled Tabview`

W75. Table



W75.1. Description

This widget is work in progress, it means that you can add it to your project and Studio will generate all the code for its creation, but for anything more than that you should do it in your custom code, for example after `ui_init()` has been called.

More info ([link](#))

W75.2. Properties

General

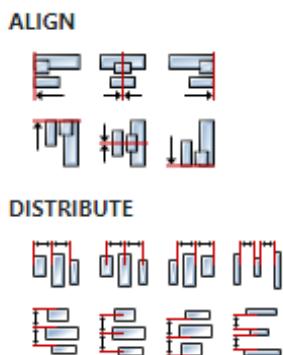
W75.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W75.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W75.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W75.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W75.2.5. Left unit *Enum*

The following options are available:

- `px` – Left is default in pixels.
- `%` – Left is set as a percentage in relation to the parent width.

W75.2.6. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W75.2.7. Top unit *Enum*

The following options are available:

- `px` – Top is set in pixels.
- `%` – The top is set as a percentage in relation to the parent height.

W75.2.8. Width *Number*

The width of the component. It is set in pixels.

W75.2.9. Width unit *Enum*

The following options are available:

- `px` – Width is given in pixels.
- `%` – Width is given as a percentage in relation to the parent width.
- `content` – Width is automatically set to fit the entire content in width.

W75.2.10. Height *Number*

The height of the component. It is set in pixels.

W75.2.11. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout

W75.2.12. Tab title *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W75.2.13. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W75.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W75.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W75.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W75.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W75.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W75.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W75.2.20. Scrollable *Boolean*

Make the object scrollable.

W75.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W75.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W75.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W75.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W75.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W75.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W75.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W75.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W75.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W75.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W75.2.43. Pressed Boolean

Being pressed.

Events**W75.2.44. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W75.2.45. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W75.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W75.2.47. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W76. Tabulator



W76.1. Description

A widget used to display or edit tables. Under the hood, Tabulator ([link](#)) is used.

W76.2. Properties

Specific

W76.2.1. Data *EXPRESSION (json)*

Table data, see Load Data From Array/JSON ([link](#)) for more informations.

W76.2.2. Basic options *Object*

Basic table options. See Tabulator documentation ([link](#)) for more informations.

W76.2.3. Advanced options *EXPRESSION (json)*

Advanced options that overrides basic options and can be specified through JSON value. Since this is the Expression property, you can change your options conditionally in runtime.

W76.2.4. Persistent configuration *EXPRESSION (json)*

Enter variable in which persistent configuration ([link](#)) is stored.

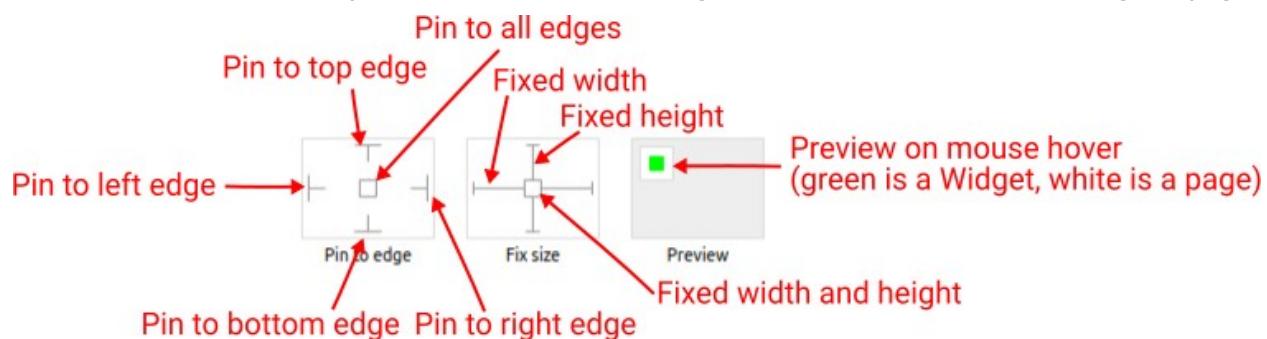
W76.2.5. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W76.2.6. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



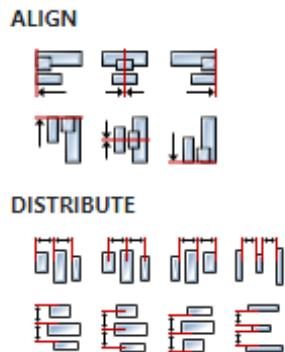
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

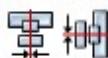
W76.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W76.2.8. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W76.2.9. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W76.2.10. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W76.2.11. Width Number

The width of the component. It is set in pixels.

W76.2.12. Height Number

The height of the component. It is set in pixels.

W76.2.13. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W76.2.14. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set

the title of the tab that contains this widget.

Events

W76.2.15. Event handlers *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W76.2.16. Output widget handle *Boolean*

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W76.2.17. Inputs *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W76.2.18. Outputs *Array*

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W76.2.19. Catch error *Boolean*

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W76.3. Inputs

W76.4. Outputs

W76.5. Examples

W77. Tabview



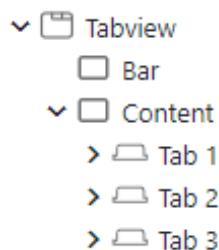
W77.1. Description

The Tab view object can be used to organize content in tabs. Tabview can be configured in two ways:

- Place Tab widgets immediately under TabView:



- Add two container widgets under Tabview. First container is for the tab Bar, second container is for the tab Content. In this configuration, Tab widgets should be placed under second container, Content. Use this configuration if you want to style tab Bar and Content.



To add Tab widget to Tabview, drag and drop directly from the Widgets Palette to the Tabview inside Widgets Structure panel.

More info ([link](#))

W77.2. Properties

Specific

W77.2.1. Position *Enum*

With this property, the tab bar can be moved to any sides.

W77.2.2. Size *Number*

The size of the tab bar. In case of vertical arrangement it means the height of the tab bar, and in horizontal arrangement it means the width.

General

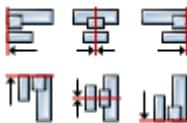
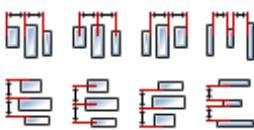
W77.2.3. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W77.2.4. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN**DISTRIBUTE****W77.2.5. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W77.2.6. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W77.2.7. Left unit Enum

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W77.2.8. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W77.2.9. Top unit Enum

The following options are available:

- px – Top is set in pixels.
- % – The top is set as a percentage in relation to the parent height.

W77.2.10. Width Number

The width of the component. It is set in pixels.

W77.2.11. Width unit Enum

The following options are available:

- px – Width is given in pixels.
- % – Width is given as a percentage in relation to the parent width.
- content – Width is automatically set to fit the entire content in width.

W77.2.12. Height *Number*

The height of the component. It is set in pixels.

W77.2.13. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W77.2.14. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W77.2.15. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W77.2.16. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W77.2.17. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W77.2.18. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W77.2.19. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W77.2.20. Click focusable *Boolean*

Add focused state to the object when clicked.

W77.2.21. Checkable *Boolean*

Toggle checked state when the object is clicked.

W77.2.22. Scrollable *Boolean*

Make the object scrollable.

W77.2.23. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W77.2.24. Scroll momentum Boolean

Make the object scroll further when "thrown".

W77.2.25. Scroll one Boolean

Allow scrolling only one snappable children.

W77.2.26. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W77.2.27. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W77.2.28. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W77.2.29. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W77.2.30. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W77.2.31. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W77.2.32. Event bubble Boolean

Propagate the events to the parent too.

W77.2.33. Gesture bubble Boolean

Propagate the gestures to the parent.

W77.2.34. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W77.2.35. Ignore layout Boolean

Make the object positionable by the layouts.

W77.2.36. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W77.2.37. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W77.2.38. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W77.2.39. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W77.2.40. Checked** *EXPRESSION (boolean)*

Toggled or checked state.

W77.2.41. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W77.2.42. Disabled *EXPRESSION (boolean)*

Disabled state

W77.2.43. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W77.2.44. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W77.2.45. Pressed *Boolean*

Being pressed.

Events**W77.2.46. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W77.2.47. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W77.2.48. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W77.2.49. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W77.3. Examples

- *Tabview*
- *Styled Tabview*

W78. Terminal



W78.1. Description

Displays a Terminal window through which the user can enter arbitrary text, as the text is entered, character by character is sent through the `onData` output. It is also possible to enter text into the terminal through flow using the `Data` property.

W78.2. Properties

Specific

W78.2.1. Data EXPRESSION (string)

The text that is entered in the Terminal window. It is necessary to add flow input of type `string` or `stream` and enter the name of that input in this property. If the flow input is of the `string` type, then it is necessary to send a string to that input that you want to enter in the terminal – this can be done multiple times, i.e. every time a string is received at that input, it will be entered in the terminal. If the flow input is of `stream` type, then the Terminal Widget listens to see if there is any new data on the stream and when it appears, it writes it to the terminal – for example, in this way it is possible to connect `stdout` or `stderr` output from `ExecuteCommand` Actions on the Terminal Widget.

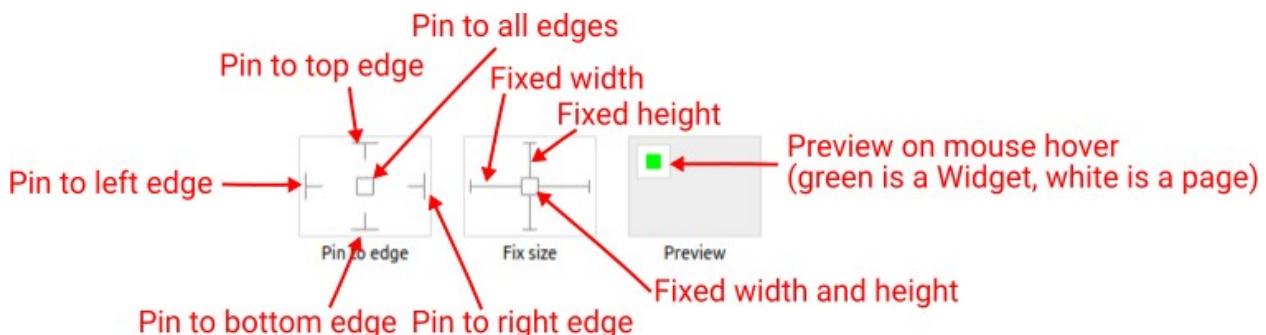
W78.2.2. Visible EXPRESSION (boolean)

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W78.2.3. Resizing Any

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

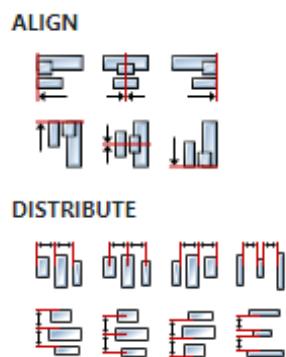
Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge*.

and *Fix width*.

W78.2.4. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W78.2.5. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W78.2.6. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W78.2.7. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W78.2.8. Width Number

The width of the component. It is set in pixels.

W78.2.9. Height Number

The height of the component. It is set in pixels.

W78.2.10. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W78.2.11. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style

W78.2.12. Default Object

Style used when rendering of the Widget.

Events**W78.2.13. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W78.2.14. Output widget handle Boolean**

If enabled then a new output named @Widget will be added. In runtime, upon a widget creation, a value of type widget will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is AddToInstrumentHistory action component when Plotly is selected for the Item type property. Then it is necessary to set the property Plotly widget to the reference to LineChart widget.

W78.2.15. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W78.2.16. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W78.2.17. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W78.3. Examples

- Dashboard Widgets Demo

W79. Textarea



W79.1. Description

The Text Area is a Widget with a Label and a cursor on it. Texts or characters can be added to it. Long lines are wrapped and when the text becomes long enough the Text area can be scrolled. One line mode and password modes are supported.

More info ([link](#))

W79.2. Properties

Specific

W79.2.1. Text *EXPRESSION (string)*

Text to be displayed.

W79.2.2. Text type *Enum*

Here we can choose that the `Text` item is calculated from the Expression.

W79.2.3. Placeholder *String*

A placeholder text can be specified – which is displayed when the `Text` area is empty.

W79.2.4. One line mode *Boolean*

If enable, the `Text` area is configured to be on a single line. In this mode the height is set automatically to show only one line, line break characters are ignored, and word wrap is disabled.

W79.2.5. Password mode *Boolean*

This enables password mode. By default, if the `\u2022` (Bullet, U+2022) character exists in the font, the entered characters are converted to it after some time or when a new character is entered. If `\u2022` does not exist in the font, `\u2022` will be used.

W79.2.6. Accepted characters *String*

We can set a list of accepted characters with this property. Other characters will be ignored.

W79.2.7. Max text length *Number*

The maximum number of characters can be limited with this property.

General

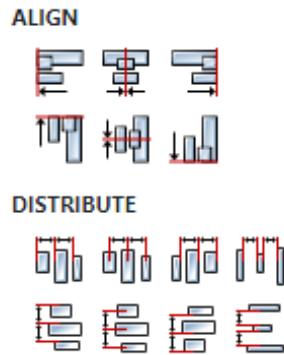
W79.2.8. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W79.2.9. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W79.2.10. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W79.2.11. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W79.2.12. Left unit Enum

The following options are available:

- px – Left is default in pixels.
- % – Left is set as a percentage in relation to the parent width.

W79.2.13. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W79.2.14. Top unit Enum

The following options are available:

- px – Top is set in pixels.
- % – The top is set as a percentage in relation to the parent height.

W79.2.15. Width Number

The width of the component. It is set in pixels.

W79.2.16. Width unit Enum

The following options are available:

- px – Width is given in pixels.
- % – Width is given as a percentage in relation to the parent width.
- content – Width is automatically set to fit the entire content in width.

W79.2.17. Height *Number*

The height of the component. It is set in pixels.

W79.2.18. Height unit *Enum*

The following options are available:

- `px` – Height is given in pixels.
- `%` – Height is given as a percentage in relation to the parent height.
- `content` – Height is automatically set to fit the entire content in height.

Layout**W79.2.19. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set the title of the tab that contains this widget.

Style**W79.2.20. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W79.2.21. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W79.2.22. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W79.2.23. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W79.2.24. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W79.2.25. Click focusable *Boolean*

Add focused state to the object when clicked.

W79.2.26. Checkable *Boolean*

Toggle checked state when the object is clicked.

W79.2.27. Scrollable *Boolean*

Make the object scrollable.

W79.2.28. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W79.2.44. Scroll direction *Enum*

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W79.2.45. Checked** *EXPRESSION (boolean)*

Toggled or checked state.

W79.2.46. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W79.2.47. Disabled *EXPRESSION (boolean)*

Disabled state

W79.2.48. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W79.2.49. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W79.2.50. Pressed *Boolean*

Being pressed.

Events**W79.2.51. Event handlers** *Array*

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W79.2.52. Inputs** *Array*

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W79.2.53. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

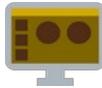
W79.2.54. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W79.3. Examples

- *LVGL Widgets Demo*

W80. Text (Dashboard)



W80.1. Description

A widget used to display text.

W80.2. Properties

Specific

W80.2.1. Text *EXPRESSION (any)*

Text to be displayed. This is an expression and if you only want to display some static text, then that text should be entered in quotation marks. If the expression uses variables, then that expression cannot be calculated in the editor, so the expression will be displayed instead of the text itself.

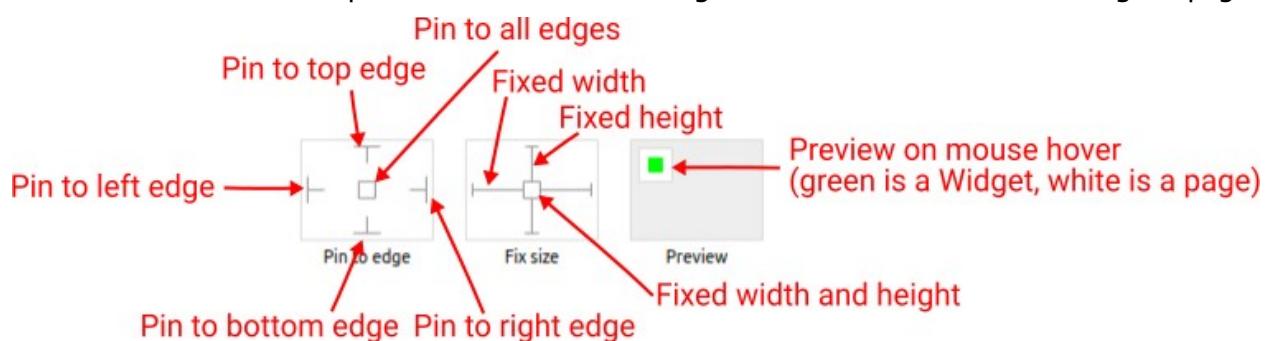
W80.2.2. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W80.2.3. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



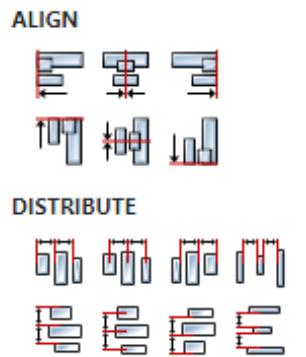
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W80.2.4. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

**W80.2.5. Center widget Any**

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W80.2.6. Left Number**

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the Top, Width and Height properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use +, -, *, and / operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: 18 + 36, 50 + 32 * 6, (100 - 32) / 2.

W80.2.7. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W80.2.8. Width Number

The width of the component. It is set in pixels.

W80.2.9. Height Number

The height of the component. It is set in pixels.

W80.2.10. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout**W80.2.11. Tab title EXPRESSION (string)**

If this widget is a child of a container with layout set to Docking Manager, use this property to set the title of the tab that contains this widget.

General**W80.2.12. Name String**

If an expression is used for the Text property that cannot be calculated during editing, then the text displayed in the editor can be set here. Also, this text will be displayed in the Widgets Structure

panel.

Style

W80.2.13. Default Object

Style that will be used to render the Widget.

Events

W80.2.14. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W80.2.15. Output widget handle Boolean

If enabled then a new output named `@Widget` will be added. In runtime, upon a widget creation, a value of type `widget` will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is `AddToInstrumentHistory` action component when `Plotly` is selected for the `Item type` property. Then it is necessary to set the property `Plotly widget` to the reference to LineChart widget.

W80.2.16. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W80.2.17. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W80.2.18. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W80.3. Examples

- *Dashboard Widgets Demo*

W81. Text (EEZ-GUI)



W81.1. Description

A widget used to display single line text.

W81.2. Properties

Specific

W81.2.1. Text *EXPRESSION (any)*

Text to be displayed. This is an expression and if you only want to display some static text, then that text should be entered in quotation marks. If the expression uses variables, then that expression cannot be calculated in the editor, so the expression will be displayed instead of the text itself.

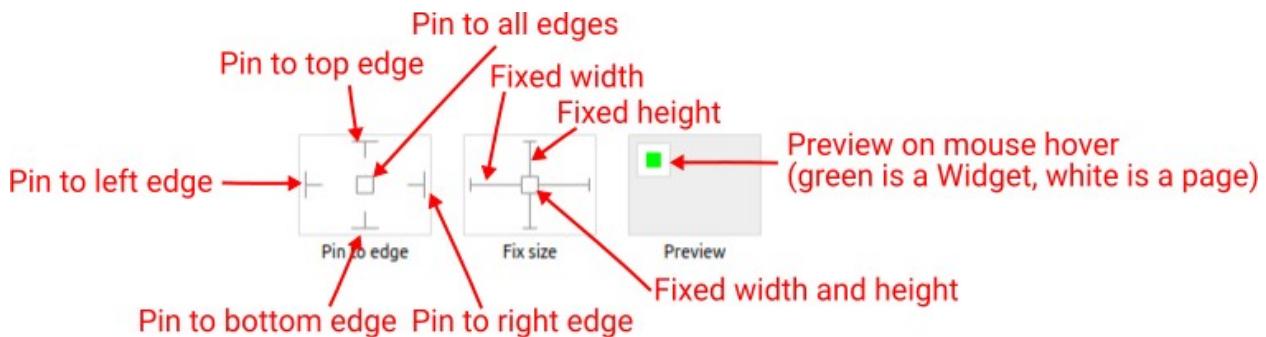
W81.2.2. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W81.2.3. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W81.2.4. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

W81.2.5. Align and distribute *Any*

If an expression is used for the `Text` property that cannot be calculated during editing, then the text displayed in the editor can be set here. Also, this text will be displayed in the *Widgets Structure* panel.

Style

W81.2.14. Default Object

Style that will be used to render the Widget.

W81.2.15. Focused Object

Style to be used for rendering if the Widget is in focus.

Events

W81.2.16. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W81.2.17. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project *Check* to check whether a data line that transmits data of that type is connected to the input or not.

W81.2.18. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component will not change the content of the variable in each step, but will send the current value through that output.

W81.2.19. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W81.3. Examples

- `eez-gui-widgets-demo`

W82. TextInput



W82.1. Description

This Widget is used when we want to enter a text.

W82.2. Properties

Specific

W82.2.1. Value *EXPRESSION (any)*

The variable in which the entered text will be stored.

W82.2.2. Read only *EXPRESSION (any)*

Set to `true` if you want for this Widget to be read only, i.e. disabled for user input.

W82.2.3. Placeholder *EXPRESSION (any)*

The text that is displayed at the beginning when nothing has been entered yet.

W82.2.4. Password *Boolean*

If password is entered, then this property should be enabled so that `*` is displayed instead of characters when entering the password.

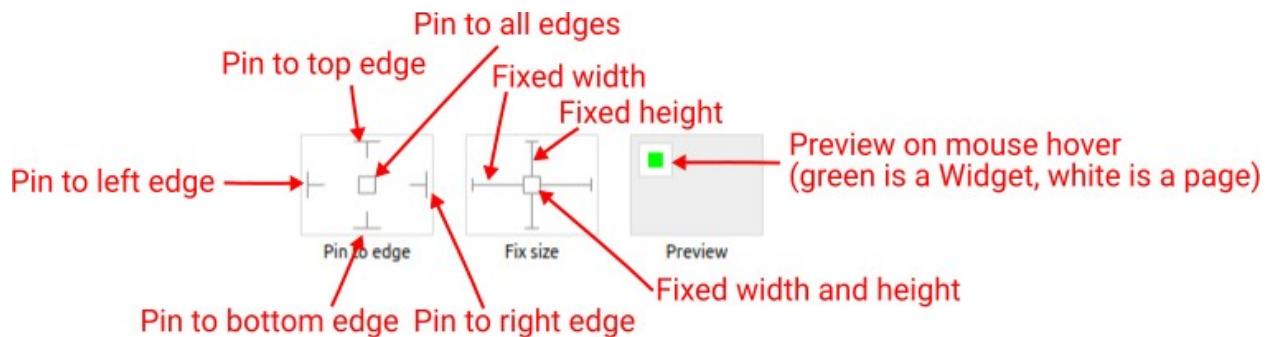
W82.2.5. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W82.2.6. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



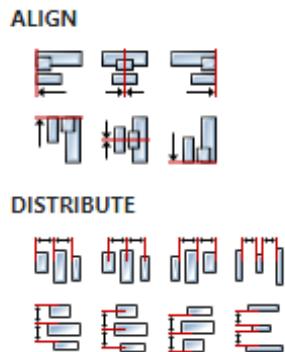
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

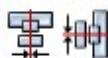
W82.2.7. Align and distribute Any

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W82.2.8. Center widget Any

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W82.2.9. Left Number

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W82.2.10. Top Number

Y position of the component in relation to the page or parent widget. It is set in pixels.

W82.2.11. Width Number

The width of the component. It is set in pixels.

W82.2.12. Height Number

The height of the component. It is set in pixels.

W82.2.13. Absolute position String

The absolute position of the component in relation to the page. This property is read-only.

Layout

W82.2.14. Tab title EXPRESSION (string)

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set

the title of the tab that contains this widget.

Style

W82.2.15. Default Object

Style used when rendering of the Widget.

Events

W82.2.16. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- Event – Event that is processed, e.g. CLICKED.
- Handler type – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- Action - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W82.2.17. Output widget handle Boolean

If enabled then a new output named @Widget will be added. In runtime, upon a widget creation, a value of type widget will be sent through this output. This value can be used in other parts of the flow when reference to the widget is required. One such example is AddToInstrumentHistory action component when Plotly is selected for the Item type property. Then it is necessary to set the property Plotly widget to the reference to LineChart widget.

W82.2.18. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W82.2.19. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W82.2.20. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W82.3. Examples

- *Dashboard Widgets Demo*

W83. TileView



W83.1. Description

This widget is work in progress, it means that you can add it to your project and Studio will generate all the code for its creation, but for anything more than that you should do it in your custom code, for example after `ui_init()` has been called.

More info ([link](#))

W83.2. Properties

General

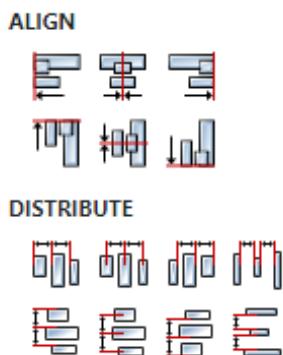
W83.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W83.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W83.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W83.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W83.2.5. Left unit *Enum*

W83.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W83.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W83.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W83.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W83.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W83.2.20. Scrollable *Boolean*

Make the object scrollable.

W83.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W83.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W83.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W83.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W83.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W83.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W83.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W83.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W83.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W83.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W83.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W83.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W83.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W83.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W83.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W83.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W83.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W83.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W83.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W83.2.40. Disabled EXPRESSION (boolean)

Disabled state

W83.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W83.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W83.2.43. Pressed Boolean

Being pressed.

Events**W83.2.44. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W83.2.45. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W83.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W83.2.47. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W84. ToggleButton



W84.1. Description

A button that can be in two states: `Default` or `Checked`.

W84.2. Properties

Specific

W84.2.1. Data *EXPRESSION (boolean)*

If the value of this property is `false` then the button is in the `Default` state, and if the value is `true` then it is in the `Checked` state

W84.2.2. Text1 *String*

The text that is displayed when the Widget is in the `Default` state.

