



EEZ Studio Reference guide

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

Ver. 0.10.3 (M17) – 08/2023
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.6
C2.The EEZ Studio overview.....	C.7
C2.1.Introduction.....	C.7
C2.2.Main sections.....	C.7
C2.3.Known issues and issue reporting.....	C.7
C2.4.Donations.....	C.7
C3.Installation.....	C.9
C3.1.System requirements.....	C.9
C3.2.Linux.....	C.9
C3.3.Mac.....	C.9
C3.4.Windows.....	C.9
C3.5.Nix package manager.....	C.10
C3.6.Build and run from source (all operating systems).....	C.10
C3.6.1.Linux only.....	C.10
C3.6.2.Raspbian only.....	C.10
C3.6.3.All platforms.....	C.10
C3.6.4.Raspbian.....	C.10
C3.6.5.Nix.....	C.10
C3.7.USB TMC.....	C.10
C3.7.1.Windows.....	C.11
C3.7.2.Linux.....	C.11
C3.8.FAQ.....	C.11
C4.Key features.....	C.13
C4.1.General.....	C.13
C4.2.EEZ Studio Project.....	C.13
C4.3.EEZ Studio Instrument.....	C.13
C5.Menu options and Settings.....	C.15
C5.1.Home page.....	C.15
C5.2.Menu options.....	C.15
C5.2.1.File.....	C.15
C5.2.2.Edit.....	C.16
C5.2.3.View.....	C.16
C5.2.4.Help.....	C.17

C1. Legal information

C1.1. Definitions

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

C1.2. Disclaimers

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

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

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

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

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

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

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

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

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

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

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

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

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

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

C1.3. Miscellaneous

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

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

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

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

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

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

C1.4. Contact information

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

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

E-mail: support@envox.eu

Website: www.envox.eu

C1.5. Revision history

Date	Version	Changes
2023-08-31	0.10.3 (M17)	Initial release

C2. The EEZ Studio overview

C2.1. Introduction

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

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

C2.2. Main sections

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

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

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

C2.3. Known issues and issue reporting

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

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

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

C2.4. Donations

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

C3. Installation

C3.1. System requirements

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

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

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

C3.2. Linux

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

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

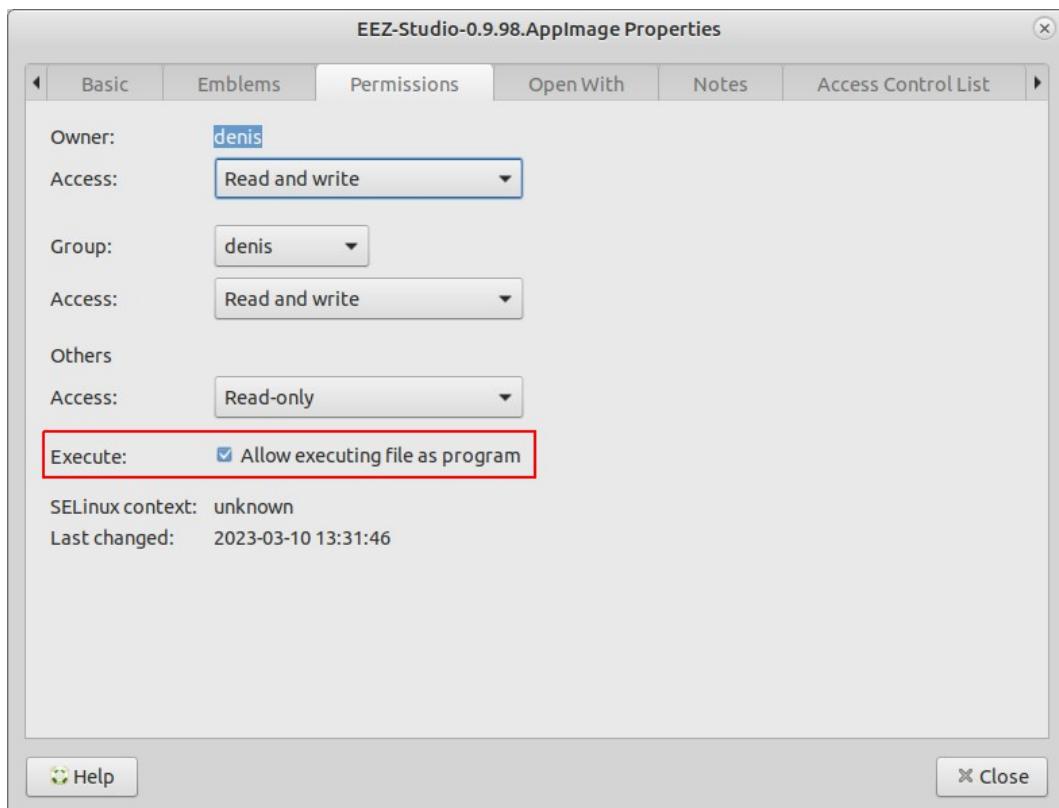


Fig. 1: .AppImage file permission

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

C3.3. Mac

Required OS version: macOS 10.10 (Yosemite) or newer

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

C3.4. Windows

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

Download and start *EEZ_Studio_setup.exe*.