W84.2.3. Text2 *String*

The text that is displayed when the Widget is in the `Checked` state.

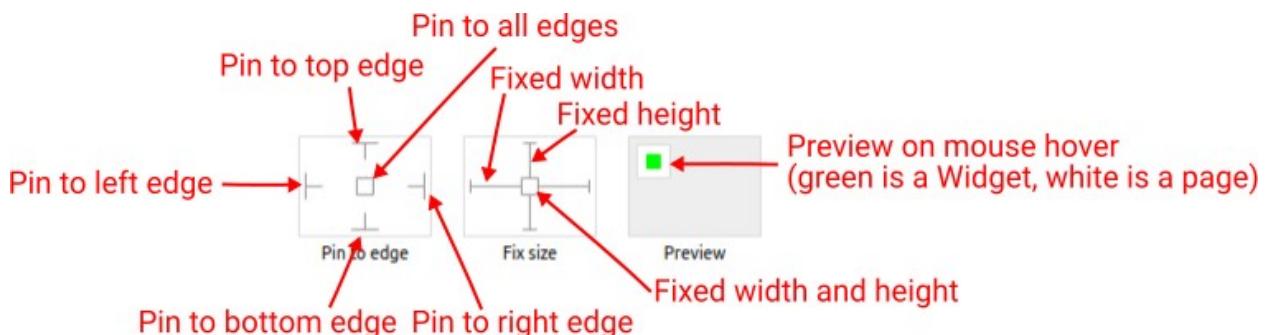
W84.2.4. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W84.2.5. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and *Pin to bottom edge* and *Fix width*.

W84.2.6. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

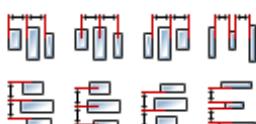
W84.2.7. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.

ALIGN



DISTRIBUTE

**W84.2.8. Center widget** *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.

**W84.2.9. Left** *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W84.2.10. Top *Number*

Y position of the component in relation to the page or parent widget. It is set in pixels.

W84.2.11. Width *Number*

The width of the component. It is set in pixels.

W84.2.12. Height *Number*

The height of the component. It is set in pixels.

W84.2.13. Absolute position *String*

The absolute position of the component in relation to the page. This property is read-only.

Layout**W84.2.14. Tab title** *EXPRESSION (string)*

If this widget is a child of a container with layout set to `Docking Manager`, use this property to set

the title of the tab that contains this widget.

Style

W84.2.15. Default Object

Style to be used for rendering if the Widget is the `Default` state.

W84.2.16. Checked Object

Style to be used for rendering if the Widget is the `Checked` state.

Events

W84.2.17. Event handlers Array

List of event handler definitions. During execution, the widget can generate certain events (e.g. the `CLICKED` event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- `Event` – Event that is processed, e.g. `CLICKED`.
- `Handler type` – There are two options: `Flow` or `Action`. If `Flow` is selected, a flow output will be added through which the event is processed, and if `Action` is selected, then it is necessary to specify which User action will be performed during event processing.
- `Action` - If the `Handler type` is set to `Action`, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow

W84.2.18. Inputs Array

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project `Check` to check whether a data line that transmits data of that type is connected to the input or not.

W84.2.19. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component will not change the content of the variable in each step, but will send the current value through that output.

W84.2.20. Catch error Boolean

If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W84.3. Examples

- `eez-gui-widgets-demo`

W85. UpDown



W85.1. Description

This Widget allows us to select a single value using the decrement and increment buttons.

W85.2. Properties

Specific

W85.2.1. Data *EXPRESSION (integer)*

The variable in which the selected value in the range of [Min, Max] is saved.

W85.2.2. Down button text *String*

The text that is displayed inside the button for the decrement value.

W85.2.3. Up button text *String*

The text that is displayed inside the button for the increment value.

W85.2.4. Min *EXPRESSION (any)*

The minimum value that can be selected.

W85.2.5. Max *EXPRESSION (any)*

The maximum value that can be selected.

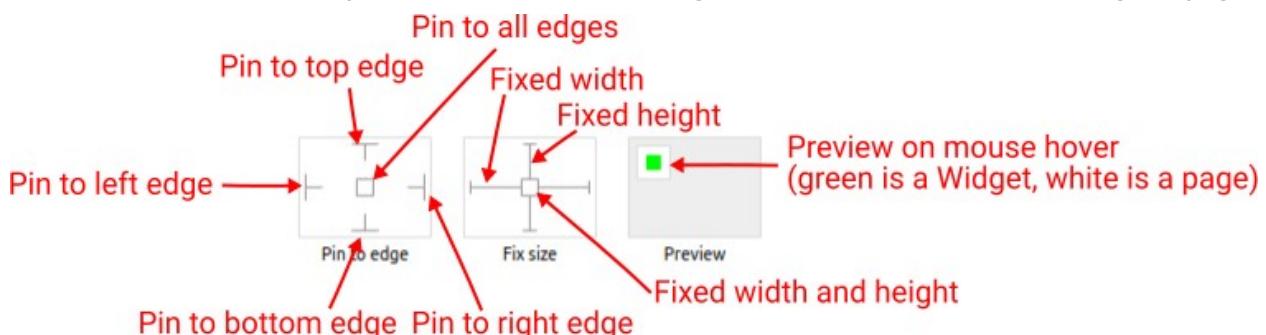
W85.2.6. Visible *EXPRESSION (boolean)*

If the calculated expression is true then the Widget is visible, and if it is false then the Widget is hidden. It can be left blank, in which case the Widget is always visible.

Position and size

W85.2.7. Resizing *Any*

If the page where this Widget is located has the "Scale to fit" option enabled, then this option can be used to control how the position and size of the widget will be calculated when scaling the page:



With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top edge of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the

W85.2.18. Buttons Object

Style used to render the button.

Events**W85.2.19. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W85.2.20. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W85.2.21. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W85.2.22. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

W85.3. Examples

- [eez-gui-widgets-demo](#)

W86. Window



W86.1. Description

This widget is work in progress, it means that you can add it to your project and Studio will generate all the code for its creation, but for anything more than that you should do it in your custom code, for example after `ui_init()` has been called.

More info ([link](#))

W86.2. Properties

General

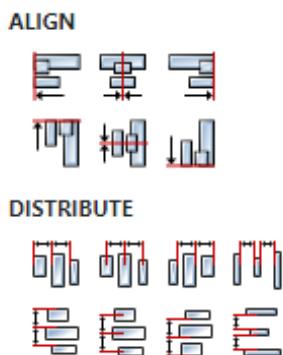
W86.2.1. Name *String*

Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

W86.2.2. Align and distribute *Any*

Alignment icons and component distribution. Alignment icons appear when two or more components are selected, and distribution icons appear when three or more components are selected.



W86.2.3. Center widget *Any*

Icons for horizontal and vertical centering of widgets within a page or parent widget.



W86.2.4. Left *Number*

X position of the component in relation to the page or parent widget. It is set in pixels.

Hint: when setting the value of this property (as well as the `Top`, `Width` and `Height` properties), simple mathematical expressions can be used. When we enter an expression and press enter, the expression will be evaluated and the result set as the value of this property. It is allowed to use `+`, `-`, `*` and `/` operators in expressions. Brackets can also be used.

Examples of such mathematical expressions: `18 + 36`, `50 + 32 * 6`, `(100 - 32) / 2`.

W86.2.5. Left unit *Enum*

W86.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W86.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W86.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W86.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W86.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W86.2.20. Scrollable *Boolean*

Make the object scrollable.

W86.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W86.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W86.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W86.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W86.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W86.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W86.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W86.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W86.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W86.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W86.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W86.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W86.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W86.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W86.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

W86.2.36. Scrollbar mode Enum

Scrollbars are displayed according to a configured mode. The following mode(s) exist:

- OFF: Never show the scrollbars
- ON: Always show the scrollbars
- ACTIVE: Show scroll bars while an object is being scrolled
- AUTO: Show scroll bars when the content is large enough to be scrolled

W86.2.37. Scroll direction Enum

Controls the direction in which scrolling happens. The following mode(s) exist:

- NONE: no scroll
- TOP: only scroll up
- LEFT: only scroll left
- BOTTOM: only scroll down
- RIGHT: only scroll right
- HOR: only scroll horizontally
- VER: only scroll vertically
- ALL: scroll any directions

States**W86.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W86.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W86.2.40. Disabled EXPRESSION (boolean)

Disabled state

W86.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W86.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W86.2.43. Pressed Boolean

Being pressed.

Events**W86.2.44. Event handlers Array**

List of event handler definitions. During execution, the widget can generate certain events (e.g. the CLICKED event is generated when the touchscreen is pressed and released within the Widget) and through this list we can specify the method of event processing. We must define these properties for each event handler:

- **Event** – Event that is processed, e.g. CLICKED.
- **Handler type** – There are two options: Flow or Action. If Flow is selected, a flow output will be added through which the event is processed, and if Action is selected, then it is necessary to specify which User action will be performed during event processing.
- **Action** - If the Handler type is set to Action, then here we need to enter the name of the User action that will be performed during the processing of the selected event.

Flow**W86.2.45. Inputs Array**

Additional component inputs that the user can add as desired in order to use them to receive additional data needed when evaluating expressions in properties. Each input is given a name and type. Name is used when referencing an input within an expression. A type is used to project Check to check whether a data line that transmits data of that type is connected to the input or not.

W86.2.46. Outputs Array

Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the Loop component, where we can put the output name for the Variable property instead of e.g. variable name. In that case, the Loop component will not change the content of the variable in each step, but will send the current value through that output.

W86.2.47. Catch error Boolean

If this checkbox is enabled then an @Error output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

*EEZ Studio
Instrument*

Table of Contents

<i>EEZ Studio Instrument</i>	I.1
I1. Home page instrument sections.....	I.5
I1.1. History.....	I.5
I1.1.1. History search.....	I.6
I1.1.2. Instrument sessions.....	I.7
I1.1.3. History filters.....	I.7
I1.1.4. Instrument Scrapbook.....	I.8
I1.2. Shortcuts and Groups.....	I.9
I1.3. Notebooks.....	I.10
I1.4. Items purge and restore.....	I.14
I1.5. Adding audio and video recordings.....	I.15
I1.6. Instrument Extension (IEXT) Manager.....	I.18
I1.7. Add instrument.....	I.19
I1.8. Establishing a connection with the instrument.....	I.21
I1.9. Export.....	I.23
I1.10. Import.....	I.25
I2. Instrument activity bar.....	I.27
I2.1. Start page (EEZ BB3 only).....	I.27
I2.2. Dashboard.....	I.29
I2.3. Terminal (SCPI protocol).....	I.29
I2.4. Terminal (Proprietary protocol).....	I.30
I2.5. Scripts.....	I.32
I2.5.1. Edit script shortcut.....	I.33
I2.6. Shortcuts.....	I.34
I2.7. Lists.....	I.35
I2.7.1. Editing a list using a table.....	I.36
I2.7.2. Editing a list using an envelope.....	I.36
I2.7.3. List view options.....	I.39
I2.7.4. List help.....	I.39

I1. Home page instrument sections

The top of the *Instruments* tab page (Fig. 1) contains a Search bar (1) for filtering by the name of the instruments whose thumbnails will be displayed (3). At the top of the page (2) there are also general options related to Instruments.

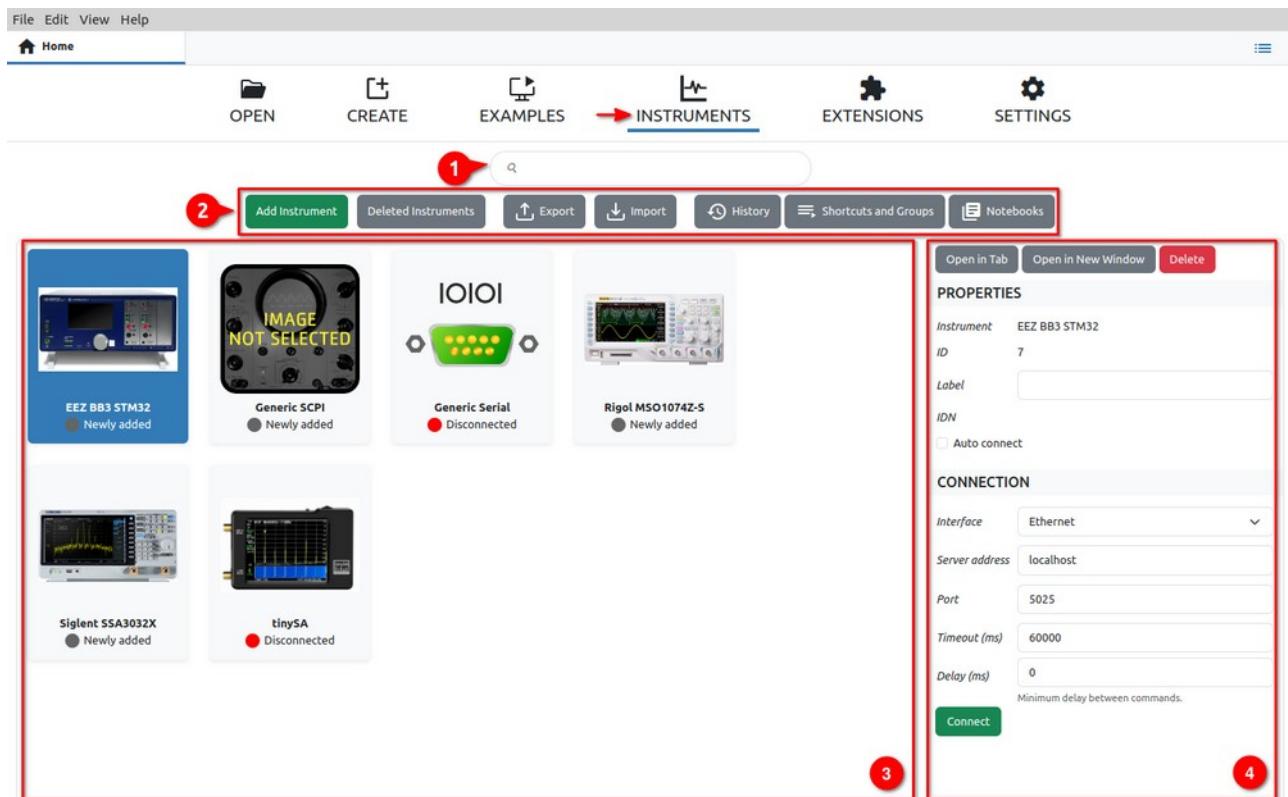


Fig. 1: Home page instrument options

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 (1) in the same way as in the *Terminal* of the currently selected instrument, as will be described below.

History can be searched (2) and several options are offered for positioning to a certain position in history according to different criteria, which are displayed in separate tabs (3).

In order to display the requested part of history faster, it is not loaded in its entirety, but only as many items as fit in the window. For navigation, in addition to the Scroll bar and navigation keys on the keyboard, it is possible to use the *Jump To Present* (4) button.

EEZ Studio Reference Guide



Fig. 2: Instruments History view

11.1.1. History search

To search history, we need to enter the phrase we are looking for, when a new *Search results* tab will be displayed with all the items that contain the searched phrase. For example in Fig. 3, a search with "no load" returned two items. Clicking on an item positions us on the item in history.

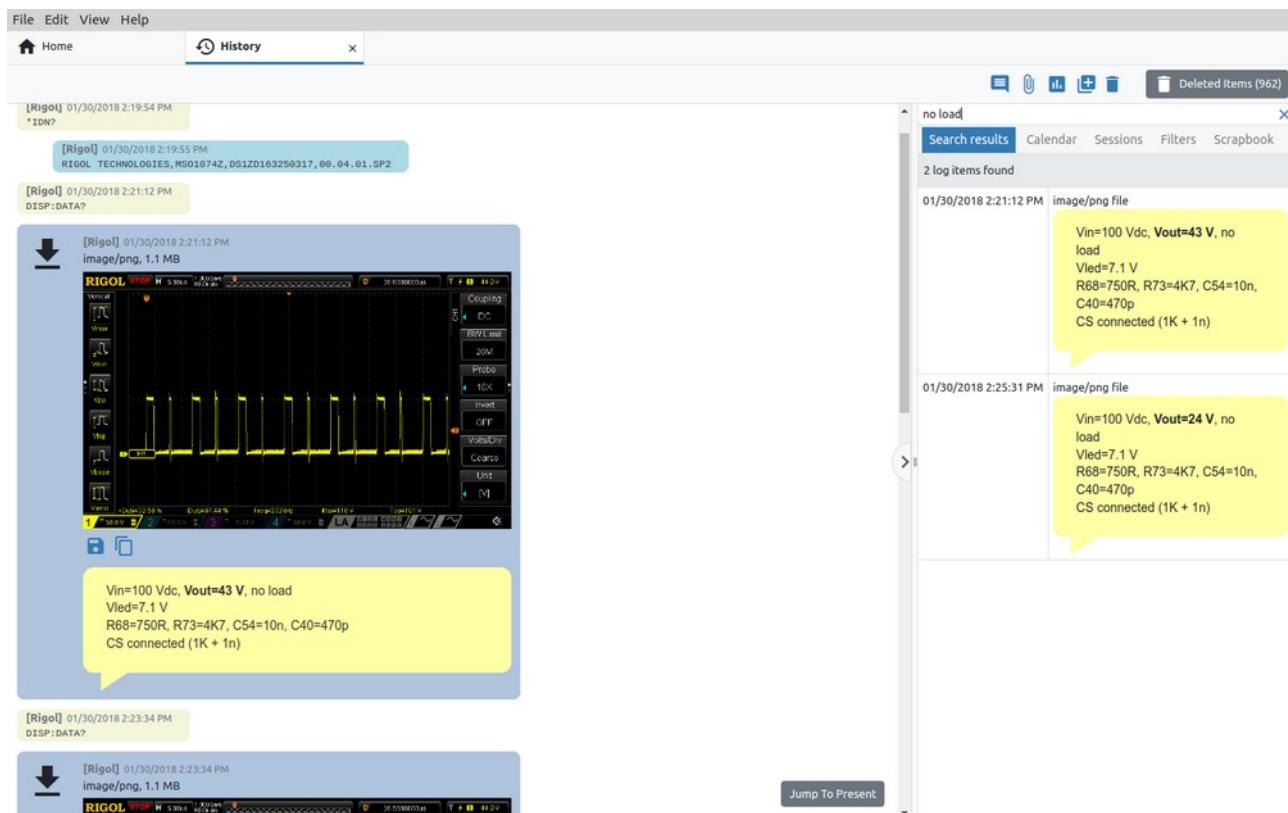


Fig. 3: Instrument history search results

11.1.2. Instrument sessions

Sessions (Fig. 4) are offered to simplify the process of documenting and navigating the history of working with instruments. A new session can be created (1) at any time and preferably contains a description of the task to be started in its name. The name can be edited later (2), and the session can be deleted (3). Deleted sessions go first to trash (4) from where they can be restored or deleted forever.

The list of sessions shows the time of the last use (*Last Activity* column) in addition to the name. The list can be sorted in descending or ascending order.

If we do not want to work with sessions or we want to display all items from the history, it is necessary to select *FREE MODE* from the list of sessions (5).

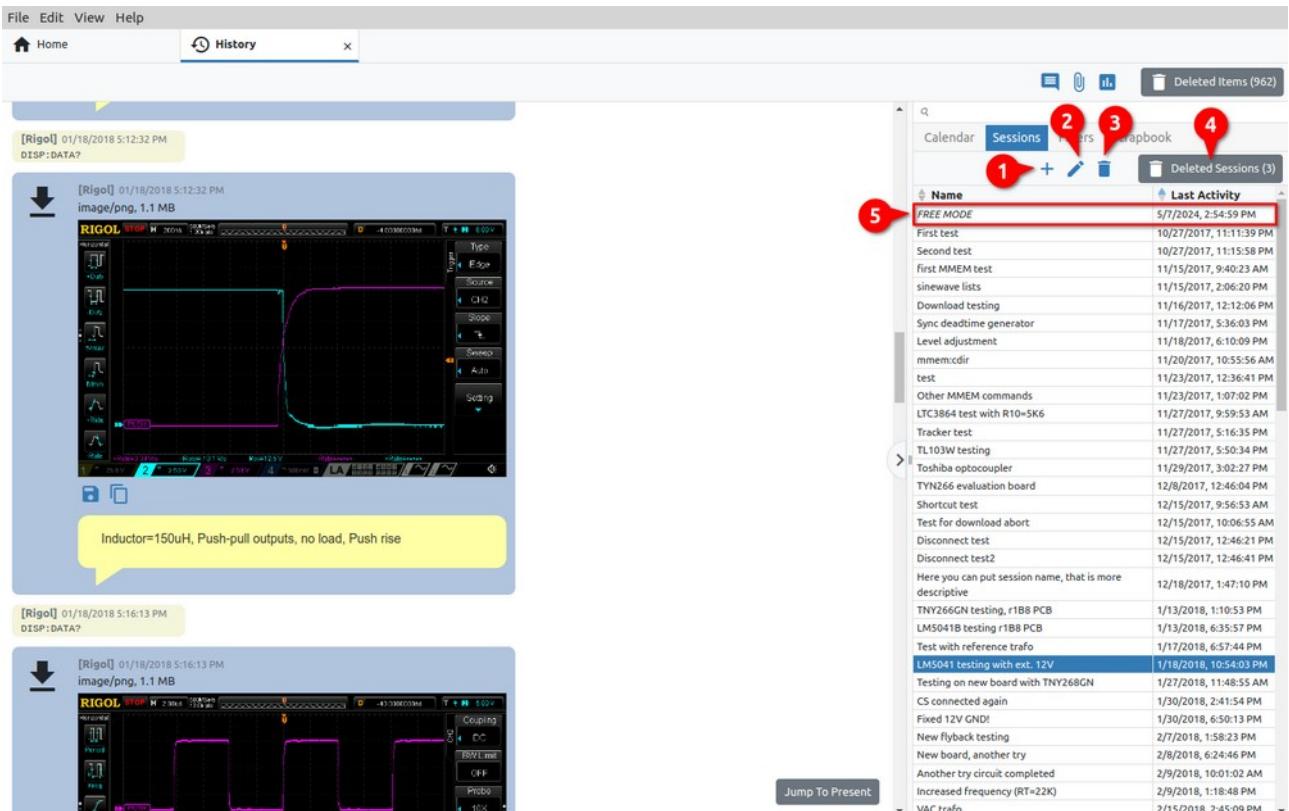


Fig. 4: Instrument sessions

An indication of an active session will be visible in any Instrument tab. In the example in Fig. 5 shows that the session with the name `new session` is active.

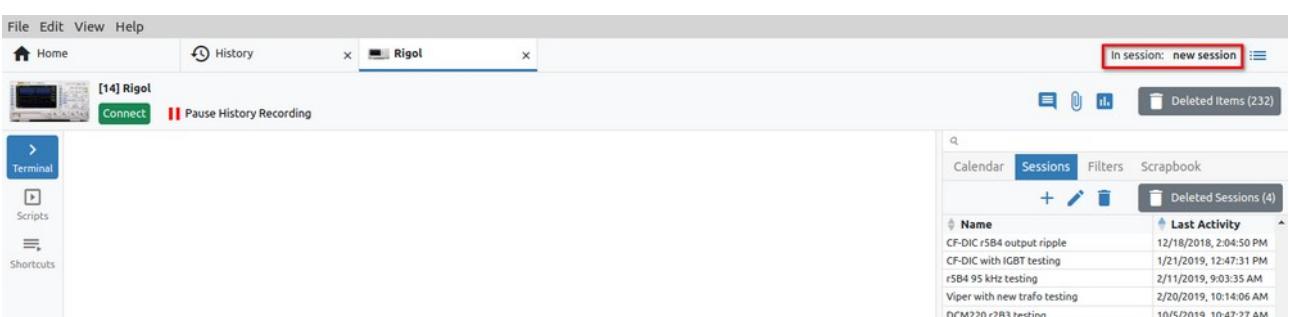


Fig. 5: Indication of an active session

11.1.3. History filters

The *Filters* tab shows the names of all types of History items and the number of them found in the database. Here we can select which of them will be displayed in the History view on the left (Fig. 6).

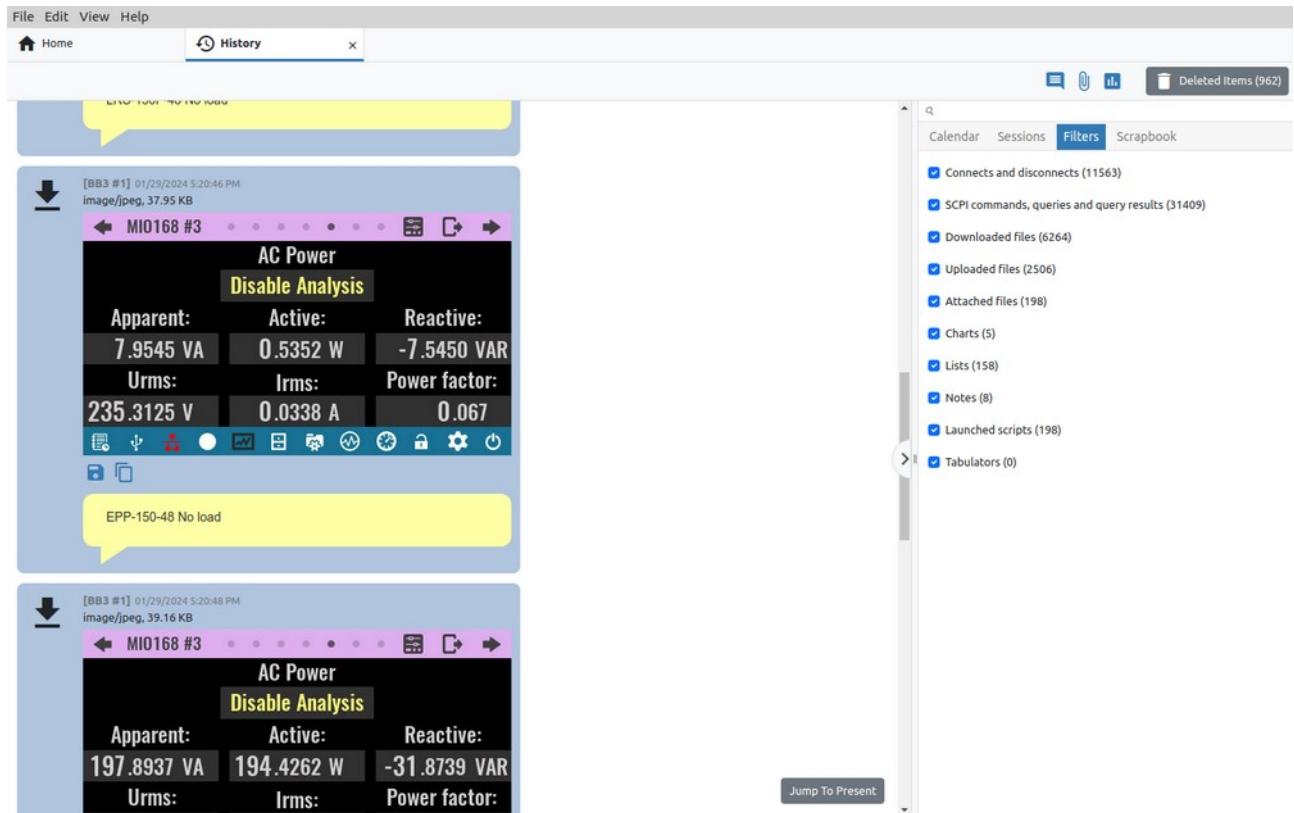


Fig. 6: History view filters

11.1.4. Instrument Scrapbook

The instrument scrapbook (Fig. 7) is a convenient way to have important items from the history at hand, regardless of which instrument we are working with and which session we are in.

To add an item to the scrapbook, it is enough to select the item from the History view and drag&drop it into the Scrapbook tab area, where the thumbnail of the item will appear. We use the Thumbnail size option to determine the size of the displayed item in the Scrapbook.

Please note that the Instrument scrapbook is in no way related to Project Scrapbook described in P3.2.

11. Home page instrument sections

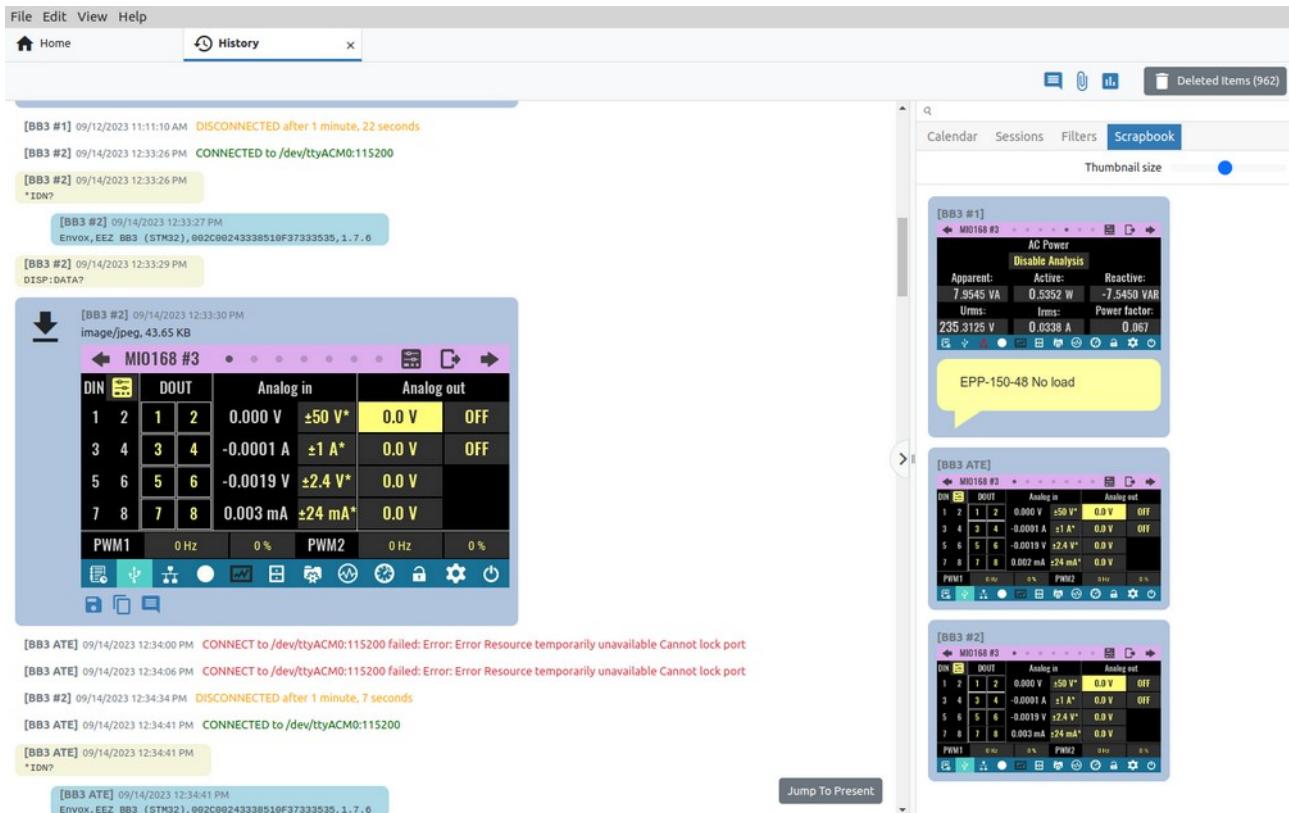


Fig. 7: Instrument scrapbook

11.2. Shortcuts and Groups

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

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

EEZ Studio Reference Guide

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	F6	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. 8: Instruments Shortcuts and Groups view

11.3. 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.

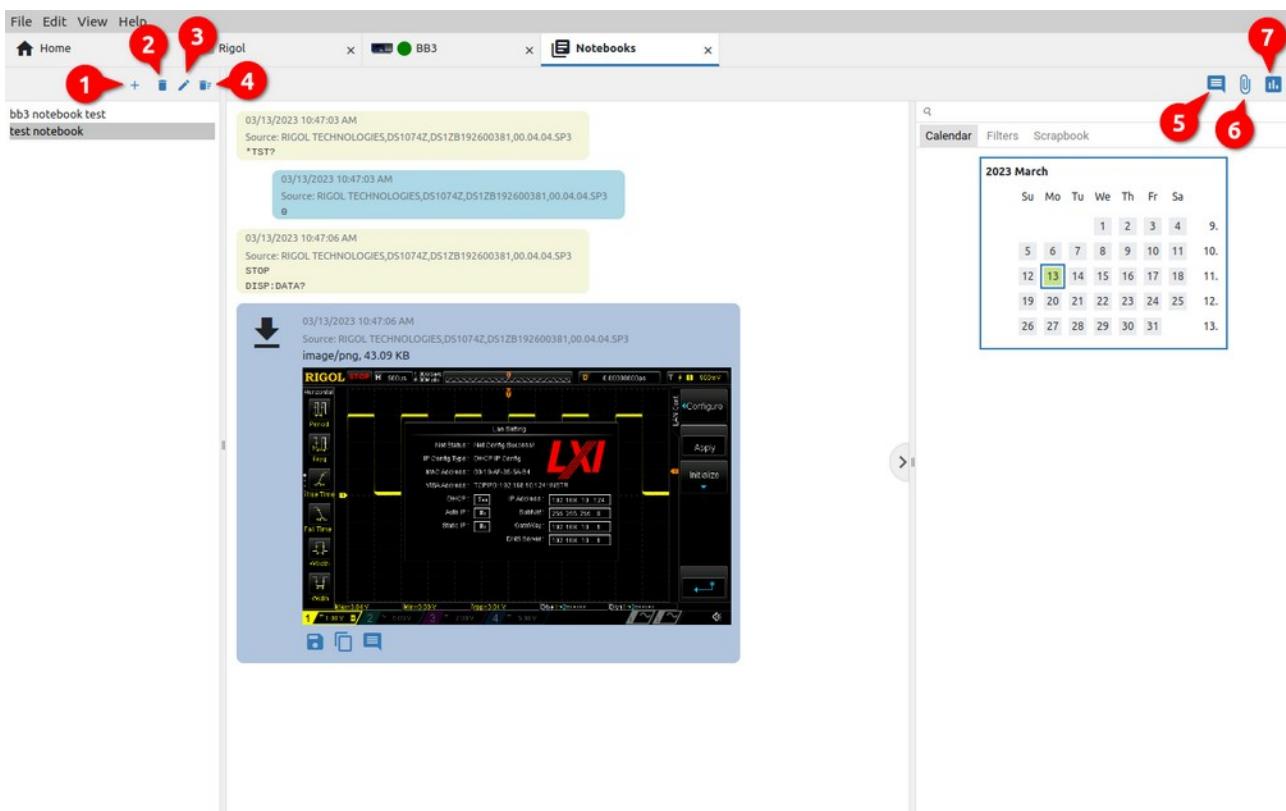


Fig. 9: Instrument Notebooks view

#	Option	Description
1	Add / Import notebook	<p>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 <i>Terminal</i>, as shown in Fig. 11: (1) go to the <i>Terminal</i> tab in the <i>Action bar</i>, (2) select one or more items and (3) export them to a notebook file, a new notebook or an already created notebook.</p> <p>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.</p>

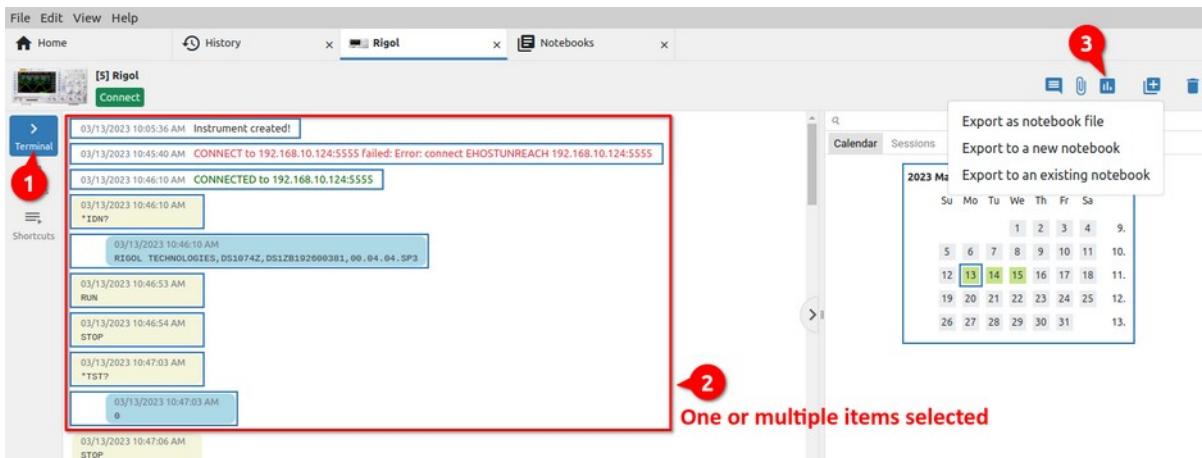


Fig. 10: Adding items to the notebook

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

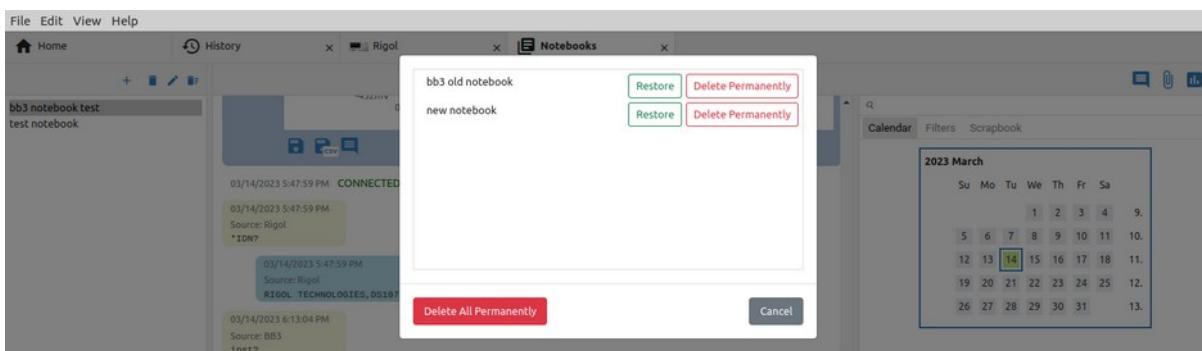


Fig. 11: Deleted notebooks

5	Add note	Adding a note to the notebook (Fig. 12). The number of notes is not limited and the last added note will appear at the bottom of the notebook (Fig. 13).
---	----------	--

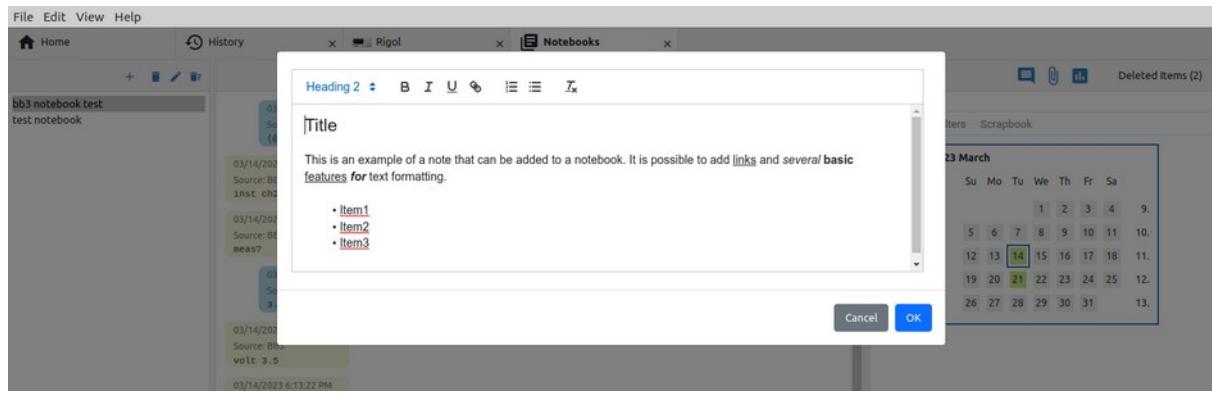


Fig. 12: Adding a new note to the notebook

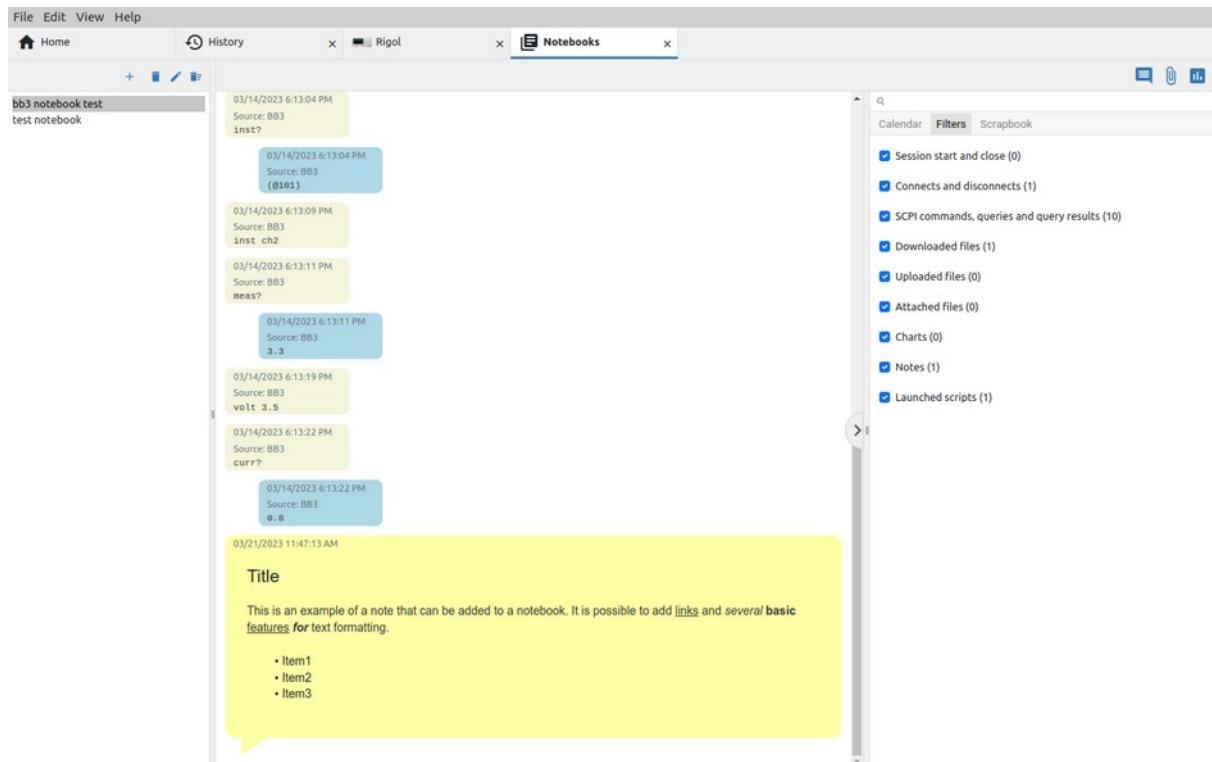


Fig. 13: Newly added note in the notebook

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. 14).

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.

11. Home page instrument sections

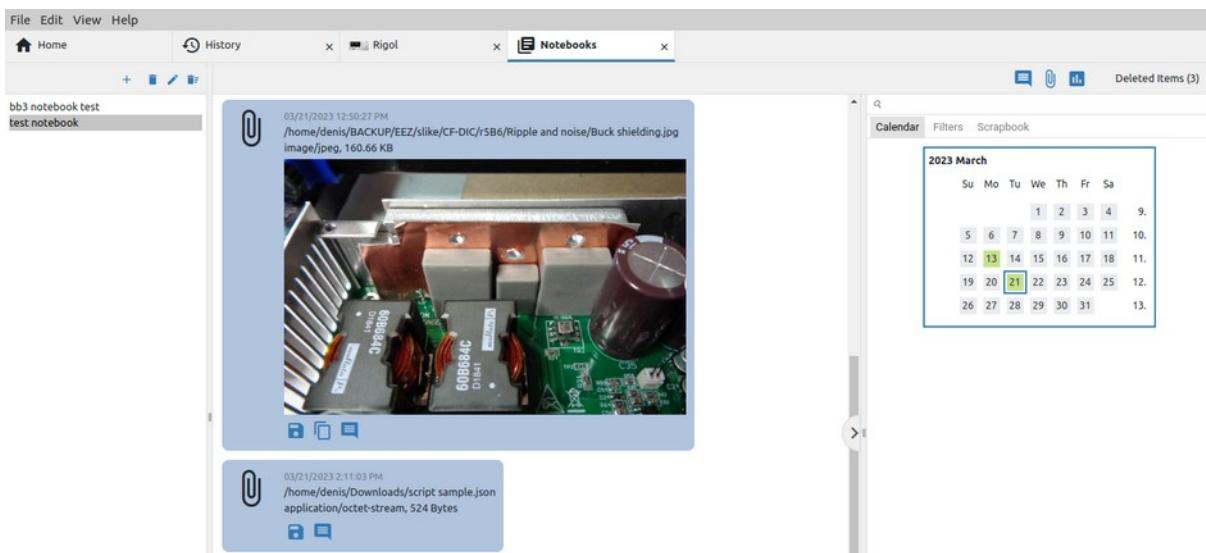


Fig. 14: Files imported into the notebook

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. 15.

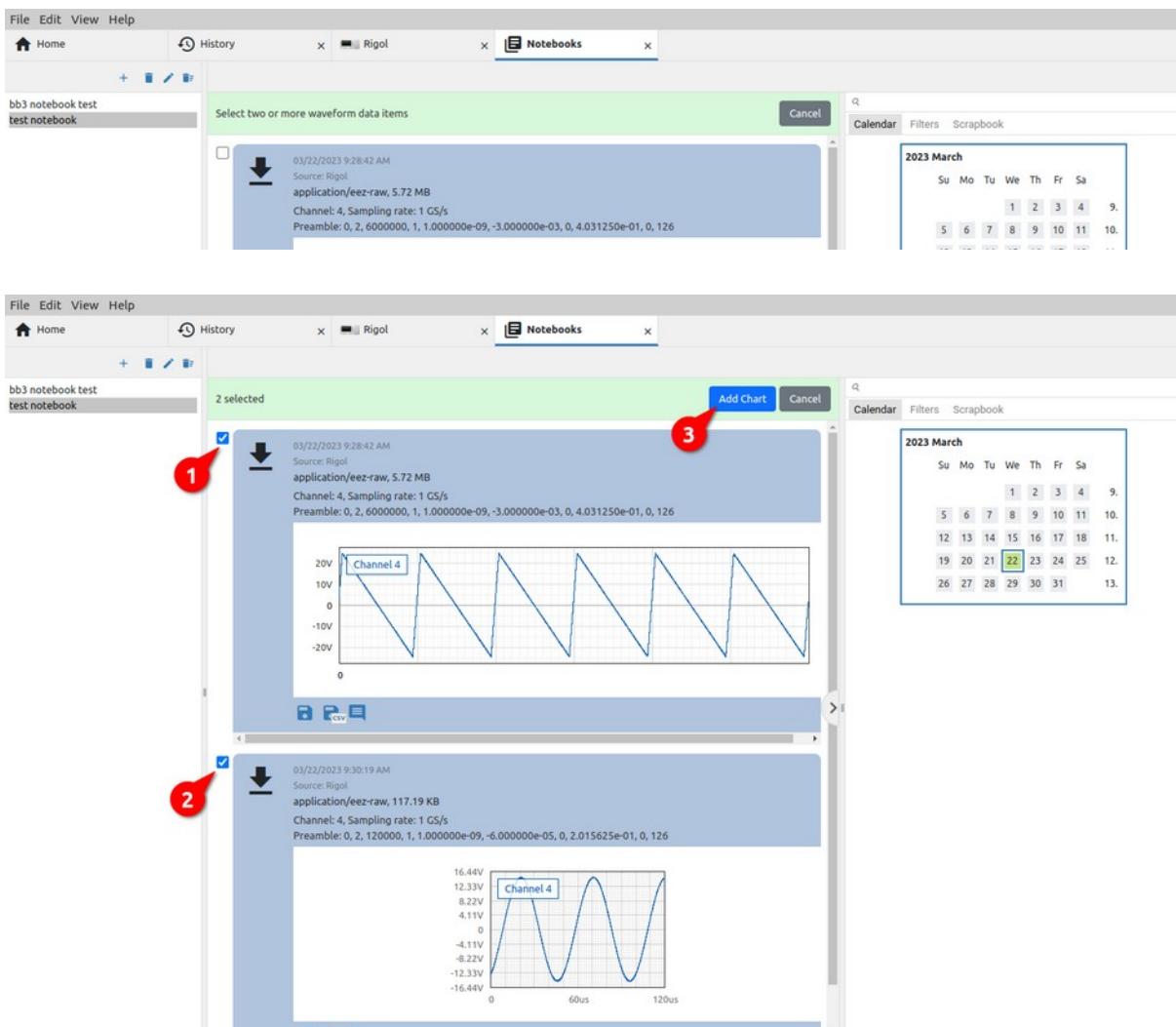


Fig. 15: A selection of graphs to add to the notebook

A successfully created graph will appear at the end of the notebook and will have a graph icon in the upper left corner (Fig. 16).

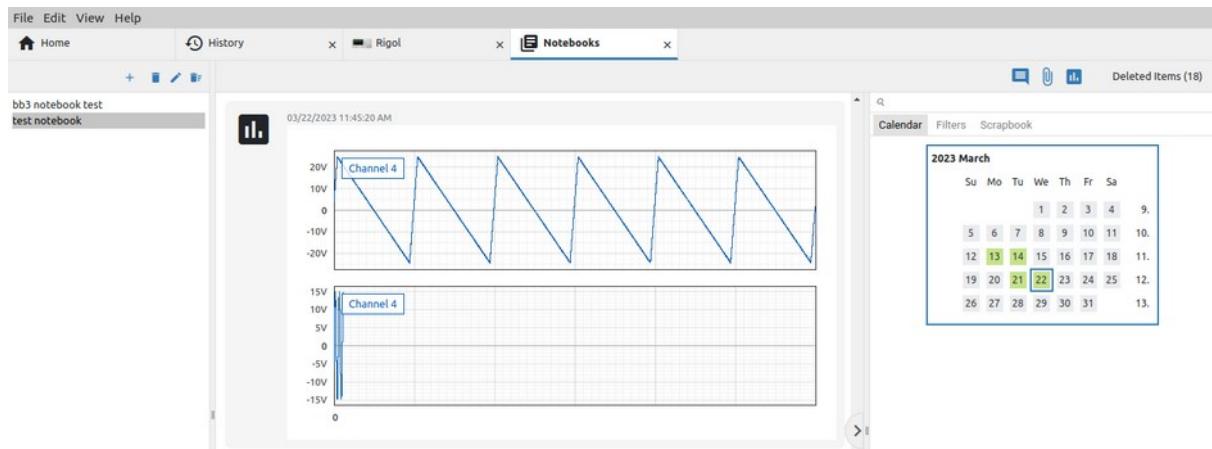


Fig. 16: Newly created graph added to notebook

11.4. 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. 17.

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.

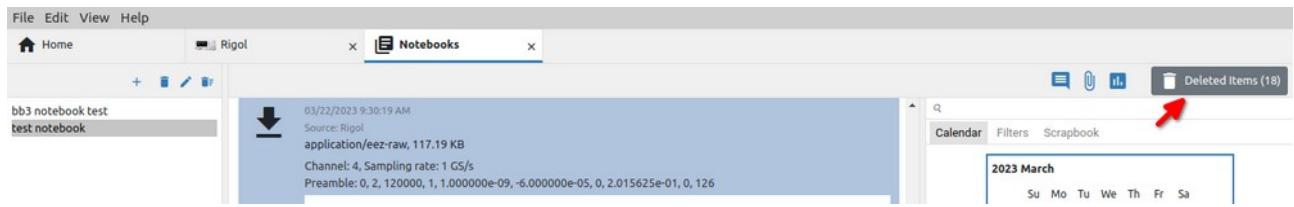


Fig. 17: Deleted items counter

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. 18).