C3.5. Nix package manager

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

C3.6. Build and run from source (all operating systems)

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

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

C3.6.1. Linux only

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

C3.6.2. Raspbian only

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

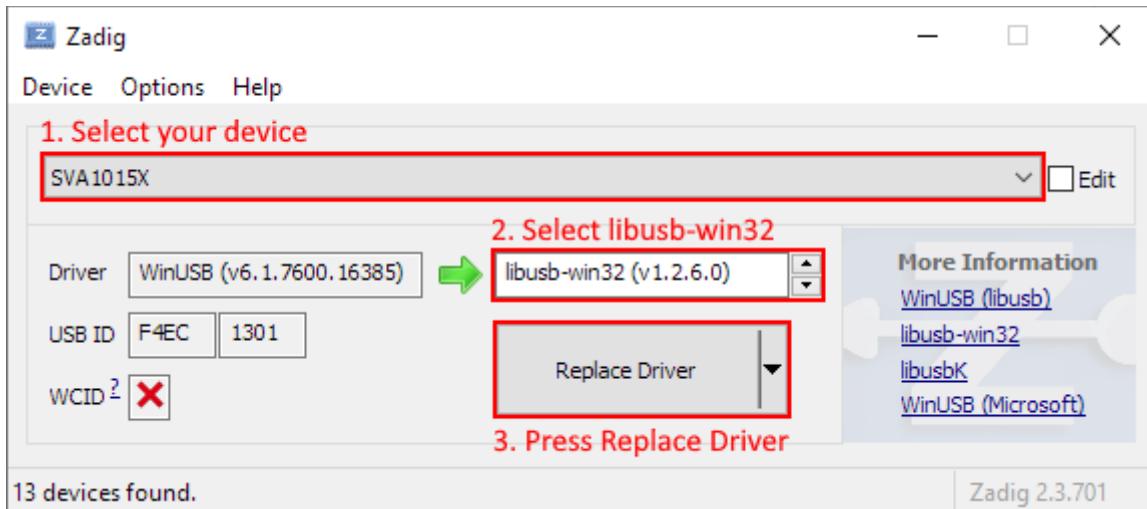


Fig. 2: Zadig driver settings

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

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, Widget styles, user created Widgets and Actions
- 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 giteo.io repositories) and comparison of projects
- Drag&drop editor for creating instrument's desktop dashboard (for remote control and management)

C4.3. EEZ Studio Instrument

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

C5. Menu options and Settings

C5.1. Home page

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

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

The *Projects* section will be described in detail in chapters started with prefix P (i.e. P.1, P.2, ...), and the *Instruments* section in chapters started with prefix I.