Fig. 18: 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. 19).

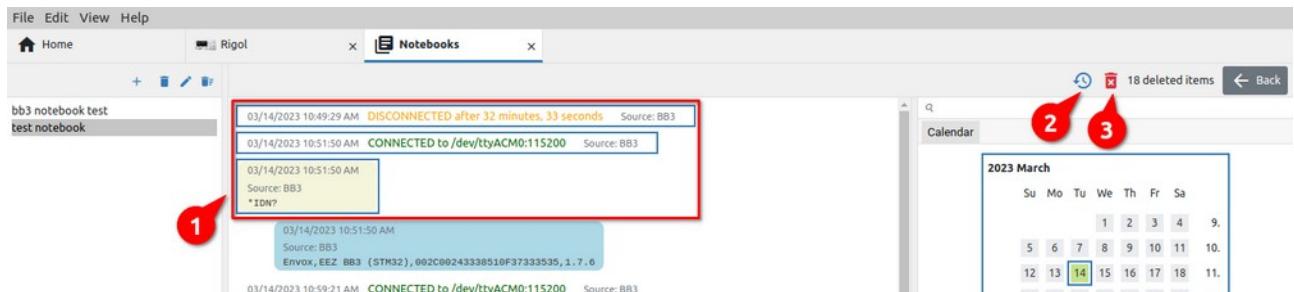


Fig. 19: Selection of deleted items for restore or purge

I1.5. Adding audio and video recordings

Sound and image recording allows adding voice comments as well as a recording of an event during the use of the instrument in an experiment, test, measurement, etc. The position of the recording options are shown in Fig. 20.

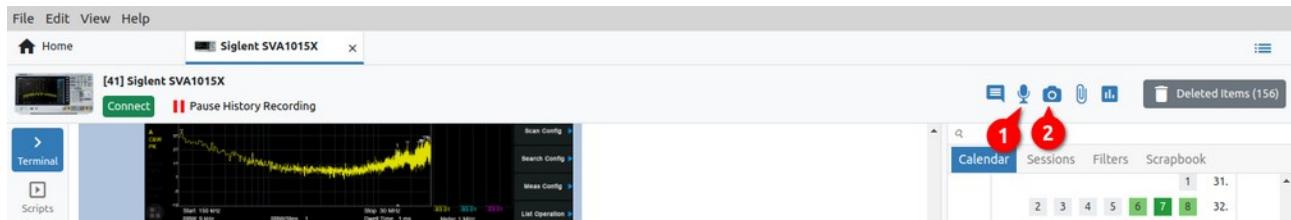


Fig. 20: Options for audio and video recording in the toolbar

If sound recording is selected, a dialog box will appear in which you can select the sound source and see a preview of the sound. Recording starts when the *Start* button is pressed Fig. 21.

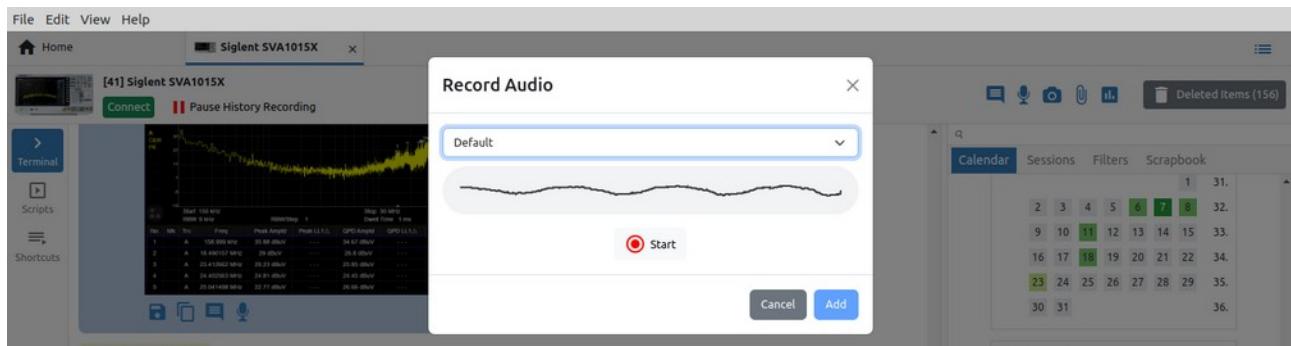


Fig. 21: Audio source selection and preview

During recording, the duration of the recording will be displayed and it will be possible to pause or stop the recording (Fig. 22).



Fig. 22: Audio recording in progress

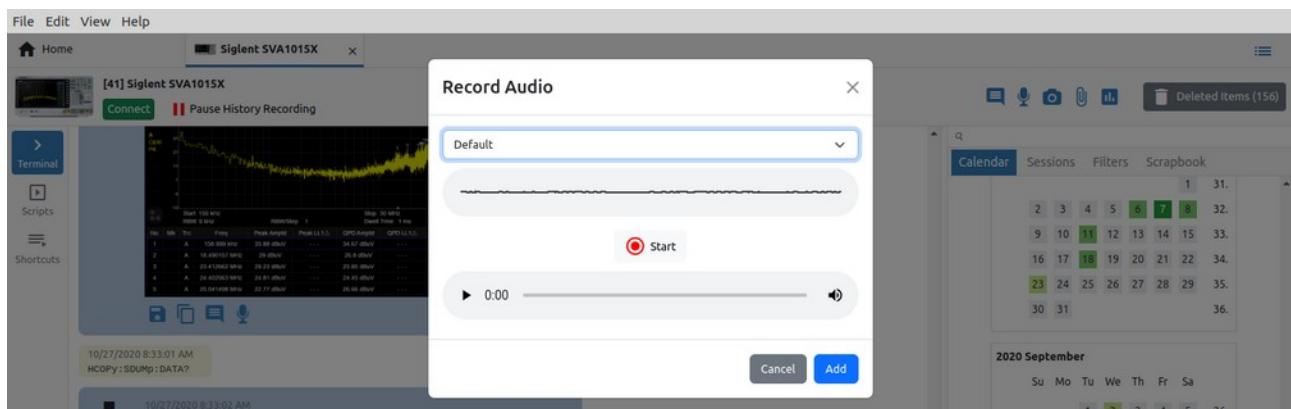


Fig. 23: Completed sound recording

When the recording is finished, a player will be displayed where you can listen to the recording. By selecting the *Add* option, the recording will be saved (Fig. 23). The saved recording will be added to the bottom of the instrument history (Fig. 24).

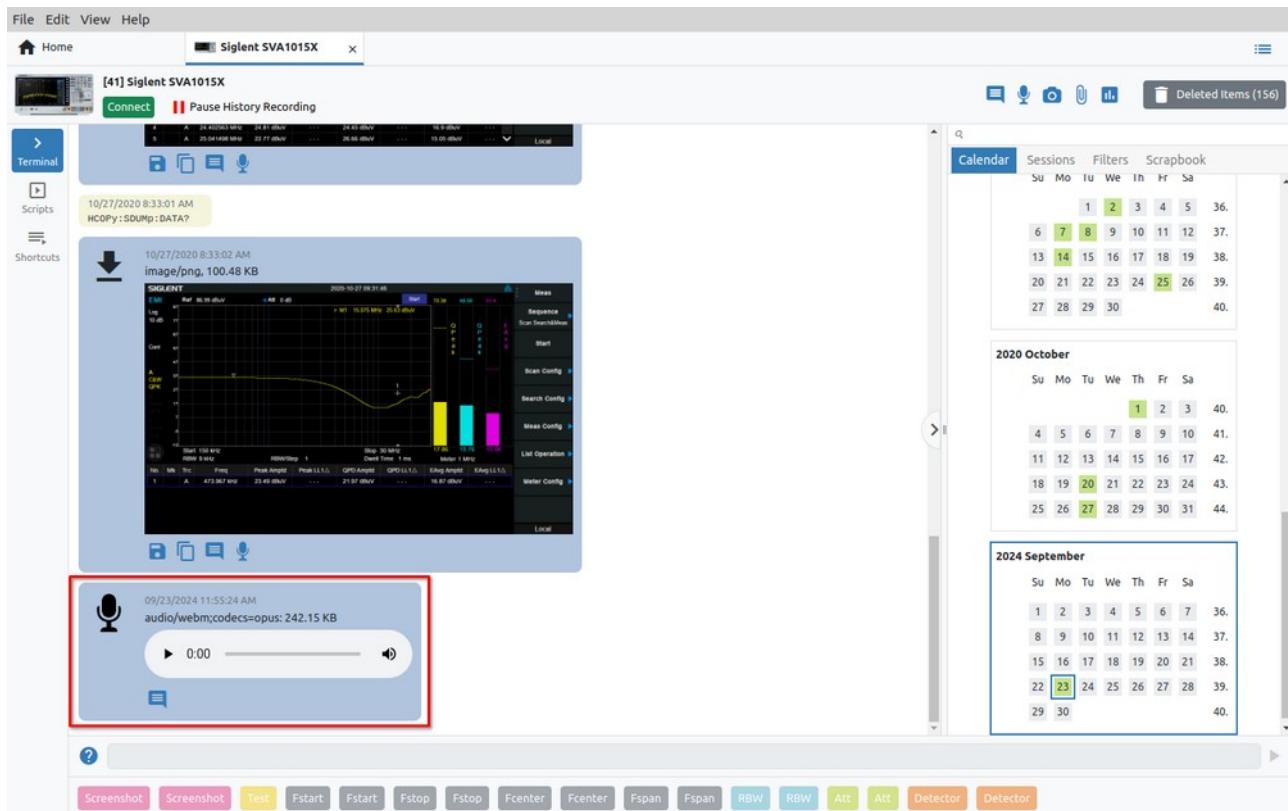


Fig. 24: Audio recordings added to instrument session

Video recording is done in a similar way to audio recording. After selecting the video recording option, and selecting the video source, a preview of that source will appear (Fig. 25).

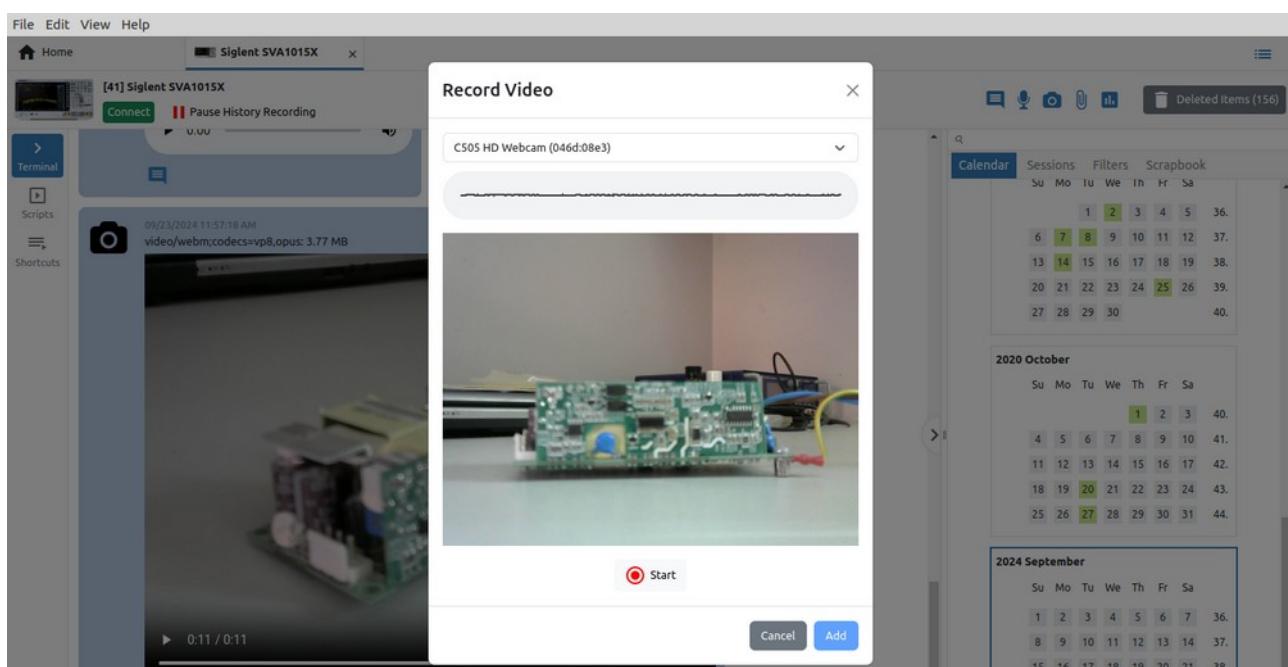


Fig. 25: Video source selection and preview

The recording starts by clicking the Start button, and after the recording is finished and saved, it will be visible at the bottom of the instrument history as in Fig. 26.

11. Home page instrument sections

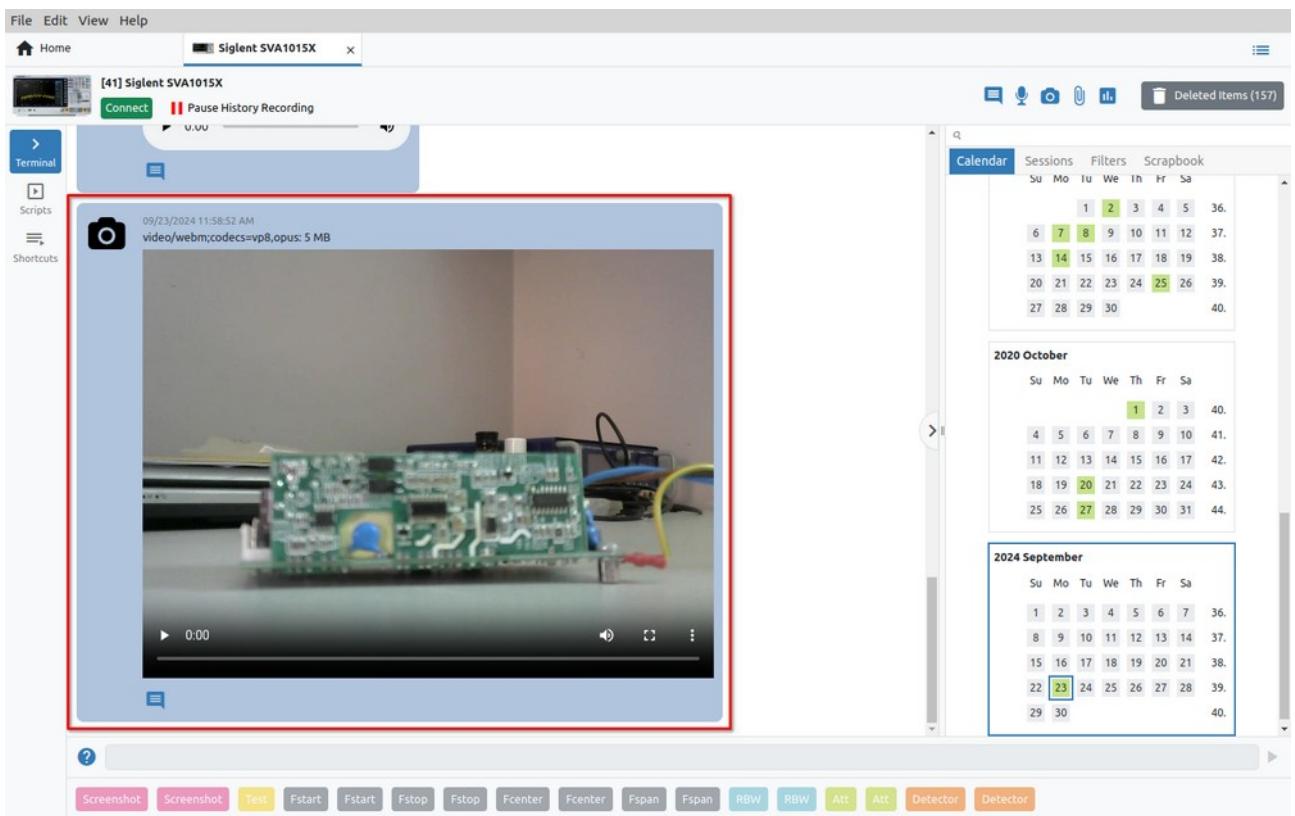


Fig. 26: Video recordings added to instrument session

I1.6. 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?).

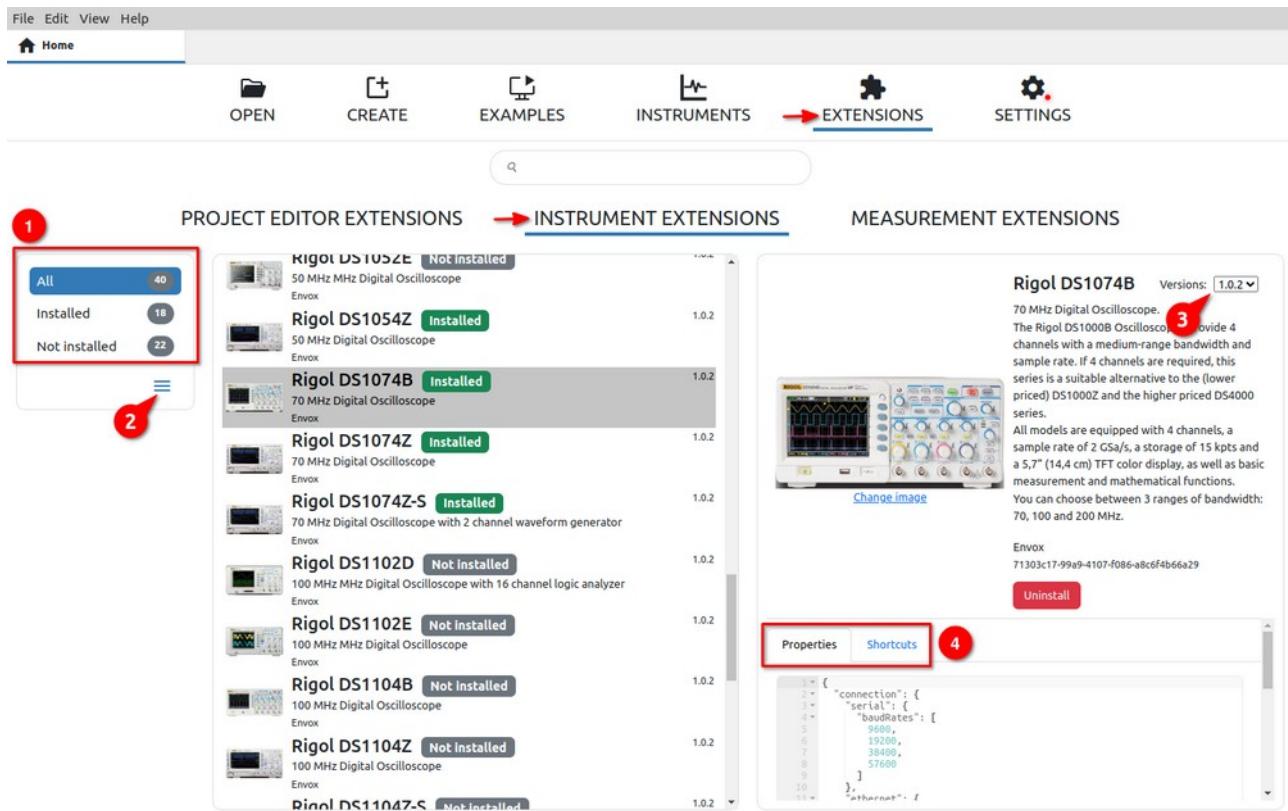


Fig. 27: Instrument extension (IEXT) Manager view

#	Option	Description
1	<i>View</i>	Filters for displaying IEXT in the list: it is possible to display all, only installed or only those that are not installed. The number of filtered IEXTs is displayed next to each option.
2	<i>Update / Install actions</i>	All approved IEXTs are in the catalog on GitHub, with which EEZ Studio synchronizes its catalog every time it is started. Synchronization with the IEXT catalog can also be started manually at any time using the <i>Upgrade Catalog</i> option. The <i>Install extension</i> option allows installing an IEXT that is not in the catalog (from local storage).
3	<i>Versions</i>	IEXT can have multiple versions. If there is more than one, it is possible to change the installed IEXT with one of the versions from the list. In this case, the Replace option will appear as in Fig. 28.



Fig. 28: 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.

I1.7. 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.

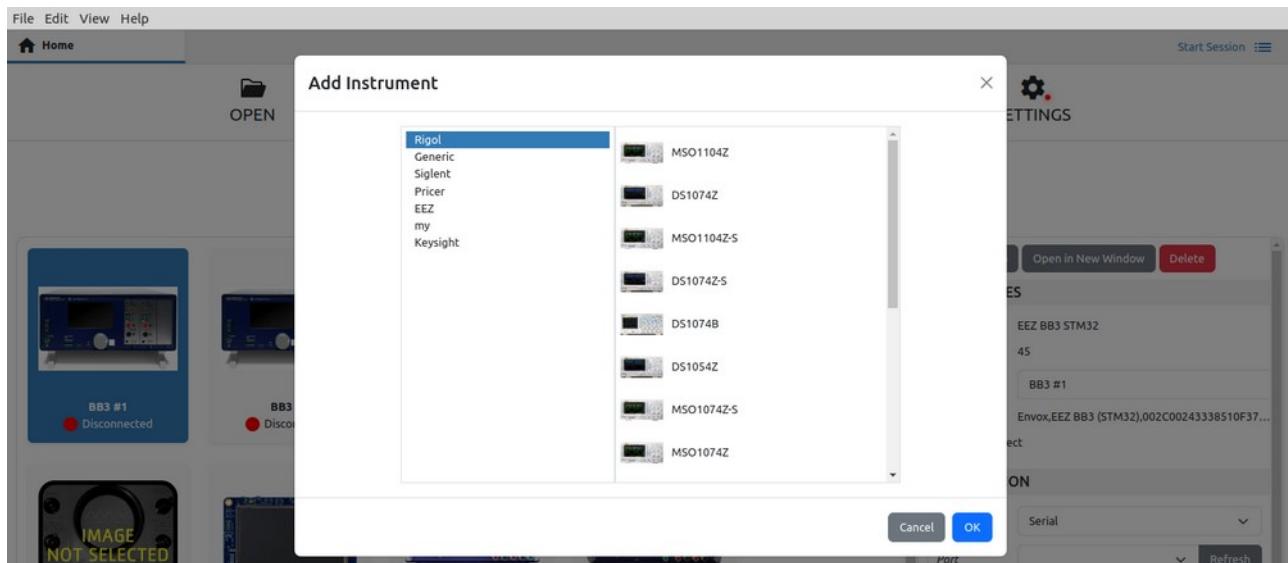


Fig. 29: Add instrument to workbench

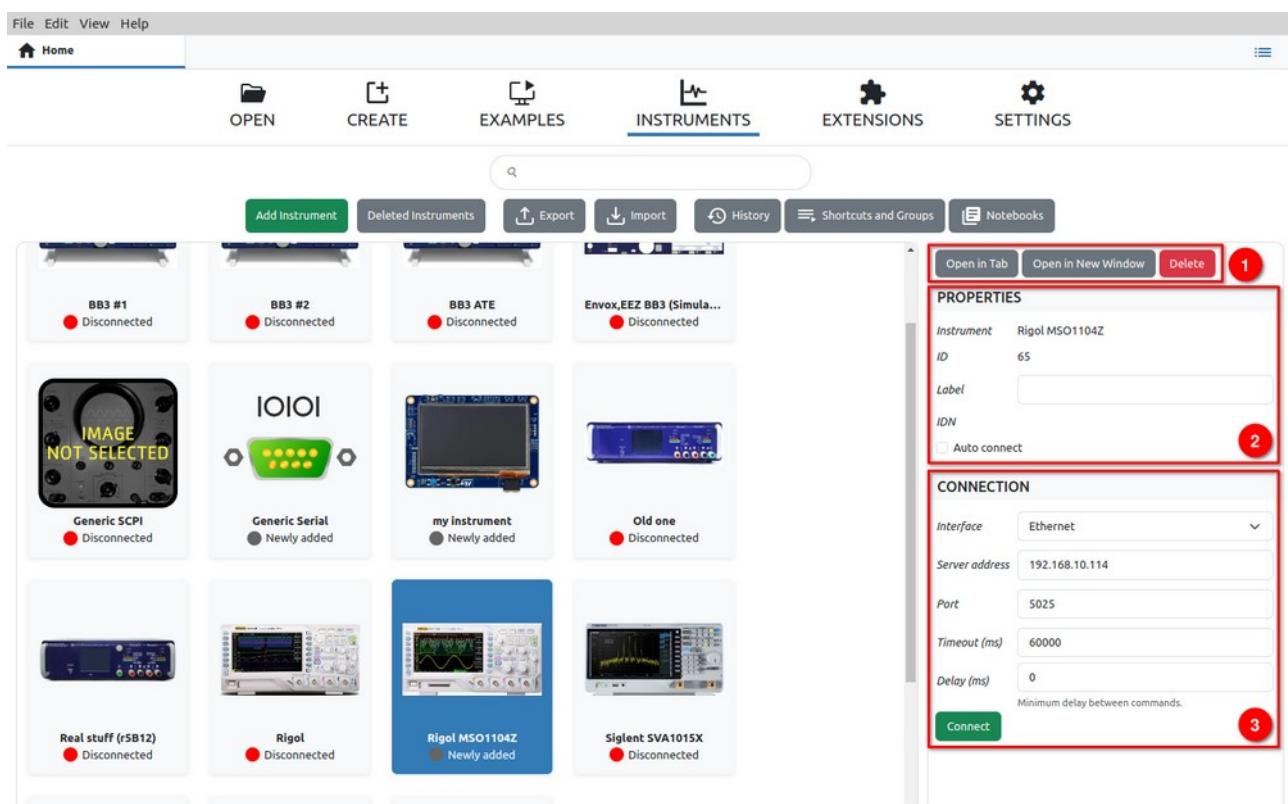


Fig. 30: New added instrument

A successfully added instrument will appear on the workbench (Fig. 30) with the label *Newly added*, and when selected, the sidebar will have the following sections:

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

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

For normal communication via the VISA interface, it will be necessary to install a free [R&S®VISA](#) driver. In case it is not installed or there is some problem in communication with it, an error message will appear as in Fig. 31.