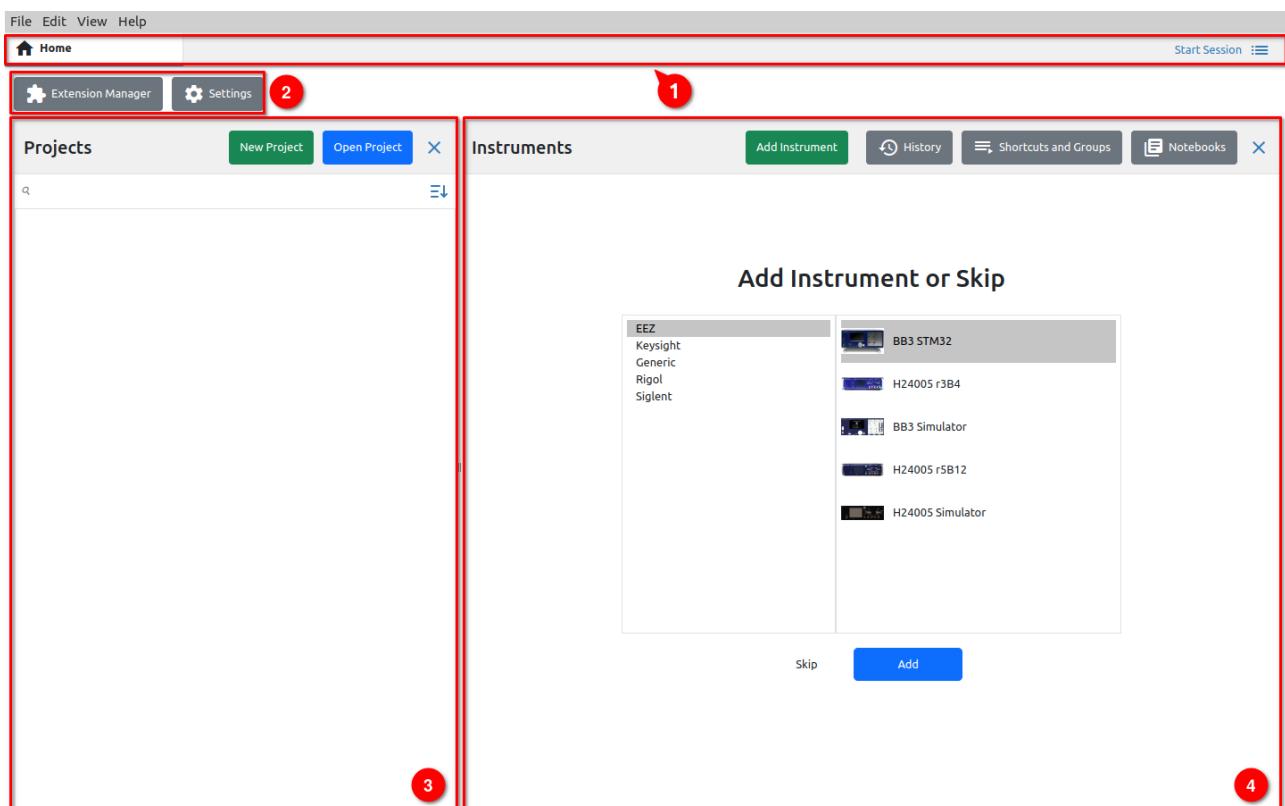


Fig. 3: Home page

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.

<i>Load Debug Info...</i> <i>(Projects only)</i>	–	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...</i> <i>(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.

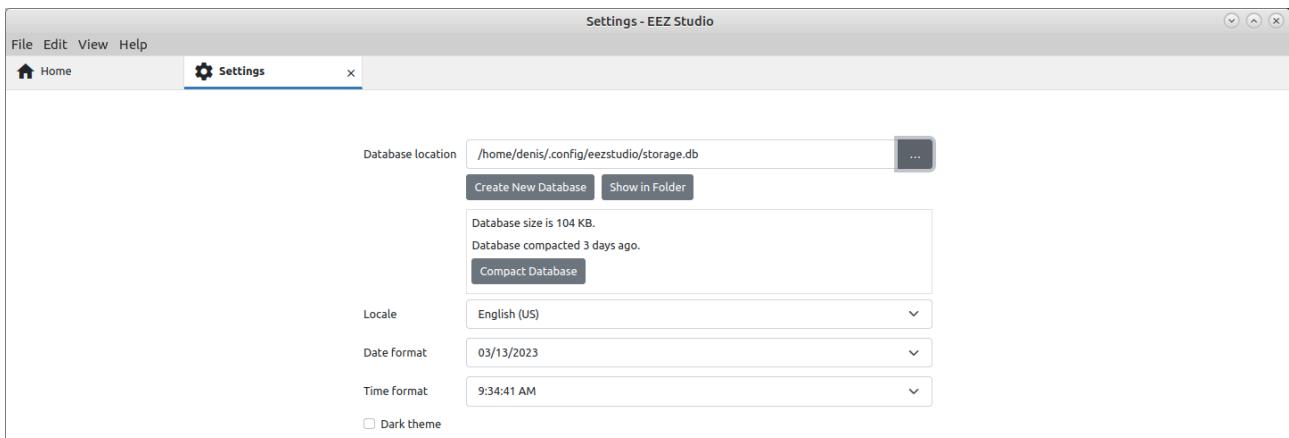
C5.2.2. Edit

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***Database location**

A database is used to store the data collected in communication with the instruments. An empty base is created at first launch and its location can be seen here. You can also change the location here to one of the existing databases (backup, imported from another EEZ Studio, etc.).

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

Create New Database

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

Show in Folder

View the folder where the database is located.

Locale

Defines the date and time formats for the selected country.

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

Date format

Display format of all date values.

Time format

Display format of all time values.

Dark theme

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

C5.2.4. Help

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

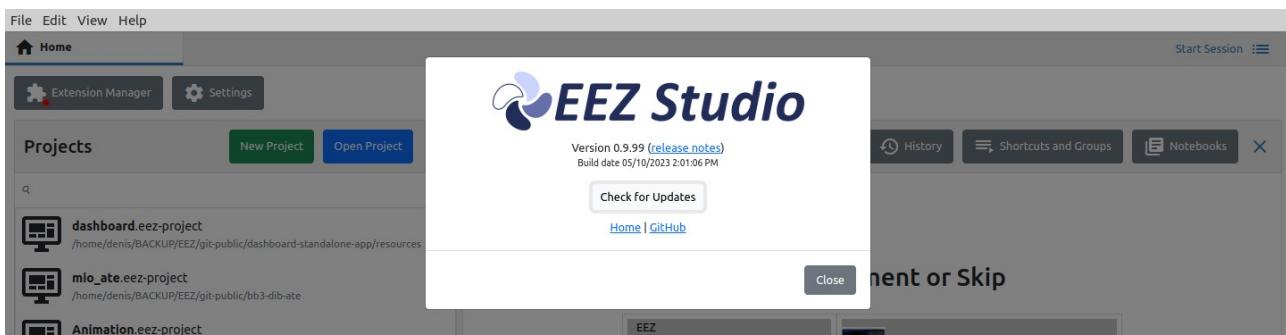


Fig. 5: About page

Check for Updates

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

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

Home

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

Github

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

*EEZ Studio
Project*

Table of Contents

P1.Home page project sections.....	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.22
P4.Project editor panels.....	P.23
P4.1.Panel items.....	P.23
P4.2.Right-click menu.....	P.24
P4.3.Edit mode panels overview.....	P.25
P4.3.1.Search and Replace.....	P.26
P4.3.2.Replace.....	P.26
P4.4.Debug mode panels overview.....	P.27
P5.Project editors/viewers.....	P.29
P5.1.Editors.....	P.29
P5.1.1.Page editor.....	P.29
P5.1.2.User Actions.....	P.29
P5.1.3.User Widgets.....	P.30
P5.1.4.Font editor.....	P.30
P5.1.5.Shortcuts (EEZ-GUI only).....	P.31
P5.1.6.MicroPython (EEZ BB3 only).....	P.31
P5.1.7.Readme.....	P.31
P5.1.8.Settings.....	P.32
P5.2.Viewers.....	P.33
P5.2.1.Page viewer.....	P.33
P5.2.2.Action viewer.....	P.34
P6.EEZ Flow.....	P.35
P6.1.EEZ Flow basic concepts.....	P.35
P6.2.Flow execution.....	P.36
P6.3.Flow examples.....	P.38
P7.Project editing.....	P.39
P7.1.1.Connecting Flow components.....	P.42
P7.1.2.Copy & Paste between two projects.....	P.43
P7.2.Working with Widgets.....	P.44
P7.2.1.Widget component's items.....	P.44
P7.2.2.Creating a User Widget.....	P.45
P7.3.Working with Actions.....	P.47
P7.3.1.Action component's items.....	P.48
P7.3.2.Creating a User Action.....	P.48
P8.Variables.....	P.51
P8.1.Variables usage in the project with EEZ Flow enabled.....	P.52
P8.2.Variable types	P.53

P8.2.1.Basic/Primitive types	P.53
P8.2.2.Structures.....	P.53
P8.2.3.Enums	P.53
P8.2.4.Objects	P.53
P8.2.5.Arrays	P.53
P8.2.6.Expressions.....	P.53
P8.2.7.Literals.....	P.54
P8.2.8.Binary Operators.....	P.54
Addition +.....	P.54
Subtraction -.....	P.55
Multiplication *	P.55
Division /.....	P.55
Remainder %.....	P.55
Left shift <<.....	P.55
Right shift >>.....	P.56
Binary AND &.....	P.56
Binary OR 	P.56
Binary XOR ^.....	P.56
P8.2.9.Logical operators.....	P.56
P8.2.10.Unary operators.....	P.56
P8.2.11.Conditional (ternary) operator.....	P.57
P8.3.Functions.....	P.57
P8.3.1.System.....	P.57
System.getTick.....	P.57
P8.3.2.Flow.....	P.57
Flow.index.....	P.57
Flow.isPageActive.....	P.57
Flow.pageTimelinePosition.....	P.57
Flow.makeValue.....	P.57
Flow.makeArrayValue.....	P.58
Flow.languages.....	P.58
Flow.translate.....	P.58
Flow.parseInteger.....	P.58
Flow.parseFloat.....	P.58
Flow.parseDouble.....	P.59
P8.3.3.Date.....	P.59
Date.now.....	P.59
Date.toString.....	P.59
Date.toLocaleString.....	P.59
Date.fromString.....	P.59
Date.getYear.....	P.60
Date.getMonth.....	P.60
Date.getDay.....	P.60
Date.getHours.....	P.60
Date.getMinutes.....	P.60
Date.getSeconds.....	P.60
Date.getMilliseconds.....	P.61
Date.make.....	P.61
P8.3.4.Math.....	P.61
Math.sin.....	P.61
Math.cos.....	P.61
Math.pow.....	P.62
Math.log.....	P.62
Math.log10.....	P.62
Math.abs.....	P.62
Math.floor.....	P.62
Math.ceil.....	P.63
Math.round.....	P.63
Math.min.....	P.63
Math.max.....	P.63
P8.3.5.String.....	P.63

String.length.....	P.63
String.substring.....	P.64
String.find.....	P.64
String.padStart.....	P.64
String.split.....	P.64
P8.3.6.Array.....	P.65
Array.length.....	P.65
Array.slice.....	P.65
Array.allocate.....	P.65
Array.append.....	P.65
Array.insert.....	P.65
Array.remove.....	P.66
Array.clone.....	P.66
P8.3.7.LVGL.....	P.66
LVGL.MeterTickIndex.....	P.66
P8.4.Expression Builder.....	P.66
P9.Styles and Color themes.....	P.69
P9.1.Overview.....	P.69
P9.2.Style properties.....	P.69
P9.3.Project Styles.....	P.70
P9.3.1.Creating a new Style.....	P.71
P9.3.4.Style hierarchy.....	P.71
P9.4.1.Setting the Style attribute color from the palette.....	P.72
P9.4.2.Setting the Style attribute color using the Color theme.....	P.73
P9.5.Style attributes.....	P.74
P9.5.1.EEZ-GUI project.....	P.74
P9.5.2.Dashboard project.....	P.75
P9.5.3.LVGL project.....	P.76
P9.5.4.Inheriting local Style attributes.....	P.78
P10(Bitmap.....)	P.79
P10.1.Adding a bitmap.....	P.79
P10.2.Bitmap properties.....	P.80
P10.3.Using a bitmap.....	P.81
P11.Fonts.....	P.83
P11.1.EEZ-GUI project fonts.....	P.83
P11.1.1.Add new font.....	P.83
P11.1.2.Add character.....	P.84
P11.2.LVGL project fonts.....	P.85
P11.2.1.Add new font.....	P.85
P11.2.2.Edit characters.....	P.87
P12.Texts.....	P.89
P12.1.Texts panel.....	P.89
P12.2.XLIFF Import/export.....	P.90
P13.Settings.....	P.93
P13.1.General.....	P.93
P13.2.Build.....	P.94
P13.2.1.Configurations.....	P.94
P13.2.2.Files.....	P.95