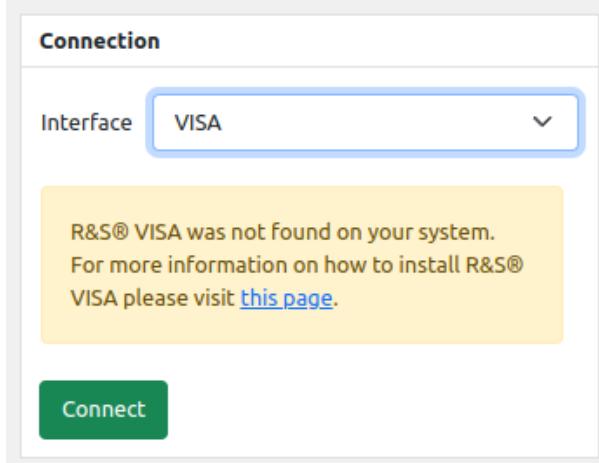


Fig. 31: VISA driver error message

I1.8. Establishing a connection with the instrument

Connection to the instrument added to the workbench will be possible as shown in Fig. 32: 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. 33 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.

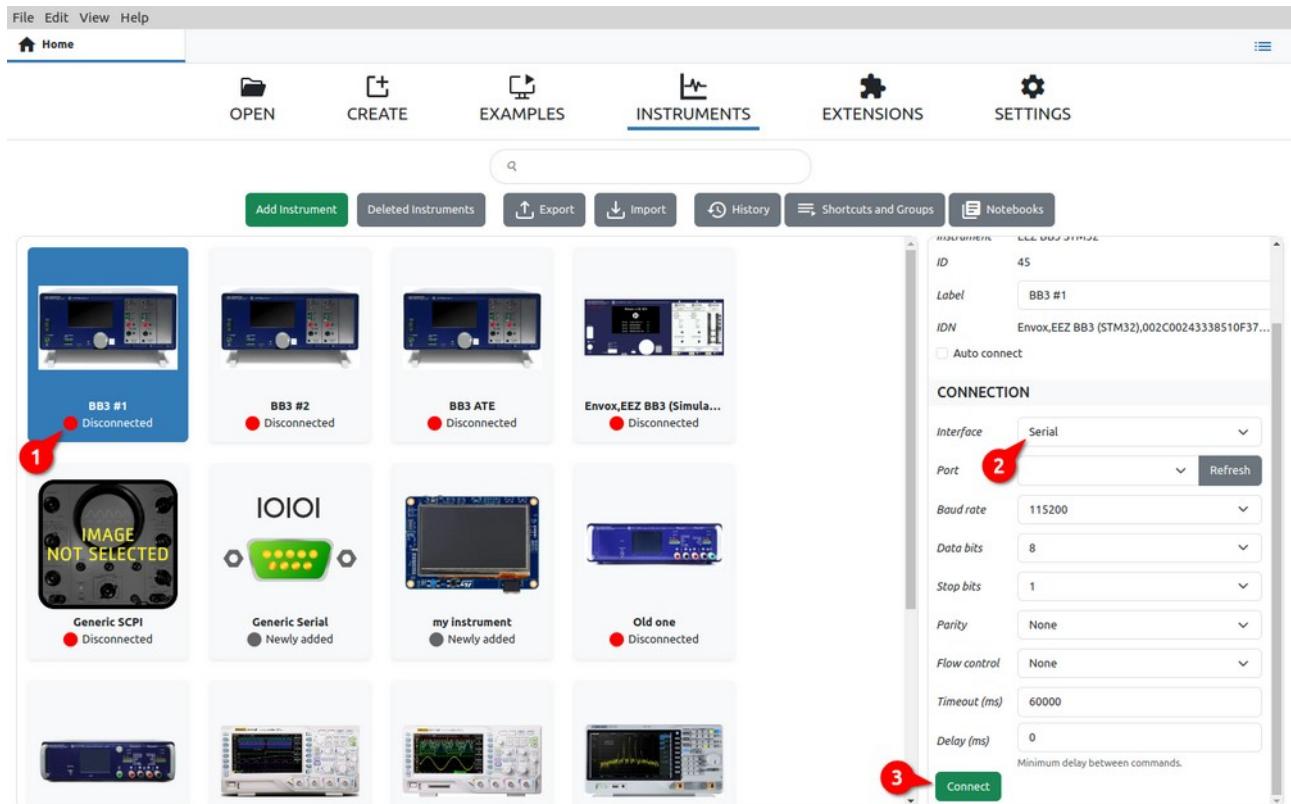


Fig. 32: Selecting an instrument on the workbench to establish a connection

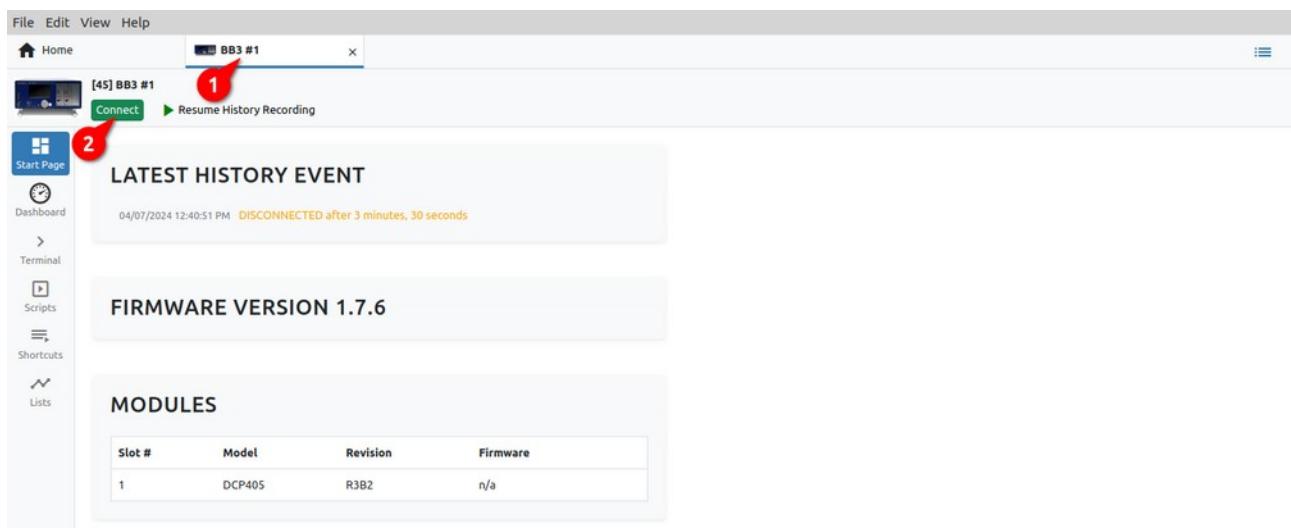


Fig. 33: Positioning on the instrument tab and establishing a connection

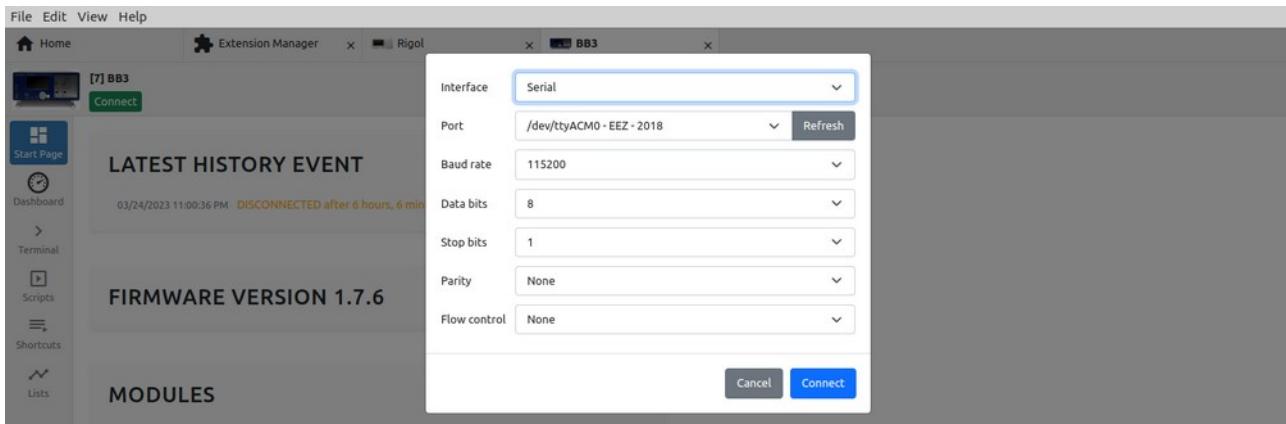


Fig. 34: Interface selection for instrument connection

Once the connection is established, it will be possible to close the connection by selecting the Disconnect button (Fig. 35).

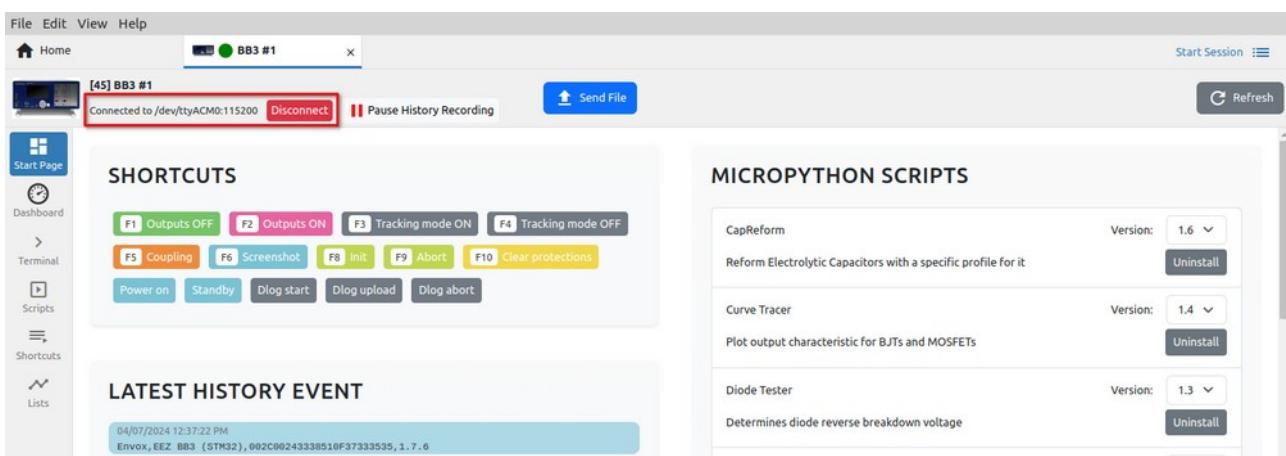


Fig. 35: Option to close the connection

11.9. Export

The *Export* feature enables the export and archiving of data from the currently active database. It is possible to export data belonging to one or more instruments or sessions.

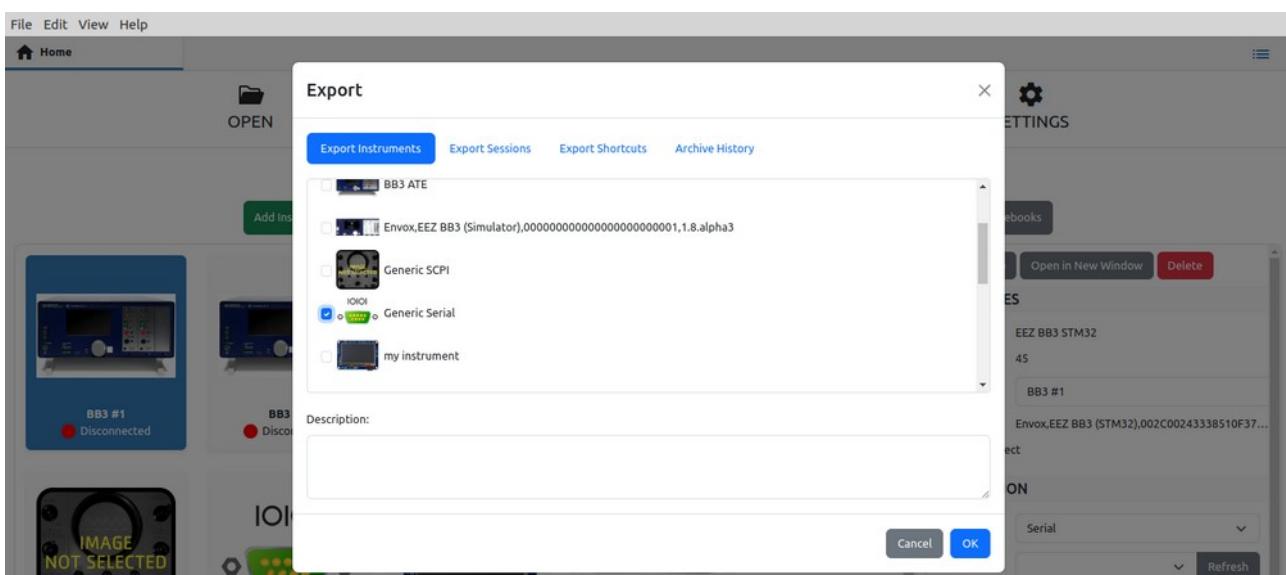


Fig. 36: Export of selected instruments to a new database file

When exporting, it is necessary to fill in the *Description* field with the text that will be seen during import. In Fig. 36 shows an example of data export of one instrument, and Fig. 31 is an example of ex-

11. Home page instrument sections

porting multiple sessions.

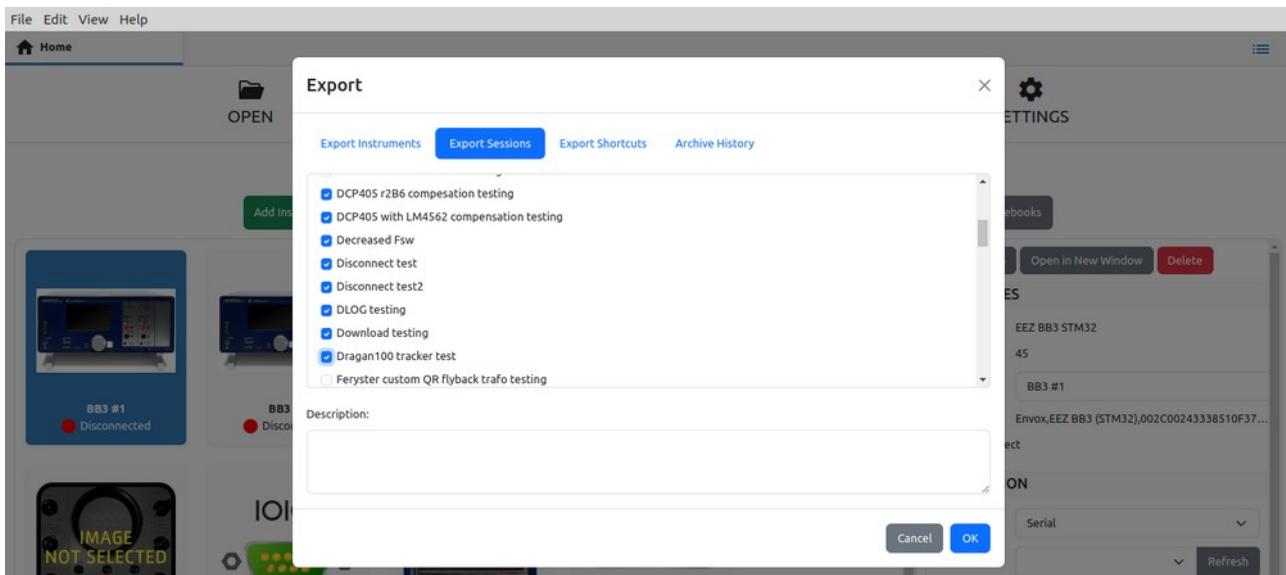


Fig. 37: Export of selected sessions to a new database file

Export Shortcuts tab allows selection from the collection of shortcuts of installed instruments (Fig. 38).

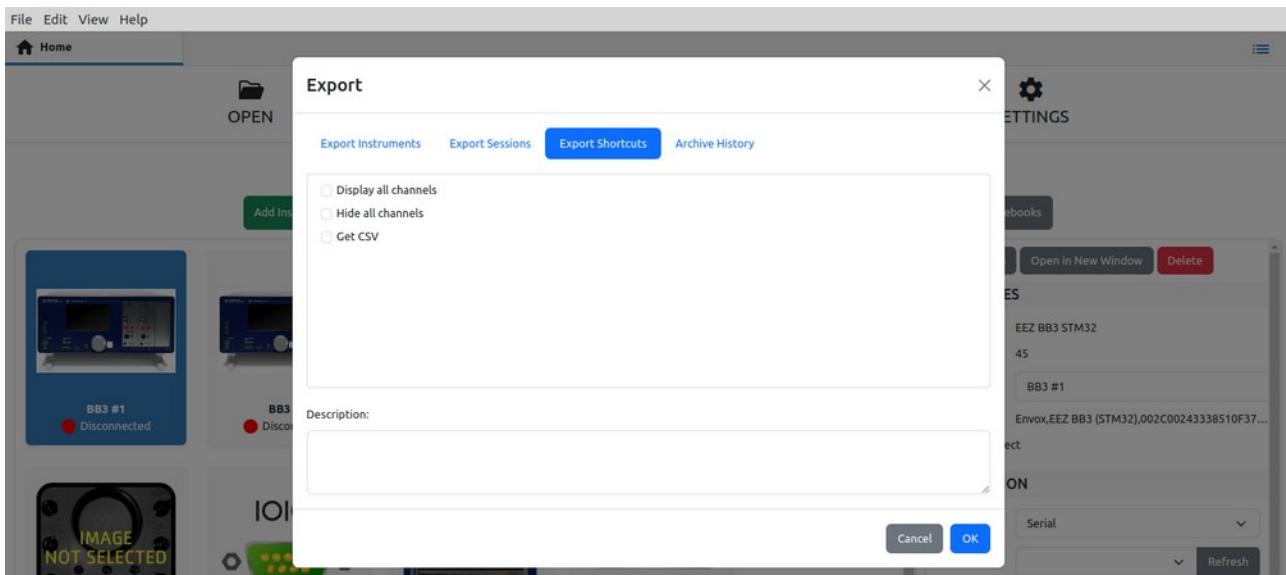


Fig. 38: Export selected shortcuts

When archiving, it is possible to choose whether the data will be deleted from the database after the archive. In Fig. 39 shows an example of archiving for the last 6 months.

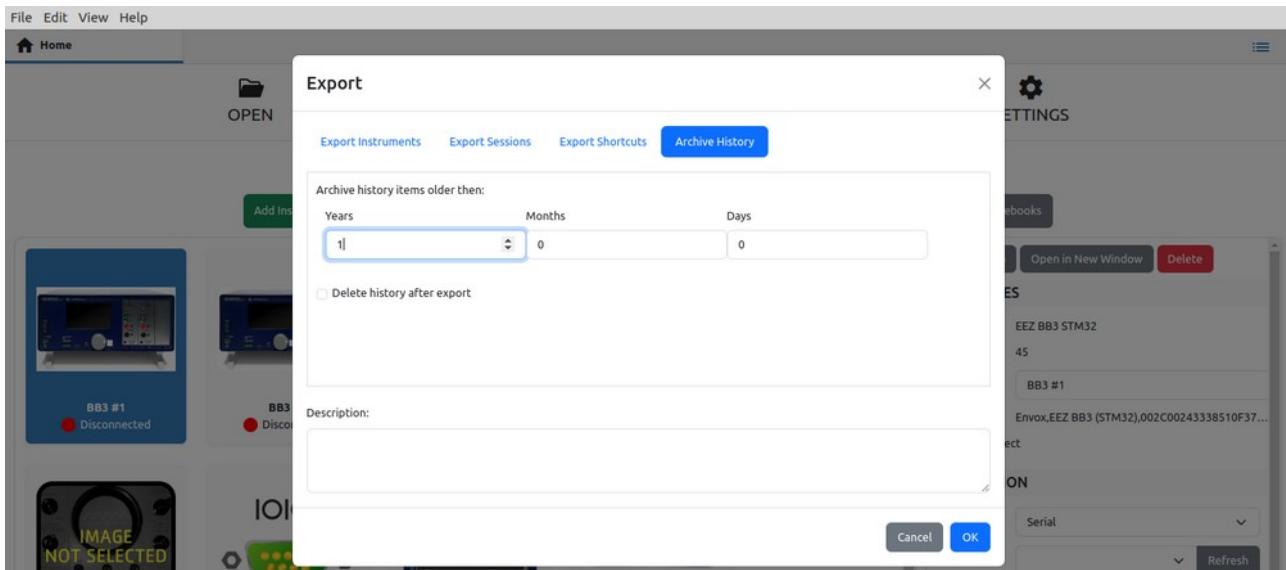


Fig. 39: Data archiving for a selected period of time

11.10. Import

The *Import* feature enables the import of data that we have previously exported or archived. It can also be data that we received from a third party.

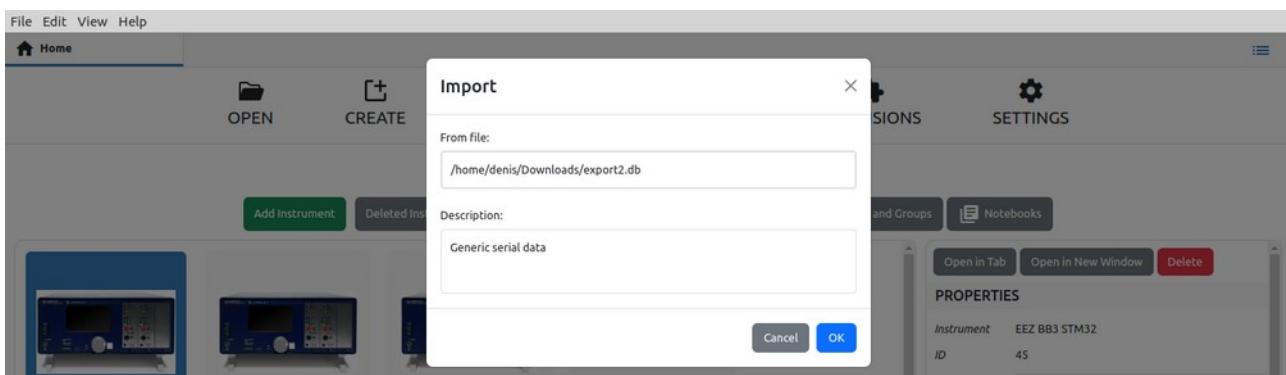


Fig. 40: Data import into the active database

By selecting this option, a dialog box for selecting the `.db` file will first be opened, and if the format is good, another dialog box will be displayed as shown in Fig. 40 for import confirmation.

12. 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.

12.1. Start page (EEZ BB3 only)

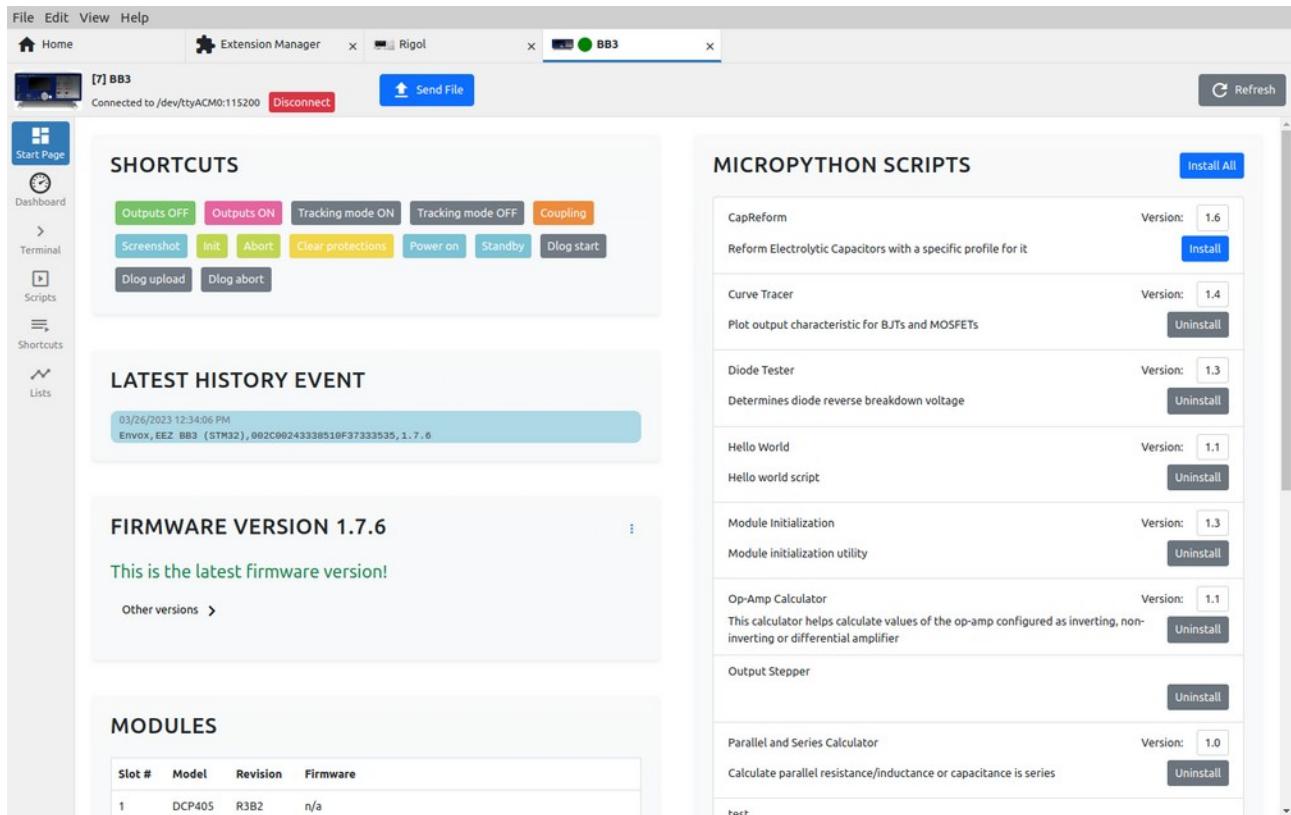


Fig. 41: EEZ BB3 start page

Section / option

Send file

Description

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

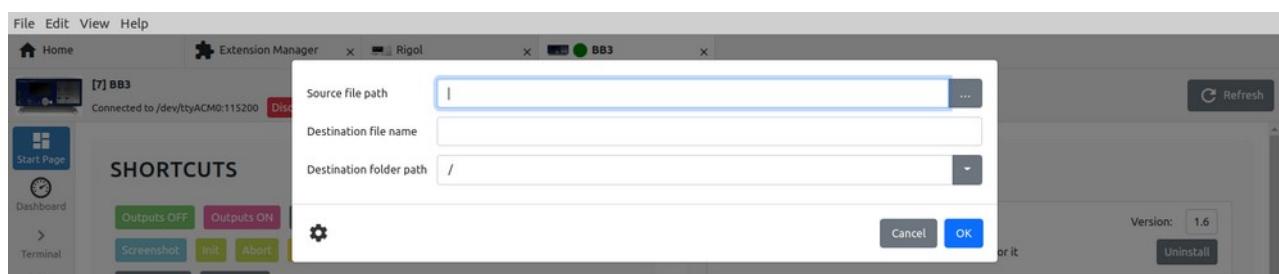


Fig. 42: Sending a file to EEZ BB3

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. 43.