P1. Home page project sections

One of the important features of EEZ Studio is that it enables the creation of projects for different target platforms using different technologies, which will be described below. The *Projects* section of the home page is shown in Fig. 1. which displays a searchable Recent Project List (RPL).

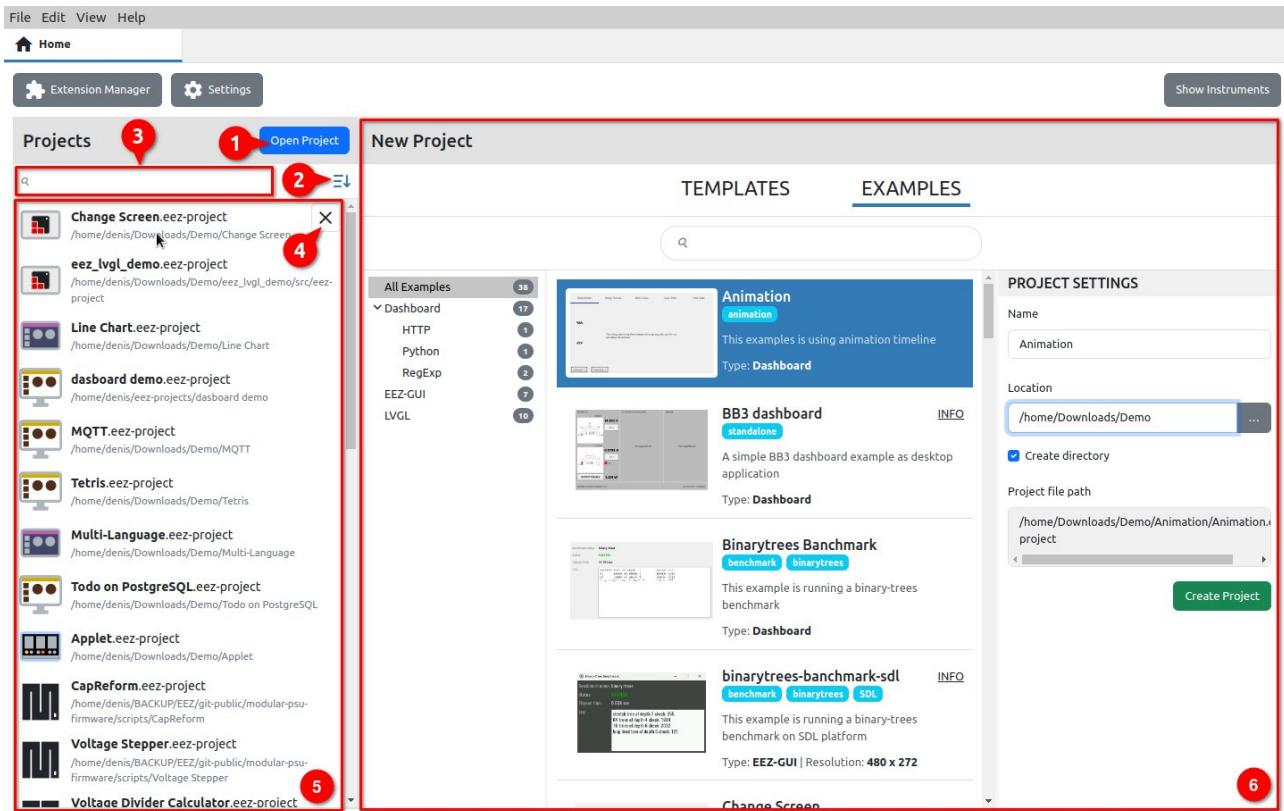


Fig. 1: Home page project options

Option

1 *Open project*

2 *RPL sort order*

3 *Search RPL*

4 *Remove from RPL*

5 *Recent Project List (RPL)*

6 *New project*

Description

Opening an existing project (will be added to RPL after successful loading).

Sorting order of projects in RPL: It can be *Show most recent first* or *Sort alphabetically*.

RPL search by project name.

Removing the project from RPL.

List of all successfully loaded projects after the first run.

Creating a new project.

P1.1. EEZ Studio project types

EEZ Studio offers the creation of the following project types:

- **Dashboard** – desktop application. GUI applications can be quickly and easily created thanks to the drag & drop of available widgets and the import of multiple fonts and ready-made bitmaps prepared by the designer. The animation editor allows adding simple animations to the desired sections of the page or navigation between pages. Finally, the flowchart method of defining program logic instead of programming in one of the programming languages will further speed up prototyping and creation of the final application. The implemented debugger will shorten the application development process and help in more efficient error detection.
- **EEZ-GUI** – embedded GUI application that uses the EEZ-GUI framework. This is a native EEZ Studio framework that was initially developed to speed up and simplify embedded GUI development for [EEZ H24005](#) and [EEZ BB3](#) firmware.

- **LVGL** – embedded GUI application that uses LVGL (Light and Versatile Graphics Library) framework. LVGL is a popular open source project that supports a large number of target platforms. For more information visit <https://lvgl.io/>
- **Applet** – GUI application that can be run on EEZ BB3. Program logic is created using EEZ Flow (flowchart-based programming).
- **MicroPython script** – GUI application that can be run on EEZ BB3. Program logic is created using MicroPython scripting.
- **Templates from gitea repository** – Completed projects located in the gitea.io repository (mostly based on the EEZ-GUI framework). They can be used as a starting point for creating new projects.
- **Empty** – Creating an empty project for advanced users who want to configure everything themselves from the start.

P1.2. Create new project

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

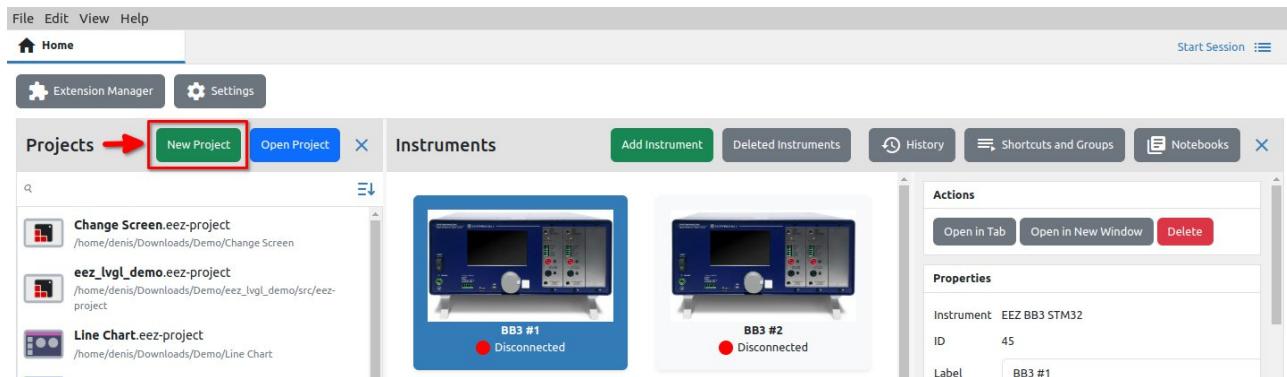


Fig. 2: New project as button when the Instruments section is also enabled

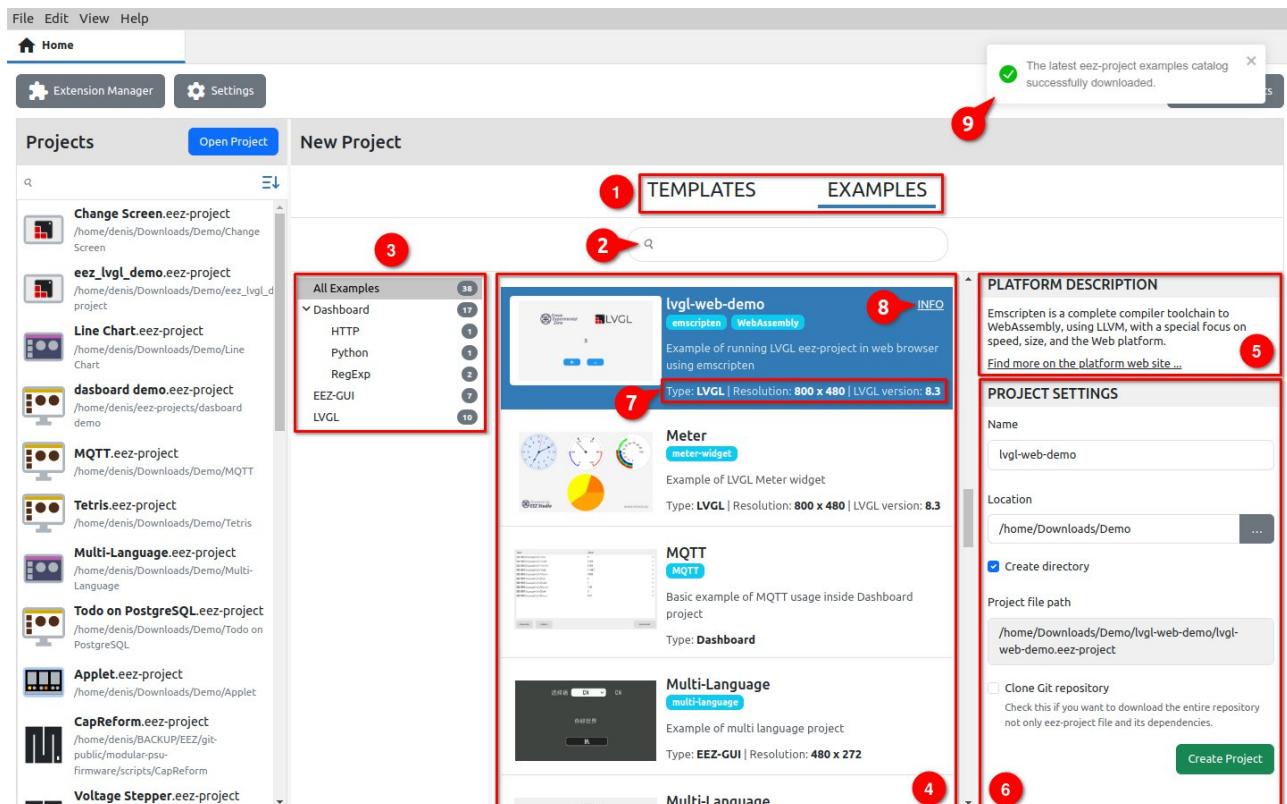


Fig. 3: Create new project

#	Option	Description
1	<i>Example category selection</i>	Examples are listed in two categories: project Templates contain the basic configuration for the selected type of project, while project Examples represent a functional application that can contain multiple Pages with User Actions and User Widgets.
2	<i>Search</i>	Search by project name.
3	<i>Project list</i>	List of all projects within the selected category, grouped into expandable sublists. If there are new examples added since the last run, the <i>New examples</i> category will appear at the top of the list.
4	<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.
5	<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.
6	<i>Project settings</i>	Project basic parameters (see below).
7	<i>Project details</i>	Basic information about the project: type, screen size and, in the case of an LVGL project, the version of the library used.
8	<i>Info</i>	If the project has a Git repository, this link will appear that takes you to the repository home page.
9	<i>Examples catalog info</i>	At the start, EEZ Studio checks whether there are changes in the example repository, and if it finds them, it displays this message after a successful download.

P1.3. Project basic settings

Name

The name of the new project.

Location

The location where the project files will be stored.

Create directory

If selected, a subdirectory (at Location) with the name of the project will be created. This option is not available if the project is taken from a Git repository (in this case a new folder is always created).

Project file path

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

Clone Git repository

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

Initialize as Git repository

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

P1.4. Additional steps for creating EEZ BB3 projects

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

EEZ Studio Reference Guide

The necessary EEZ BB3 master project can be downloaded from GitHub (Fig. 4) when creating a new project or set a reference to a local copy of the repository (Fig. 5).