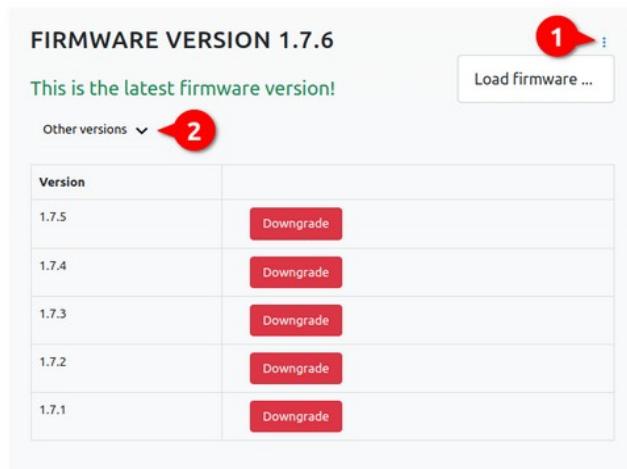


Fig. 43: EEZ BB3 firmware version section

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.

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

Fig. 44: 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 I2.5), and which are located on EEZ BB3. Lists can be downloaded, uploaded and edited.

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

Fig. 45: EEZ BB3 program lists

12.2. Dashboard

Fig. 46 shows an example Dashboard that enables simple operations with EEZ BB3 modules.

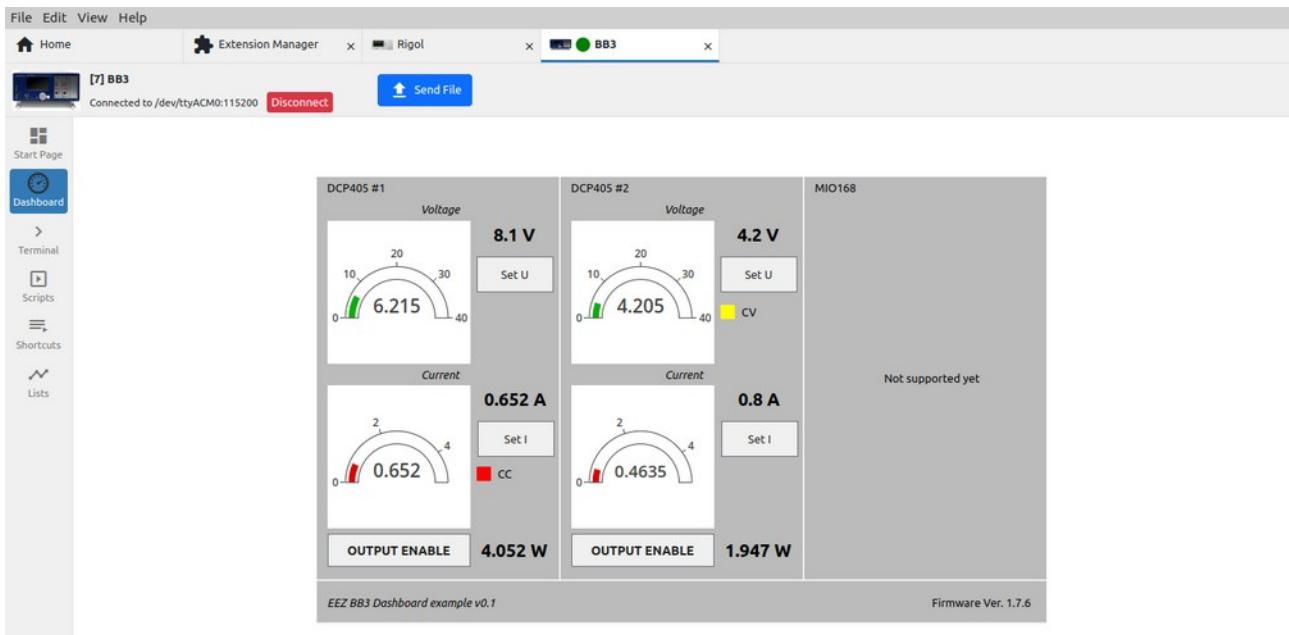


Fig. 46: Instrument dashboard example

12.3. Terminal (SCPI protocol)

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).

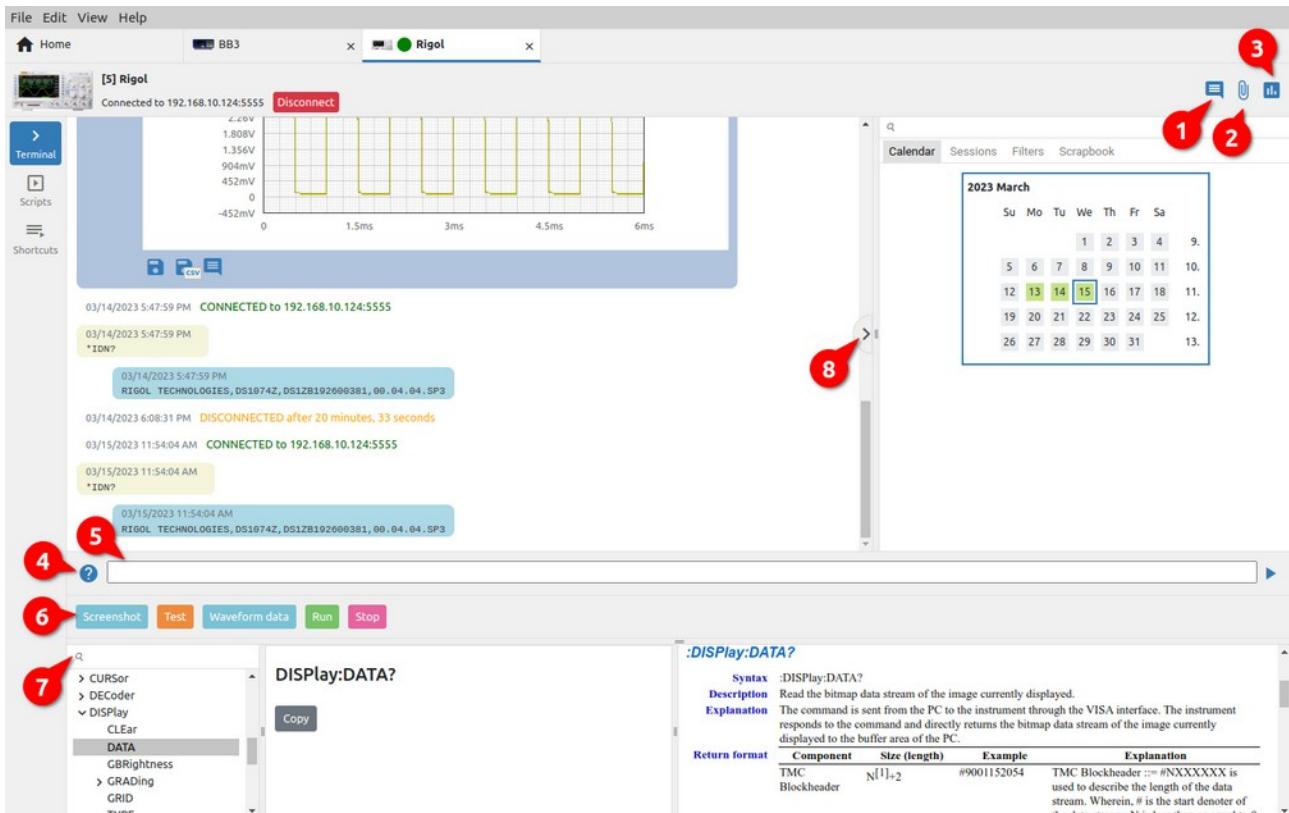


Fig. 47: Terminal section of SCPI instrument

Option

1 Add note

Description

Adds a note at the cursor position. If the note is not added to the end of the conversation, it will receive the timestamp of the item on which the cursor was placed (see Fig. 49).



Fig. 48: Place the cursor at the position where the notes will be added

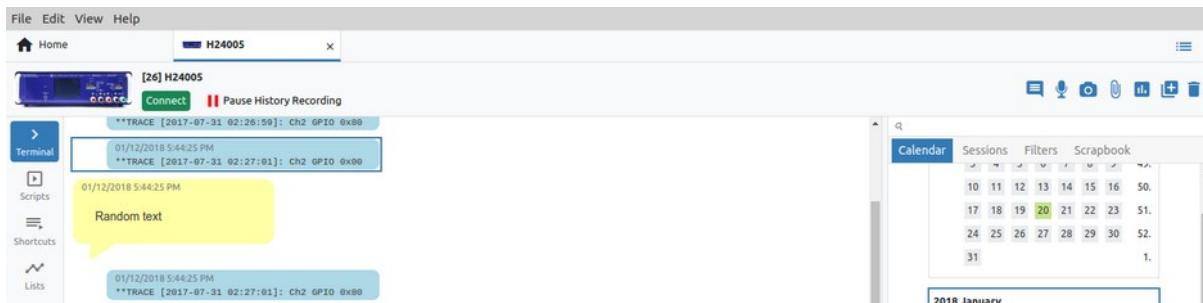


Fig. 49: Showing the newly added note

2 Attach file

Attaching a file (see Section I1.3).

3 Add chart

Creating a new chart from two or more charts (see Section I1.3).

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.

12.4. Terminal (Proprietary protocol)

For instruments that do not support communication with the SCPI protocol, it is possible to use IEXT, which implements a proprietary protocol that does not have strict definitions of the format of the data to be exchanged.

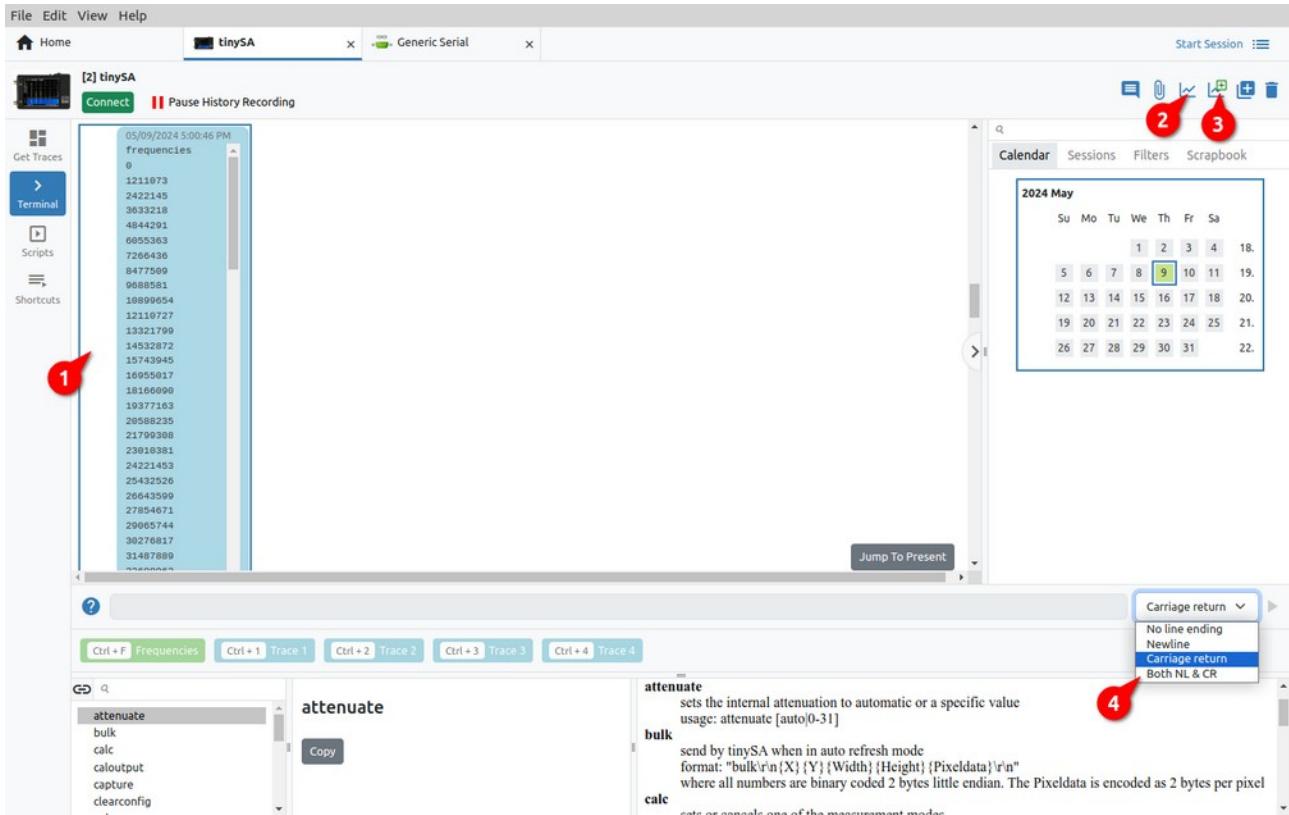


Fig. 50: Terminal section of proprietary protocol instrument

The terminal for this type of instrument (i.e. IEXT) has several additional options that are not found in the terminal for the SCPI instrument.

The terminal allows receiving a continuous stream of data that will be displayed as (1) in Fig. 50. We can subsequently convert this data into a graph using *Create chart from selected items* option (3). Whether the graph will start drawing immediately as new data arrives depends on the status of the *Show/Hide Plotter* option (2). If graph plotting is active and the data has not yet started arriving, a message will appear in the terminal as shown in Fig. 51.

EEZ Studio Reference Guide



Fig. 51: Waiting for data to plot the graph

Finally, the instrument terminal with proprietary protocol offers four different end of line sequences (item 4, Fig. 50) depending on what the instrument supports.