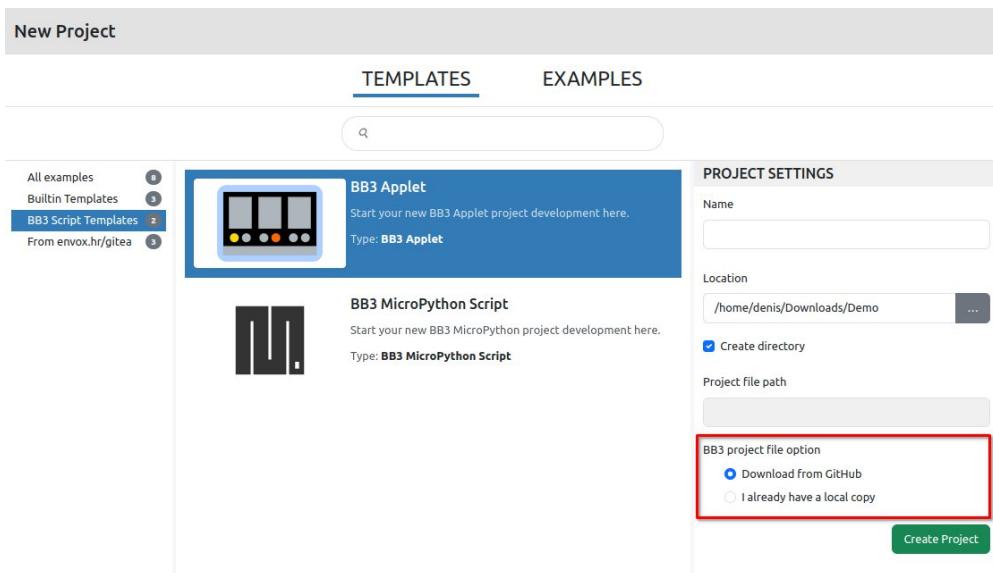


Fig. 4: EEZ BB3 applet new project additional option

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

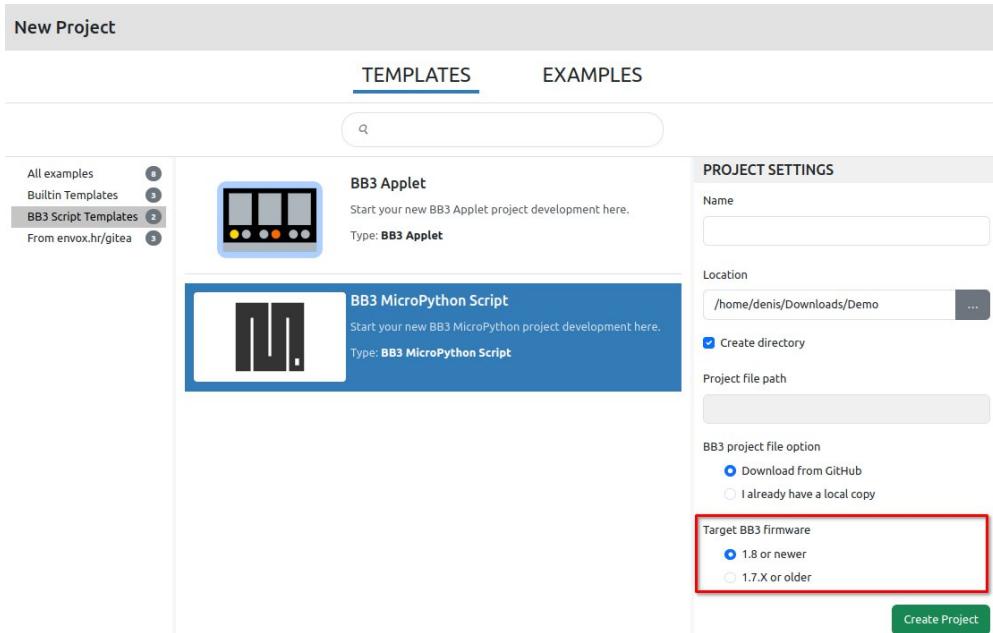


Fig. 5: EEZ BB3 MicroPython new project additional options

After all the basic parameters have been entered, the new project will be displayed in the project editor in *Edit* mode. Fig. 6 shows a new project for the *EEZ BB3 applet*. An overview of the project editor can be found in the next chapter.