12.5. Scripts

The figure shows the EEZ Studio interface with a script editor open. The editor has several numbered callouts: 1 points to the "Edit Shortcut" button; 2 points to the "+" button; 3 points to the "Edit Shortcut" dialog box; 4 points to the "Run" button; 5 points to the "Show Terminal" button; and 6 points to the "Search and replace" bar. The script code is as follows:

```

1 //<Preamble>
2 //<format>, <type>, <points>, <count>, <xincrement>, <xorigin>, <xreference>, <yincrement>, <yorigin>, <yreference>
3 //<type>: 0 (BYTE), 1 (WORD) or 2 (ASC),
4 //<points>: an integer between 1 and 128000
5 //<xincrement>: the time difference between two neighboring points in the X direction.
6 //<xorigin>: the start time of the waveform data in the X direction.
7 //<xreference>: the reference time of the data point in the X direction.
8 //<yincrement>: the vertical offset relative to the "Vertical Reference Position" in the Y direction.
9 //<yorigin>: the vertical reference position in the Y direction.
10
11 const CHUNK_MAX_POINTS = 1000000;
12 const NUM_CHANNELS = instrument.properties.channels.length;
13
14 connection.acquire();
15
16 var displayedChannels = [];
17
18 for (var i=channel = 1; i<channel <= NUM_CHANNELS; ++i) {
19     var channel = await connection.query(`CHANnel${i}:DISPLAY?`);
20     if (channel) {
21         displayedChannels.push(channel);
22     }
23 }
24
25 if (displayedChannels.length > 0) {
26     connection.command("ACQuire:DEPth auto");
27     connection.command("WAveform:MODE raw");
28     connection.command("STOP");
29     connection.command("WAveform:FORMAT byte");
30
31     var samplingRate = await connection.query("ACQuire:SRATE?");
32     var timeScale = await connection.query("TImebase:SCALE?");
33
34     var chunk_points = Math.floor(CHUNK_MAX_POINTS / displayedChannels.length);
35
36     for (var iDisplayedChannel = 0; iDisplayedChannel < displayedChannels.length; ++iDisplayedChannel) {
37         var channel = displayedChannels[iDisplayedChannel];
38
39         connection.command(`WAveform:SOURce ${channel}`);
40
41         var channelScale = await connection.query(`CHANnel${i}:SCALE?`);
42         var channelOffset = await connection.query(`CHANnel${i}:OFFSET?`);
43
44         var unit = await connection.query(`CHANnel${i}:UNIT?`);
45         var color = instrument.properties.channels[i].color;
46         var colorInverse = instrument.properties.channels[i].colorInverse;
47         var label = `CHANnel ${i}`;
48
49         var preamble = await connection.query("WAveform:PREamble");
50         var preambleArray = preamble.split(',');
51
52         var numPoints = parseInt(preambleArray[0]);
53         var yinc = parseFloat(preambleArray[1]);
54         var yor = parseFloat(preambleArray[2]);
55         var yref = parseFloat(preambleArray[3]);
56
57         var data = new Uint8Array(chunk_points * numPoints);
58
59         for (var j=0; j<chunk_points; ++j) {
60             for (var k=0; k<numPoints; ++k) {
61                 data[j * numPoints + k] = ...
62             }
63         }
64
65         var dataString = data.toString('hex');
66
67         var dataObject = {
68             "X": 0,
69             "Y": yor,
70             "Label": label,
71             "Color": color,
72             "Scale": channelScale,
73             "Offset": channelOffset,
74             "Unit": unit,
75             "Data": dataString
76         };
77
78         displayedChannels[i].push(dataObject);
79     }
80
81     connection.command("STOP");
82
83     var waveform = await connection.query("WAveform:DATA?");
84     var waveformArray = waveform.split(',');
85
86     for (var i=0; i<displayedChannels.length; ++i) {
87         var channel = displayedChannels[i];
88
89         for (var j=0; j<channel.length; ++j) {
90             var dataObject = channel[j];
91
92             dataObject["X"] = dataObject["X"] * timeScale;
93             dataObject["Y"] = dataObject["Y"] + yinc * j;
94
95             if (dataObject["Label"] === null) {
96                 dataObject["Label"] = label;
97             }
98
99             if (dataObject["Color"] === null) {
100                dataObject["Color"] = color;
101            }
102
103             if (dataObject["Scale"] === null) {
104                 dataObject["Scale"] = channelScale;
105             }
106
107             if (dataObject["Offset"] === null) {
108                 dataObject["Offset"] = channelOffset;
109             }
110
111             if (dataObject["Unit"] === null) {
112                 dataObject["Unit"] = unit;
113             }
114
115             if (dataObject["Data"] === null) {
116                 dataObject["Data"] = waveformArray[j];
117             }
118
119             if (dataObject["Label"] === null) {
120                 dataObject["Label"] = label;
121             }
122
123             if (dataObject["Color"] === null) {
124                 dataObject["Color"] = color;
125             }
126
127             if (dataObject["Scale"] === null) {
128                 dataObject["Scale"] = channelScale;
129             }
130
131             if (dataObject["Offset"] === null) {
132                 dataObject["Offset"] = channelOffset;
133             }
134
135             if (dataObject["Unit"] === null) {
136                 dataObject["Unit"] = unit;
137             }
138
139             if (dataObject["Data"] === null) {
140                 dataObject["Data"] = waveformArray[j];
141             }
142
143             if (dataObject["Label"] === null) {
144                 dataObject["Label"] = label;
145             }
146
147             if (dataObject["Color"] === null) {
148                 dataObject["Color"] = color;
149             }
150
151             if (dataObject["Scale"] === null) {
152                 dataObject["Scale"] = channelScale;
153             }
154
155             if (dataObject["Offset"] === null) {
156                 dataObject["Offset"] = channelOffset;
157             }
158
159             if (dataObject["Unit"] === null) {
160                 dataObject["Unit"] = unit;
161             }
162
163             if (dataObject["Data"] === null) {
164                 dataObject["Data"] = waveformArray[j];
165             }
166
167             if (dataObject["Label"] === null) {
168                 dataObject["Label"] = label;
169             }
170
171             if (dataObject["Color"] === null) {
172                 dataObject["Color"] = color;
173             }
174
175             if (dataObject["Scale"] === null) {
176                 dataObject["Scale"] = channelScale;
177             }
178
179             if (dataObject["Offset"] === null) {
180                 dataObject["Offset"] = channelOffset;
181             }
182
183             if (dataObject["Unit"] === null) {
184                 dataObject["Unit"] = unit;
185             }
186
187             if (dataObject["Data"] === null) {
188                 dataObject["Data"] = waveformArray[j];
189             }
190
191             if (dataObject["Label"] === null) {
192                 dataObject["Label"] = label;
193             }
194
195             if (dataObject["Color"] === null) {
196                 dataObject["Color"] = color;
197             }
198
199             if (dataObject["Scale"] === null) {
200                 dataObject["Scale"] = channelScale;
201             }
202
203             if (dataObject["Offset"] === null) {
204                 dataObject["Offset"] = channelOffset;
205             }
206
207             if (dataObject["Unit"] === null) {
208                 dataObject["Unit"] = unit;
209             }
210
211             if (dataObject["Data"] === null) {
212                 dataObject["Data"] = waveformArray[j];
213             }
214
215             if (dataObject["Label"] === null) {
216                 dataObject["Label"] = label;
217             }
218
219             if (dataObject["Color"] === null) {
220                 dataObject["Color"] = color;
221             }
222
223             if (dataObject["Scale"] === null) {
224                 dataObject["Scale"] = channelScale;
225             }
226
227             if (dataObject["Offset"] === null) {
228                 dataObject["Offset"] = channelOffset;
229             }
230
231             if (dataObject["Unit"] === null) {
232                 dataObject["Unit"] = unit;
233             }
234
235             if (dataObject["Data"] === null) {
236                 dataObject["Data"] = waveformArray[j];
237             }
238
239             if (dataObject["Label"] === null) {
240                 dataObject["Label"] = label;
241             }
242
243             if (dataObject["Color"] === null) {
244                 dataObject["Color"] = color;
245             }
246
247             if (dataObject["Scale"] === null) {
248                 dataObject["Scale"] = channelScale;
249             }
250
251             if (dataObject["Offset"] === null) {
252                 dataObject["Offset"] = channelOffset;
253             }
254
255             if (dataObject["Unit"] === null) {
256                 dataObject["Unit"] = unit;
257             }
258
259             if (dataObject["Data"] === null) {
260                 dataObject["Data"] = waveformArray[j];
261             }
262
263             if (dataObject["Label"] === null) {
264                 dataObject["Label"] = label;
265             }
266
267             if (dataObject["Color"] === null) {
268                 dataObject["Color"] = color;
269             }
270
271             if (dataObject["Scale"] === null) {
272                 dataObject["Scale"] = channelScale;
273             }
274
275             if (dataObject["Offset"] === null) {
276                 dataObject["Offset"] = channelOffset;
277             }
278
279             if (dataObject["Unit"] === null) {
280                 dataObject["Unit"] = unit;
281             }
282
283             if (dataObject["Data"] === null) {
284                 dataObject["Data"] = waveformArray[j];
285             }
286
287             if (dataObject["Label"] === null) {
288                 dataObject["Label"] = label;
289             }
290
291             if (dataObject["Color"] === null) {
292                 dataObject["Color"] = color;
293             }
294
295             if (dataObject["Scale"] === null) {
296                 dataObject["Scale"] = channelScale;
297             }
298
299             if (dataObject["Offset"] === null) {
300                 dataObject["Offset"] = channelOffset;
301             }
302
303             if (dataObject["Unit"] === null) {
304                 dataObject["Unit"] = unit;
305             }
306
307             if (dataObject["Data"] === null) {
308                 dataObject["Data"] = waveformArray[j];
309             }
310
311             if (dataObject["Label"] === null) {
312                 dataObject["Label"] = label;
313             }
314
315             if (dataObject["Color"] === null) {
316                 dataObject["Color"] = color;
317             }
318
319             if (dataObject["Scale"] === null) {
320                 dataObject["Scale"] = channelScale;
321             }
322
323             if (dataObject["Offset"] === null) {
324                 dataObject["Offset"] = channelOffset;
325             }
326
327             if (dataObject["Unit"] === null) {
328                 dataObject["Unit"] = unit;
329             }
330
331             if (dataObject["Data"] === null) {
332                 dataObject["Data"] = waveformArray[j];
333             }
334
335             if (dataObject["Label"] === null) {
336                 dataObject["Label"] = label;
337             }
338
339             if (dataObject["Color"] === null) {
340                 dataObject["Color"] = color;
341             }
342
343             if (dataObject["Scale"] === null) {
344                 dataObject["Scale"] = channelScale;
345             }
346
347             if (dataObject["Offset"] === null) {
348                 dataObject["Offset"] = channelOffset;
349             }
350
351             if (dataObject["Unit"] === null) {
352                 dataObject["Unit"] = unit;
353             }
354
355             if (dataObject["Data"] === null) {
356                 dataObject["Data"] = waveformArray[j];
357             }
358
359             if (dataObject["Label"] === null) {
360                 dataObject["Label"] = label;
361             }
362
363             if (dataObject["Color"] === null) {
364                 dataObject["Color"] = color;
365             }
366
367             if (dataObject["Scale"] === null) {
368                 dataObject["Scale"] = channelScale;
369             }
370
371             if (dataObject["Offset"] === null) {
372                 dataObject["Offset"] = channelOffset;
373             }
374
375             if (dataObject["Unit"] === null) {
376                 dataObject["Unit"] = unit;
377             }
378
379             if (dataObject["Data"] === null) {
380                 dataObject["Data"] = waveformArray[j];
381             }
382
383             if (dataObject["Label"] === null) {
384                 dataObject["Label"] = label;
385             }
386
387             if (dataObject["Color"] === null) {
388                 dataObject["Color"] = color;
389             }
390
391             if (dataObject["Scale"] === null) {
392                 dataObject["Scale"] = channelScale;
393             }
394
395             if (dataObject["Offset"] === null) {
396                 dataObject["Offset"] = channelOffset;
397             }
398
399             if (dataObject["Unit"] === null) {
400                 dataObject["Unit"] = unit;
401             }
402
403             if (dataObject["Data"] === null) {
404                 dataObject["Data"] = waveformArray[j];
405             }
406
407             if (dataObject["Label"] === null) {
408                 dataObject["Label"] = label;
409             }
410
411             if (dataObject["Color"] === null) {
412                 dataObject["Color"] = color;
413             }
414
415             if (dataObject["Scale"] === null) {
416                 dataObject["Scale"] = channelScale;
417             }
418
419             if (dataObject["Offset"] === null) {
420                 dataObject["Offset"] = channelOffset;
421             }
422
423             if (dataObject["Unit"] === null) {
424                 dataObject["Unit"] = unit;
425             }
426
427             if (dataObject["Data"] === null) {
428                 dataObject["Data"] = waveformArray[j];
429             }
430
431             if (dataObject["Label"] === null) {
432                 dataObject["Label"] = label;
433             }
434
435             if (dataObject["Color"] === null) {
436                 dataObject["Color"] = color;
437             }
438
439             if (dataObject["Scale"] === null) {
440                 dataObject["Scale"] = channelScale;
441             }
442
443             if (dataObject["Offset"] === null) {
444                 dataObject["Offset"] = channelOffset;
445             }
446
447             if (dataObject["Unit"] === null) {
448                 dataObject["Unit"] = unit;
449             }
450
451             if (dataObject["Data"] === null) {
452                 dataObject["Data"] = waveformArray[j];
453             }
454
455             if (dataObject["Label"] === null) {
456                 dataObject["Label"] = label;
457             }
458
459             if (dataObject["Color"] === null) {
460                 dataObject["Color"] = color;
461             }
462
463             if (dataObject["Scale"] === null) {
464                 dataObject["Scale"] = channelScale;
465             }
466
467             if (dataObject["Offset"] === null) {
468                 dataObject["Offset"] = channelOffset;
469             }
470
471             if (dataObject["Unit"] === null) {
472                 dataObject["Unit"] = unit;
473             }
474
475             if (dataObject["Data"] === null) {
476                 dataObject["Data"] = waveformArray[j];
477             }
478
479             if (dataObject["Label"] === null) {
480                 dataObject["Label"] = label;
481             }
482
483             if (dataObject["Color"] === null) {
484                 dataObject["Color"] = color;
485             }
486
487             if (dataObject["Scale"] === null) {
488                 dataObject["Scale"] = channelScale;
489             }
490
491             if (dataObject["Offset"] === null) {
492                 dataObject["Offset"] = channelOffset;
493             }
494
495             if (dataObject["Unit"] === null) {
496                 dataObject["Unit"] = unit;
497             }
498
499             if (dataObject["Data"] === null) {
500                 dataObject["Data"] = waveformArray[j];
501             }
502
503             if (dataObject["Label"] === null) {
504                 dataObject["Label"] = label;
505             }
506
507             if (dataObject["Color"] === null) {
508                 dataObject["Color"] = color;
509             }
510
511             if (dataObject["Scale"] === null) {
512                 dataObject["Scale"] = channelScale;
513             }
514
515             if (dataObject["Offset"] === null) {
516                 dataObject["Offset"] = channelOffset;
517             }
518
519             if (dataObject["Unit"] === null) {
520                 dataObject["Unit"] = unit;
521             }
522
523             if (dataObject["Data"] === null) {
524                 dataObject["Data"] = waveformArray[j];
525             }
526
527             if (dataObject["Label"] === null) {
528                 dataObject["Label"] = label;
529             }
530
531             if (dataObject["Color"] === null) {
532                 dataObject["Color"] = color;
533             }
534
535             if (dataObject["Scale"] === null) {
536                 dataObject["Scale"] = channelScale;
537             }
538
539             if (dataObject["Offset"] === null) {
540                 dataObject["Offset"] = channelOffset;
541             }
542
543             if (dataObject["Unit"] === null) {
544                 dataObject["Unit"] = unit;
545             }
546
547             if (dataObject["Data"] === null) {
548                 dataObject["Data"] = waveformArray[j];
549             }
550
551             if (dataObject["Label"] === null) {
552                 dataObject["Label"] = label;
553             }
554
555             if (dataObject["Color"] === null) {
556                 dataObject["Color"] = color;
557             }
558
559             if (dataObject["Scale"] === null) {
560                 dataObject["Scale"] = channelScale;
561             }
562
563             if (dataObject["Offset"] === null) {
564                 dataObject["Offset"] = channelOffset;
565             }
566
567             if (dataObject["Unit"] === null) {
568                 dataObject["Unit"] = unit;
569             }
570
571             if (dataObject["Data"] === null) {
572                 dataObject["Data"] = waveformArray[j];
573             }
574
575             if (dataObject["Label"] === null) {
576                 dataObject["Label"] = label;
577             }
578
579             if (dataObject["Color"] === null) {
580                 dataObject["Color"] = color;
581             }
582
583             if (dataObject["Scale"] === null) {
584                 dataObject["Scale"] = channelScale;
585             }
586
587             if (dataObject["Offset"] === null) {
588                 dataObject["Offset"] = channelOffset;
589             }
590
591             if (dataObject["Unit"] === null) {
592                 dataObject["Unit"] = unit;
593             }
594
595             if (dataObject["Data"] === null) {
596                 dataObject["Data"] = waveformArray[j];
597             }
598
599             if (dataObject["Label"] === null) {
600                 dataObject["Label"] = label;
601             }
602
603             if (dataObject["Color"] === null) {
604                 dataObject["Color"] = color;
605             }
606
607             if (dataObject["Scale"] === null) {
608                 dataObject["Scale"] = channelScale;
609             }
610
611             if (dataObject["Offset"] === null) {
612                 dataObject["Offset"] = channelOffset;
613             }
614
615             if (dataObject["Unit"] === null) {
616                 dataObject["Unit"] = unit;
617             }
618
619             if (dataObject["Data"] === null) {
620                 dataObject["Data"] = waveformArray[j];
621             }
622
623             if (dataObject["Label"] === null) {
624                 dataObject["Label"] = label;
625             }
626
627             if (dataObject["Color"] === null) {
628                 dataObject["Color"] = color;
629             }
630
631             if (dataObject["Scale"] === null) {
632                 dataObject["Scale"] = channelScale;
633             }
634
635             if (dataObject["Offset"] === null) {
636                 dataObject["Offset"] = channelOffset;
637             }
638
639             if (dataObject["Unit"] === null) {
640                 dataObject["Unit"] = unit;
641             }
642
643             if (dataObject["Data"] === null) {
644                 dataObject["Data"] = waveformArray[j];
645             }
646
647             if (dataObject["Label"] === null) {
648                 dataObject["Label"] = label;
649             }
650
651             if (dataObject["Color"] === null) {
652                 dataObject["Color"] = color;
653             }
654
655             if (dataObject["Scale"] === null) {
656                 dataObject["Scale"] = channelScale;
657             }
658
659             if (dataObject["Offset"] === null) {
660                 dataObject["Offset"] = channelOffset;
661             }
662
663             if (dataObject["Unit"] === null) {
664                 dataObject["Unit"] = unit;
665             }
666
667             if (dataObject["Data"] === null) {
668                 dataObject["Data"] = waveformArray[j];
669             }
670
671             if (dataObject["Label"] === null) {
672                 dataObject["Label"] = label;
673             }
674
675             if (dataObject["Color"] === null) {
676                 dataObject["Color"] = color;
677             }
678
679             if (dataObject["Scale"] === null) {
680                 dataObject["Scale"] = channelScale;
681             }
682
683             if (dataObject["Offset"] === null) {
684                 dataObject["Offset"] = channelOffset;
685             }
686
687             if (dataObject["Unit"] === null) {
688                 dataObject["Unit"] = unit;
689             }
690
691             if (dataObject["Data"] === null) {
692                 dataObject["Data"] = waveformArray[j];
693             }
694
695             if (dataObject["Label"] === null) {
696                 dataObject["Label"] = label;
697             }
698
699             if (dataObject["Color"] === null) {
700                 dataObject["Color"] = color;
701             }
702
703             if (dataObject["Scale"] === null) {
704                 dataObject["Scale"] = channelScale;
705             }
706
707             if (dataObject["Offset"] === null) {
708                 dataObject["Offset"] = channelOffset;
709             }
710
711             if (dataObject["Unit"] === null) {
712                 dataObject["Unit"] = unit;
713             }
714
715             if (dataObject["Data"] === null) {
716                 dataObject["Data"] = waveformArray[j];
717             }
718
719             if (dataObject["Label"] === null) {
720                 dataObject["Label"] = label;
721             }
722
723             if (dataObject["Color"] === null) {
724                 dataObject["Color"] = color;
725             }
726
727             if (dataObject["Scale"] === null) {
728                 dataObject["Scale"] = channelScale;
729             }
730
731             if (dataObject["Offset"] === null) {
732                 dataObject["Offset"] = channelOffset;
733             }
734
735             if (dataObject["Unit"] === null) {
736                 dataObject["Unit"] = unit;
737             }
738
739             if (dataObject["Data"] === null) {
740                 dataObject["Data"] = waveformArray[j];
741             }
742
743             if (dataObject["Label"] === null) {
744                 dataObject["Label"] = label;
745             }
746
747             if (dataObject["Color"] === null) {
748                 dataObject["Color"] = color;
749             }
750
751             if (dataObject["Scale"] === null) {
752                 dataObject["Scale"] = channelScale;
753             }
754
755             if (dataObject["Offset"] === null) {
756                 dataObject["Offset"] = channelOffset;
757             }
758
759             if (dataObject["Unit"] === null) {
760                 dataObject["Unit"] = unit;
761             }
762
763             if (dataObject["Data"] === null) {
764                 dataObject["Data"] = waveformArray[j];
765             }
766
767             if (dataObject["Label"] === null) {
768                 dataObject["Label"] = label;
769             }
770
771             if (dataObject["Color"] === null) {
772                 dataObject["Color"] = color;
773             }
774
775             if (dataObject["Scale"] === null) {
776                 dataObject["Scale"] = channelScale;
777             }
778
779             if (dataObject["Offset"] === null) {
780                 dataObject["Offset"] = channelOffset;
781             }
782
783             if (dataObject["Unit"] === null) {
784                 dataObject["Unit"] = unit;
785             }
786
787             if (dataObject["Data"] === null) {
788                 dataObject["Data"] = waveformArray[j];
789             }
790
791             if (dataObject["Label"] === null) {
792                 dataObject["Label"] = label;
793             }
794
795             if (dataObject["Color"] === null) {
796                 dataObject["Color"] = color;
797             }
798
799             if (dataObject["Scale"] === null) {
800                 dataObject["Scale"] = channelScale;
801             }
802
803             if (dataObject["Offset"] === null) {
804                 dataObject["Offset"] = channelOffset;
805             }
806
807             if (dataObject["Unit"] === null) {
808                 dataObject["Unit"] = unit;
809             }
810
811             if (dataObject["Data"] === null) {
812                 dataObject["Data"] = waveformArray[j];
813             }
814
815             if (dataObject["Label"] === null) {
816                 dataObject["Label"] = label;
817             }
818
819             if (dataObject["Color"] === null) {
820                 dataObject["Color"] = color;
821             }
822
823             if (dataObject["Scale"] === null) {
824                 dataObject["Scale"] = channelScale;
825             }
826
827             if (dataObject["Offset"] === null) {
828                 dataObject["Offset"] = channelOffset;
829             }
830
831             if (dataObject["Unit"] === null) {
832                 dataObject["Unit"] = unit;
833             }
834
835             if (dataObject["Data"] === null) {
836                 dataObject["Data"] = waveformArray[j];
837             }
838
839             if (dataObject["Label"] === null) {
840                 dataObject["Label"] = label;
841             }
842
843             if (dataObject["Color"] === null) {
844                 dataObject["Color"] = color;
845             }
846
847             if (dataObject["Scale"] === null) {
848                 dataObject["Scale"] = channelScale;
849             }
850
851             if (dataObject["Offset"] === null) {
852                 dataObject["Offset"] = channelOffset;
853             }
854
855             if (dataObject["Unit"] === null) {
856                 dataObject["Unit"] = unit;
857             }
858
859             if (dataObject["Data"] === null) {
860                 dataObject["Data"] = waveformArray[j];
861             }
862
863             if (dataObject["Label"] === null) {
864                 dataObject["Label"] = label;
865             }
866
867             if (dataObject["Color"] === null) {
868                 dataObject["Color"] = color;
869             }
870
871             if (dataObject["Scale"] === null) {
872                 dataObject["Scale"] = channelScale;
873             }
874
875             if (dataObject["Offset"] === null) {
876                 dataObject["Offset"] = channelOffset;
877             }
878
879             if (dataObject["Unit"] === null) {
880                 dataObject["Unit"] = unit;
881             }
882
883             if (dataObject["Data"] === null) {
884                 dataObject["Data"] = waveformArray[j];
885             }
886
887             if (dataObject["Label"] === null) {
888                 dataObject["Label"] = label;
889             }
890
891             if (dataObject["Color"] === null) {
892                 dataObject["Color"] = color;
893             }
894
895             if (dataObject["Scale"] === null) {
896                 dataObject["Scale"] = channelScale;
897             }
898
899             if (dataObject["Offset"] === null) {
900                 dataObject["Offset"] = channelOffset;
901             }
902
903             if (dataObject["Unit"] === null) {
904                 dataObject["Unit"] = unit;
905             }
906
907             if (dataObject["Data"] === null) {
908                 dataObject["Data"] = waveformArray[j];
909             }
910
911             if (dataObject["Label"] === null) {
912                 dataObject["Label"] = label;
913             }
914
915             if (dataObject["Color"] === null) {
916                 dataObject["Color"] = color;
917             }
918
919             if (dataObject["Scale"] === null) {
920                 dataObject["Scale"] = channelScale;
921             }
922
923             if (dataObject["Offset"] === null) {
924                 dataObject["Offset"] = channelOffset;
925             }
926
927             if (dataObject["Unit"] === null) {
928                 dataObject["Unit"] = unit;
929             }
930
931             if (dataObject["Data"] === null) {
932                 dataObject["Data"] = waveformArray[j];
933             }
934
935             if (dataObject["Label"] === null) {
936                 dataObject["Label"] = label;
937             }
938
939             if (dataObject["Color"] === null) {
940                 dataObject["Color"] = color;
941             }
942
943             if (dataObject["Scale"] === null) {
944                 dataObject["Scale"] = channelScale;
945             }
946
947             if (dataObject["Offset"] === null) {
948                 dataObject["Offset"] = channelOffset;
949             }
950
951             if (dataObject["Unit"] === null) {
952                 dataObject["Unit"] = unit;
953             }
954
955             if (dataObject["Data"] === null) {
956                 dataObject["Data"] = waveformArray[j];
957             }
958
959             if (dataObject["Label"] === null) {
960                 dataObject["Label"] = label;
961             }
962
963             if (dataObject["Color"] === null) {
964                 dataObject["Color"] = color;
965             }
966
967             if (dataObject["Scale"] === null) {
968                 dataObject["Scale"] = channelScale;
969             }
970
971             if (dataObject["Offset"] === null) {
972                 dataObject["Offset"] = channelOffset;
973             }
974
975             if (dataObject["Unit"] === null) {
976                 dataObject["Unit"] = unit;
977             }
978
979             if (dataObject["Data"] === null) {
980                 dataObject["Data"] = waveformArray[j];
981             }
982
983             if (dataObject["Label"] === null) {
984                 dataObject["Label"] = label;
985             }
986
987             if (dataObject["Color"] === null) {
988                 dataObject["Color"] = color;
989             }
990
991             if (dataObject["Scale"] === null) {
992                 dataObject["Scale"] = channelScale;
993             }
994
995             if (dataObject["Offset"] === null) {
996                 dataObject["Offset"] = channelOffset;
997             }
998
999             if (dataObject["Unit"] === null) {
1000                dataObject["Unit"] = unit;
1001            }
1002
1003             if (dataObject["Data"] === null) {
1004                 dataObject["Data"] = waveformArray[j];
1005             }
1006
1007             if (dataObject["Label"] === null) {
1008                 dataObject["Label"] = label;
1009             }
1010
1011             if (dataObject["Color"] === null) {
1012                 dataObject["Color"] = color;
1013             }
1014
1015             if (dataObject["Scale"] === null) {
1016                 dataObject["Scale"] = channelScale;
1017             }
1018
1019             if (dataObject["Offset"] === null) {
1020                 dataObject["Offset"] = channelOffset;
1021             }
1022
1023             if (dataObject["Unit"] === null) {
1024                 dataObject["Unit"] = unit;
1025             }
1026
1027             if (dataObject["Data"] === null) {
1028                 dataObject["Data"] = waveformArray[j];
1029             }
1030
1031             if (dataObject["Label"] === null) {
1032                 dataObject["Label"] = label;
1033             }
1034
1035             if (dataObject["Color"] === null) {
1036                 dataObject["Color"] = color;
1037             }
1038
1039             if (dataObject["Scale"] === null) {
1040                 dataObject["Scale"] = channelScale;
1041             }
1042
1043             if (dataObject["Offset"] === null) {
1044                 dataObject["Offset"] = channelOffset;
1045             }
1046
1047             if (dataObject["Unit"] === null) {
1048                 dataObject["Unit"] = unit;
1049             }
1050
1051             if (dataObject["Data"] === null) {
1052                 dataObject["Data"] = waveformArray[j];
1053             }
1054
1055             if (dataObject["Label"] === null) {
1056                 dataObject["Label"] = label;
1057             }
1058
1059             if (dataObject["Color"] === null) {
1060                 dataObject["Color"] = color;
1061             }
1062
1063             if (dataObject["Scale"] === null) {
1064                 dataObject["Scale"] = channelScale;
1065             }
1066
1067             if (dataObject["Offset"] === null) {
1068                 dataObject["Offset"] = channelOffset;
1069             }
1070
1071             if (dataObject["Unit"] === null) {
1072                 dataObject["Unit"] = unit;
1073             }
1074
1075             if (dataObject["Data"] === null) {
1076                 dataObject["Data"] = waveformArray[j];
1077             }
1078
1079             if (dataObject["Label"] === null) {
1080                 dataObject["Label"] = label;
1081             }
1082
1083             if (dataObject["Color"] === null) {
1084                 dataObject["Color"] = color;
1085             }
1086
1087             if (dataObject["Scale"] === null) {
1088                 dataObject["Scale"] = channelScale;
1089             }
1090
1091             if (dataObject["Offset"] === null) {
1092                 dataObject["Offset"] = channelOffset;
1093             }
1094
1095             if (dataObject["Unit"] === null) {
1096                 dataObject["Unit"] = unit;
1097             }
1098
1099             if (dataObject["Data"] === null) {
1100                dataObject["Data"] = waveformArray[j];
1101            }
1102
1103             if (dataObject["Label"] === null) {
1104                 dataObject["Label"] = label;
1105             }
1106
1107             if (dataObject["Color"] === null) {
1108                 dataObject["Color"] = color;
1109             }
1110
1111             if (dataObject["Scale"] === null) {
1112                 dataObject["Scale"] = channelScale;
1113             }
1114
1
```

I2. Instrument activity bar

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.).

#	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).

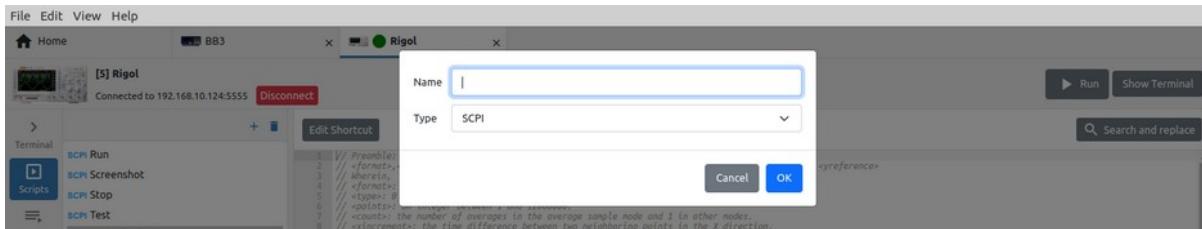


Fig. 53: Adding a new script

The content of the script is entered in the editor (Fig. 54).

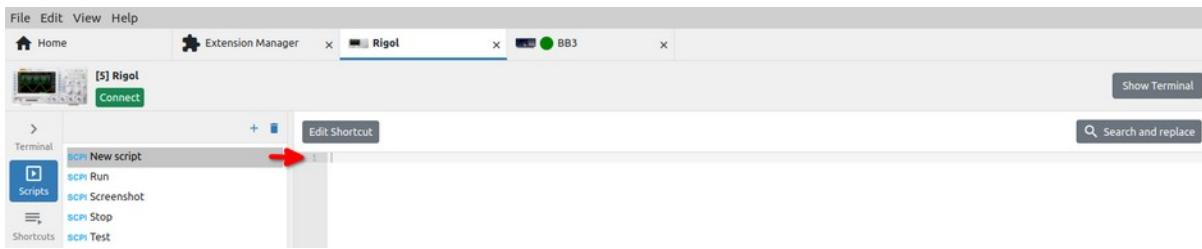


Fig. 54: Script editing

2	Delete script	Deleting the selected script.
3	Edit shortcut	Editing a script shortcut (see Section I2.5.1)
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.

I2.5.1. Edit script shortcut

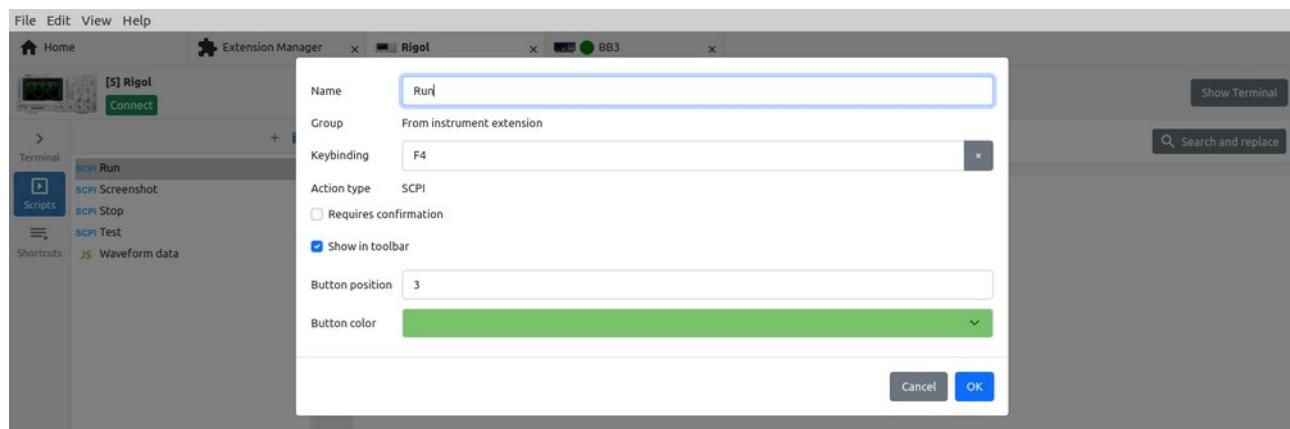


Fig. 55: Script shortcut editing

Option	Description
Name	The name of the script shortcut as it will be displayed in the shortcut bar.
Group	The name of the group to which the shortcut belongs. If the shortcut is defined in IEXT, the label <i>From instrument extension</i> will be displayed.
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 <i>Terminal's Shortcuts bar</i> .
Button position	The position of the shortcut button in the <i>Shortcuts bar</i> . When displaying, the shortcut with a lower value will be displayed first. If there are multiple shortcuts with the same value, they will be sorted alphabetically.
Button color	Color coding of shortcut button.

I2.6. Shortcuts

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

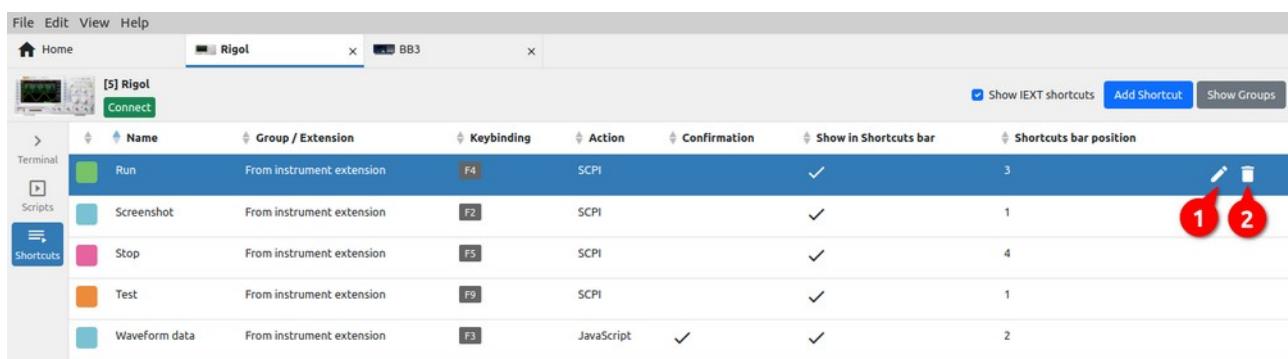


Fig. 56: Instrument shortcuts

#	Option	Description
1	Edit shortcut	Editing the shortcut (see Section I2.5.1)
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. 57.

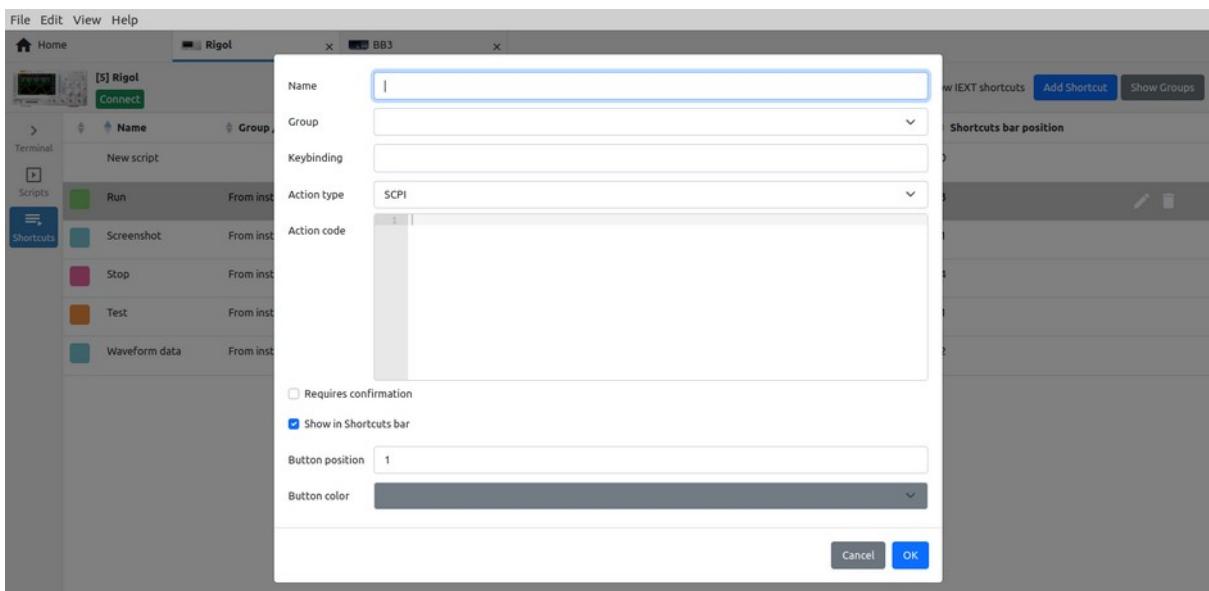


Fig. 57: Add new shortcut

Show Groups / Show Shortcuts

Toggle between displaying a list of shortcuts and groups (Fig. 58) of shortcuts.

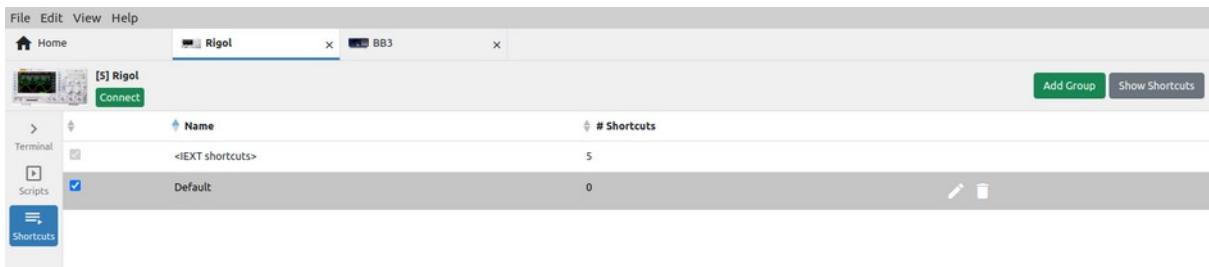


Fig. 58: Instrument shortcut groups

12.7. 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.

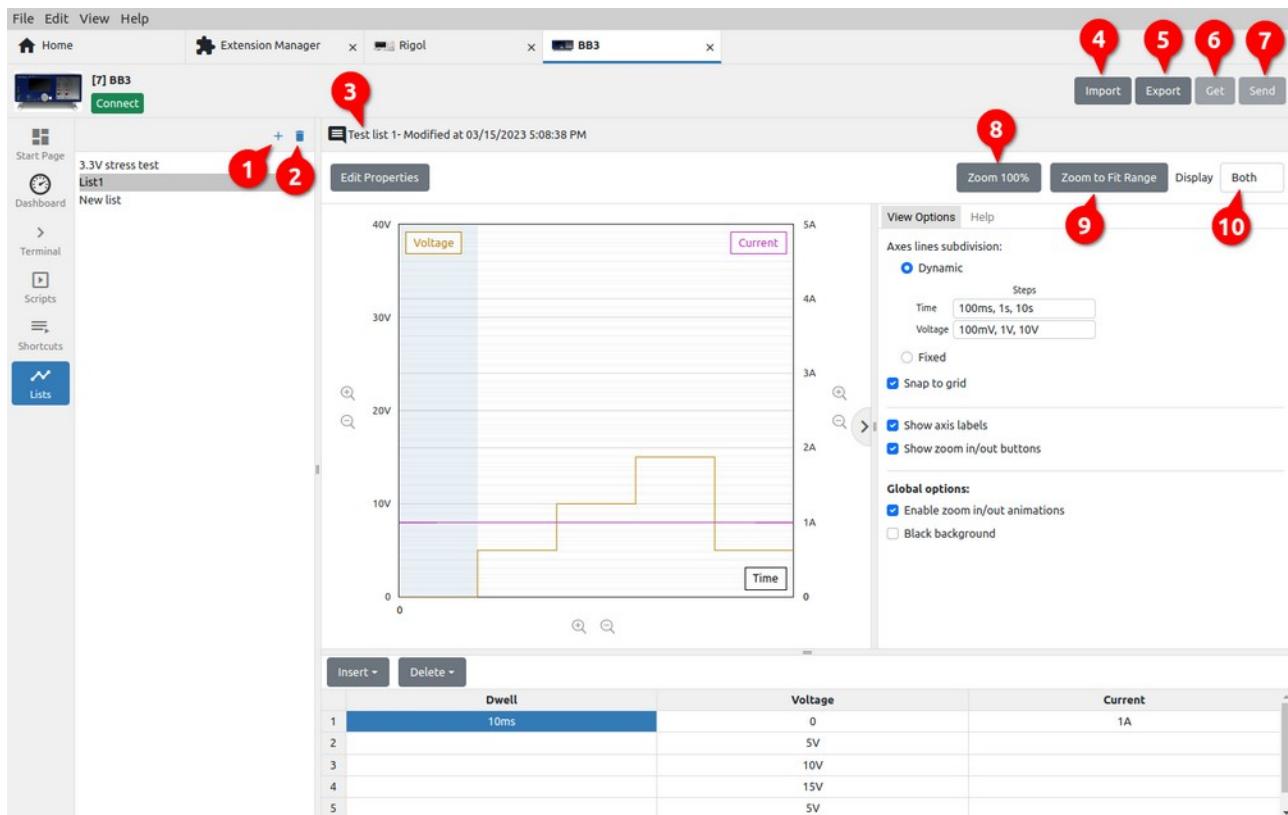


Fig. 59: Instrument programming lists

Option

1 Add list

Description

Creating a new list. The parameters of the list can be specified through a table (Fig. 59) 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. 60) 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. 61).

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

12. Instrument activity bar

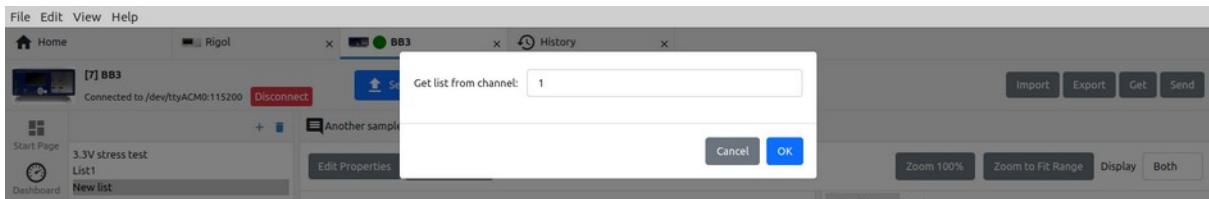


Fig. 60: List source selection

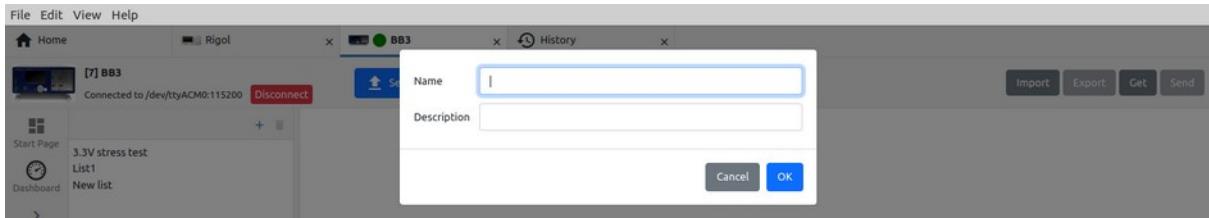


Fig. 61: Imported list parameters

- | | |
|----------------------------|---|
| 7 <i>Send</i> | Sending the list to the instrument. The option will be disabled if connection is not established with the instrument. |
| 8 <i>Zoom 100%</i> | Display graph without scaling. |
| 9 <i>Zoom to Fit Range</i> | Graph display scaled according to the largest defined value. |
| 10 <i>Display</i> | Selection of graphs to be displayed (e.g. voltage only, current only, both). |

12.7.1. Editing a list using a table

Editing the list via the table is shown in Fig. 59. 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. 62 and Fig. 63 shows all options for inserting new lines and deleting existing ones.

Insert	Delete	Dwell	Voltage	Current
Insert row above		10ms	0	1A
Insert row below			5V	2A
3			10V	1A
4			15V	1A
5			5V	2A
6				

Fig. 62: Table insertion options

Insert	Delete	Voltage	Current
1	Delete row	0	1A
2	Clear column from cursor down	5V	2A
3	Delete all from cursor down	10V	1A
4	Delete all	15V	1A
5		5V	2A
6			

Fig. 63: Table deletion options

12.7.2. 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.

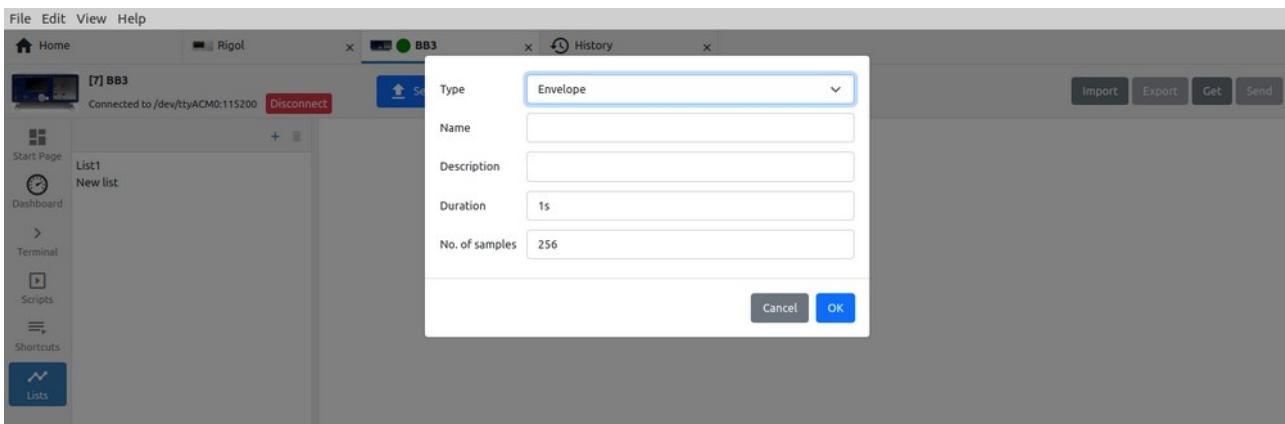


Fig. 64: Adding a new list in envelope mode

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. 64). 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.

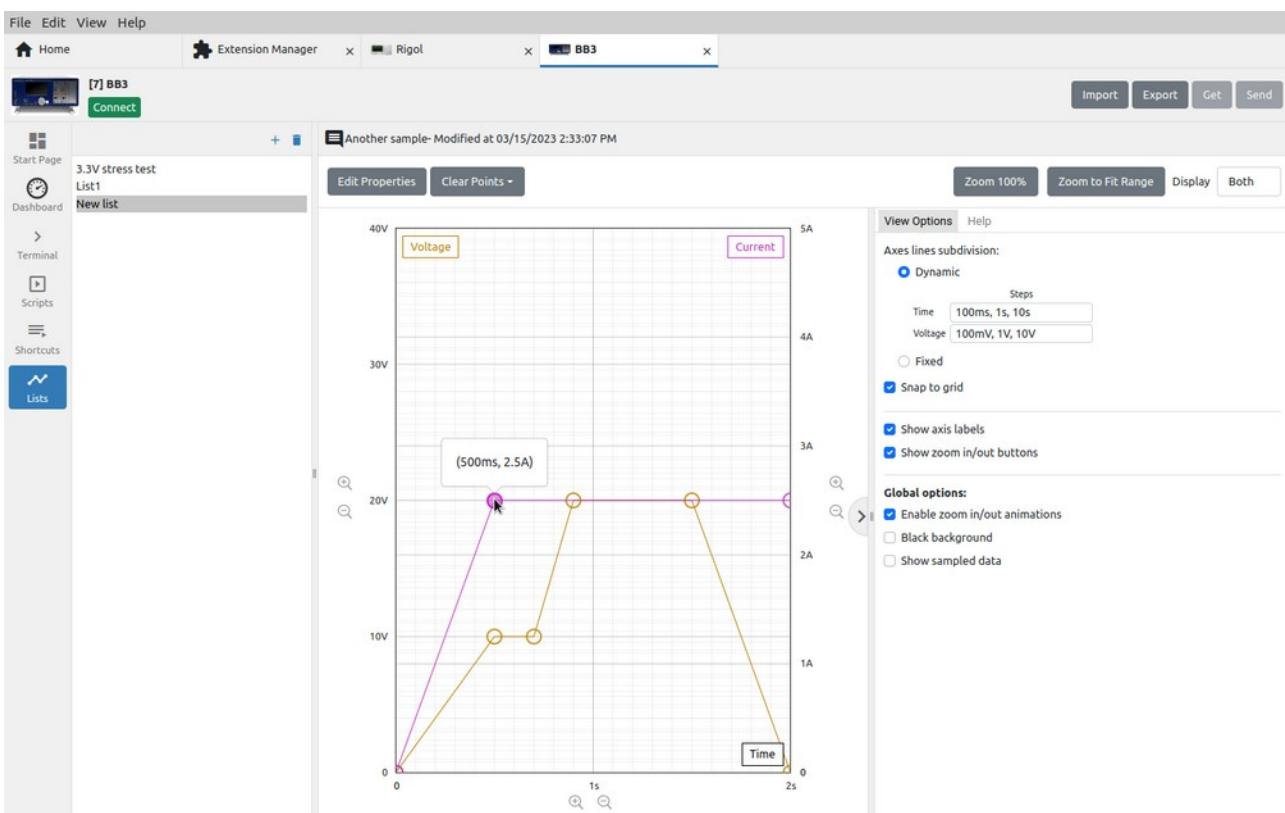


Fig. 65: Graph editing in envelope mode

The example in Fig. 65 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.

12. Instrument activity bar

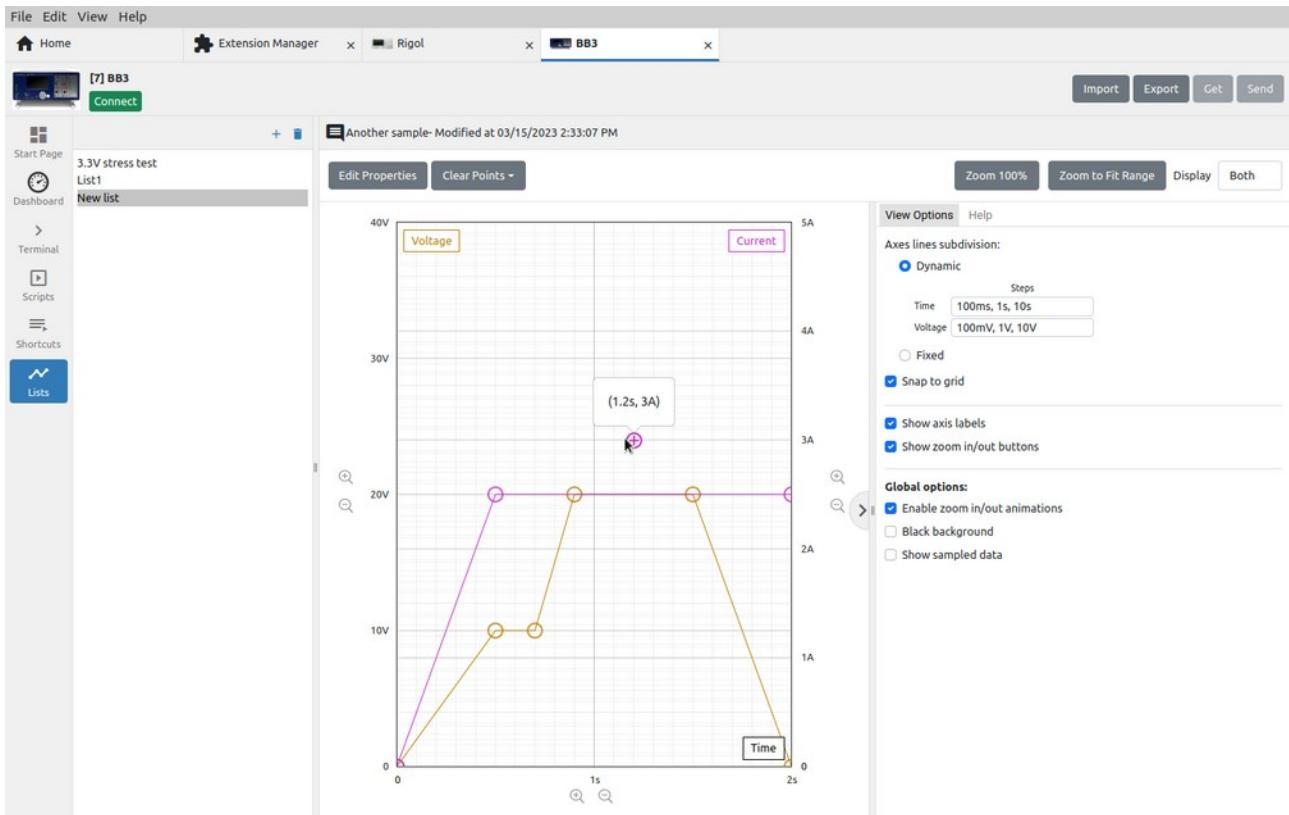


Fig. 66: Adding a new point in envelope mode

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. 67.

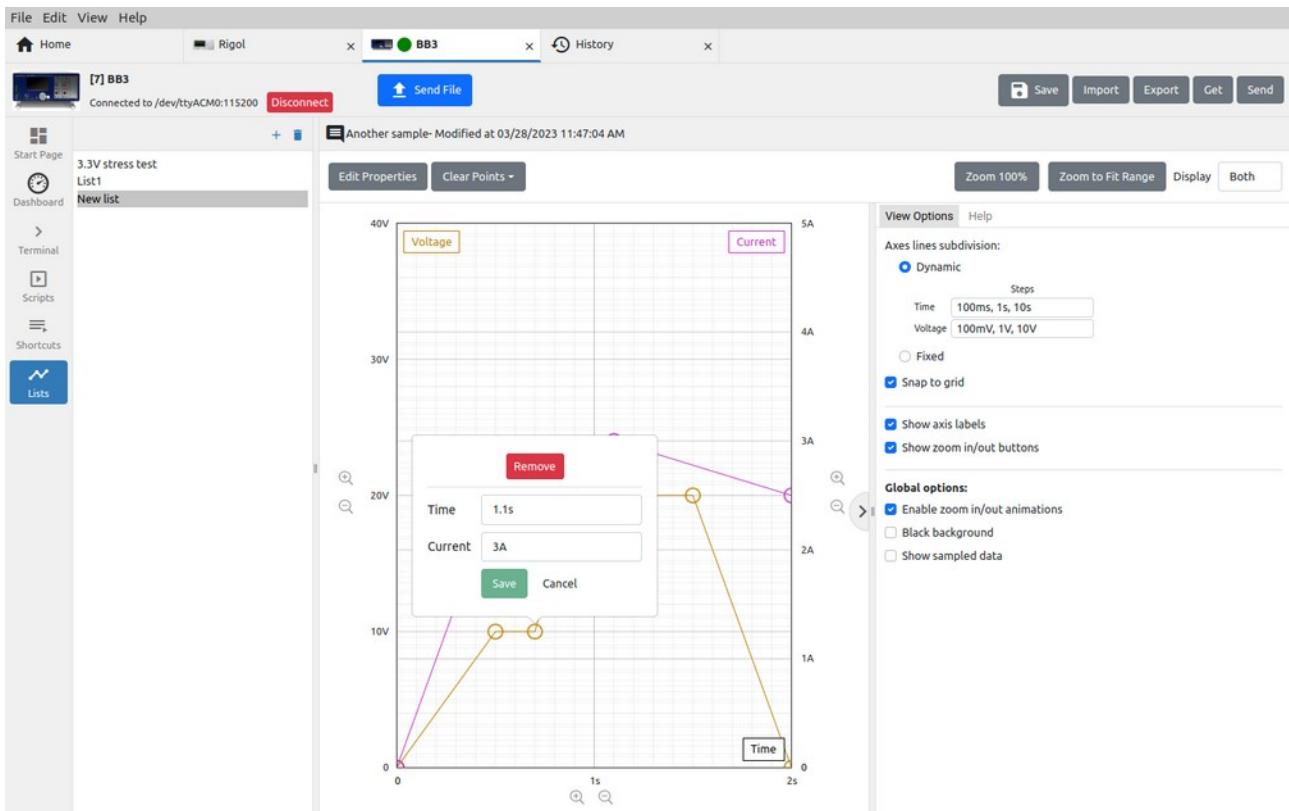


Fig. 67: Program point editing in envelope mode

I2.7.3. List view options

The display of the graph can be dynamically changed (Fig. 59) depending on the resize of the window or the number of graticules can be fixed (Fig. 68).

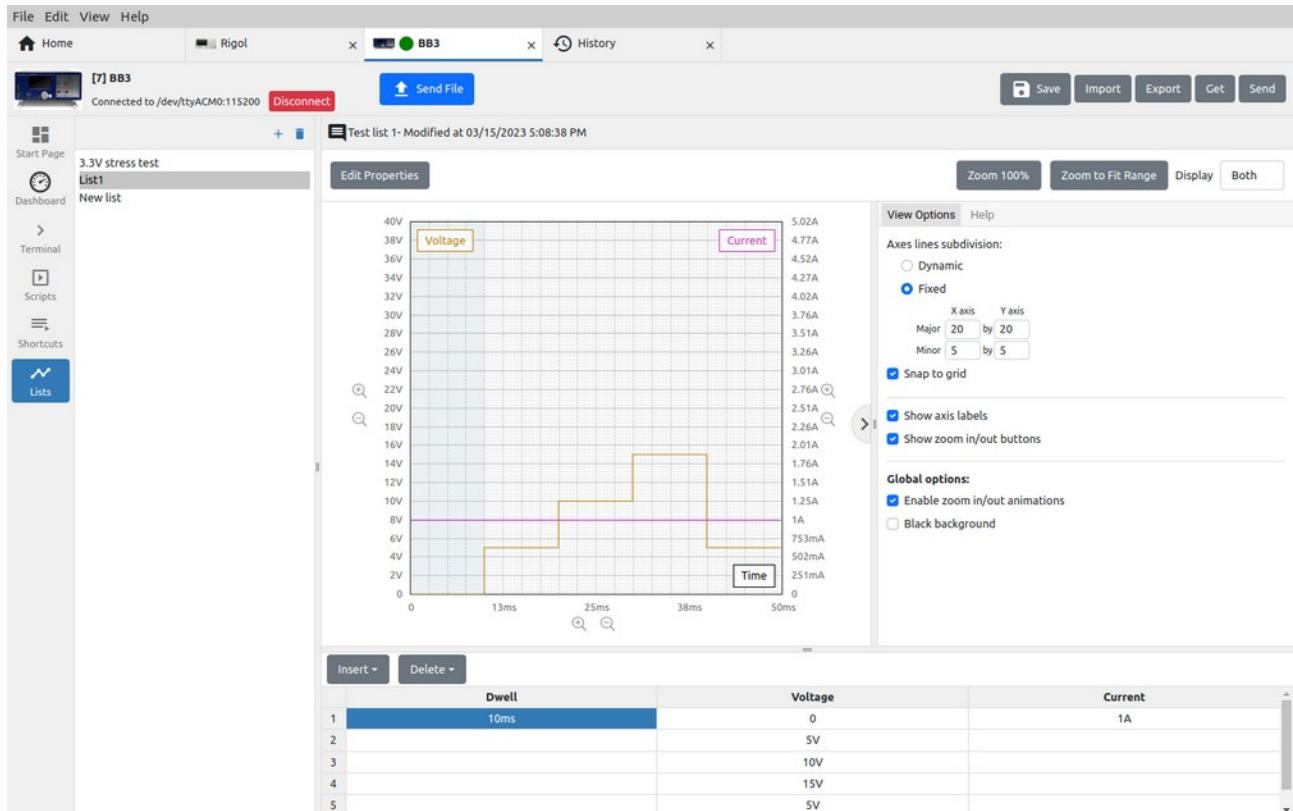


Fig. 68: Fixed graph view

I2.7.4. 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. 69.

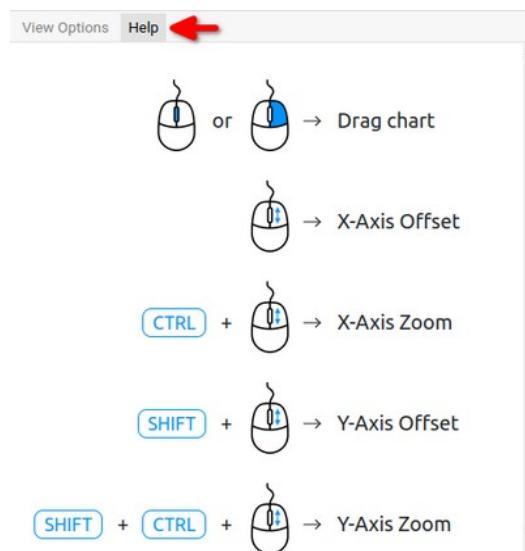


Fig. 69: Graph navigation and zoom help

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

Version: 0.22.0
Date: 2025-02-08