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

P1.Home page project sections

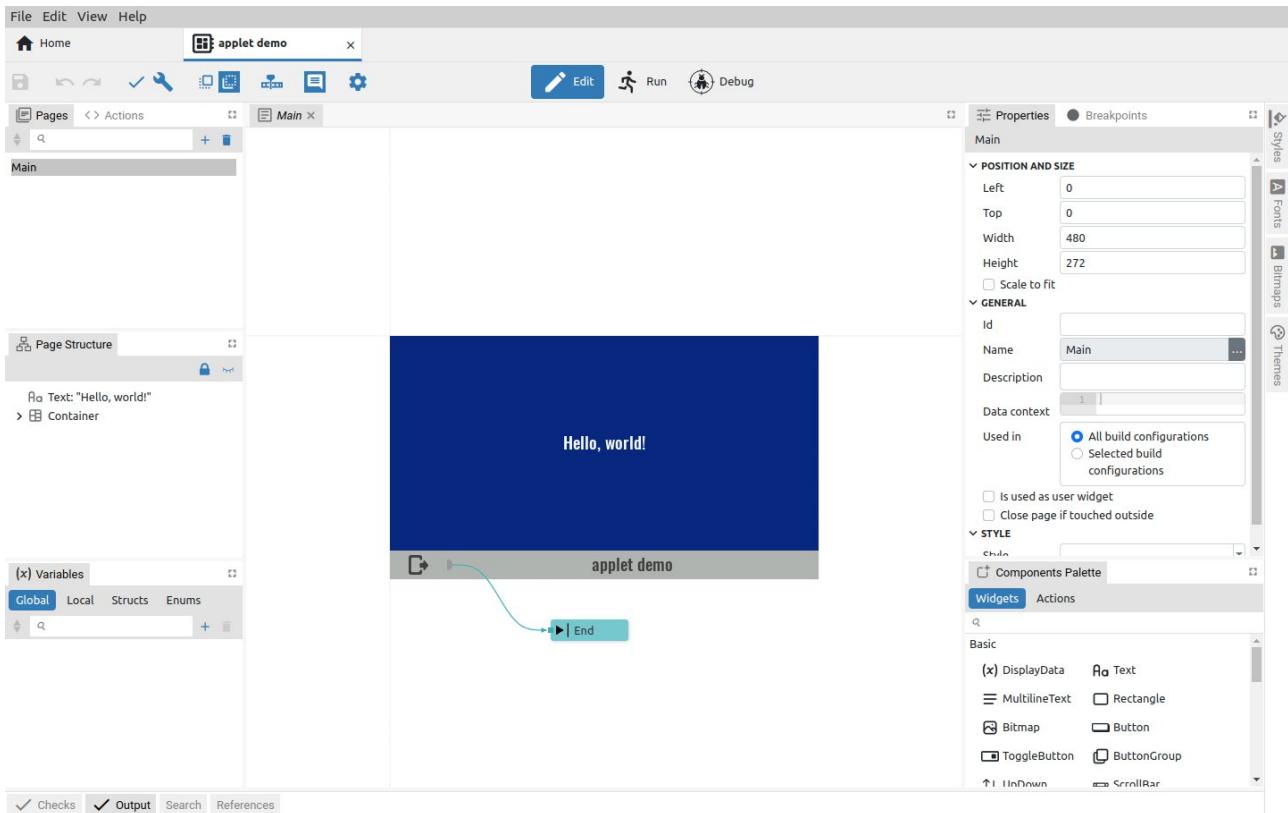


Fig. 6: Newly created project in Edit mode

The basic project settings set by the New project can be seen by clicking on the *Settings* option (1) when the project *Settings* will open in a new tab (2) as shown in Fig. 7.

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

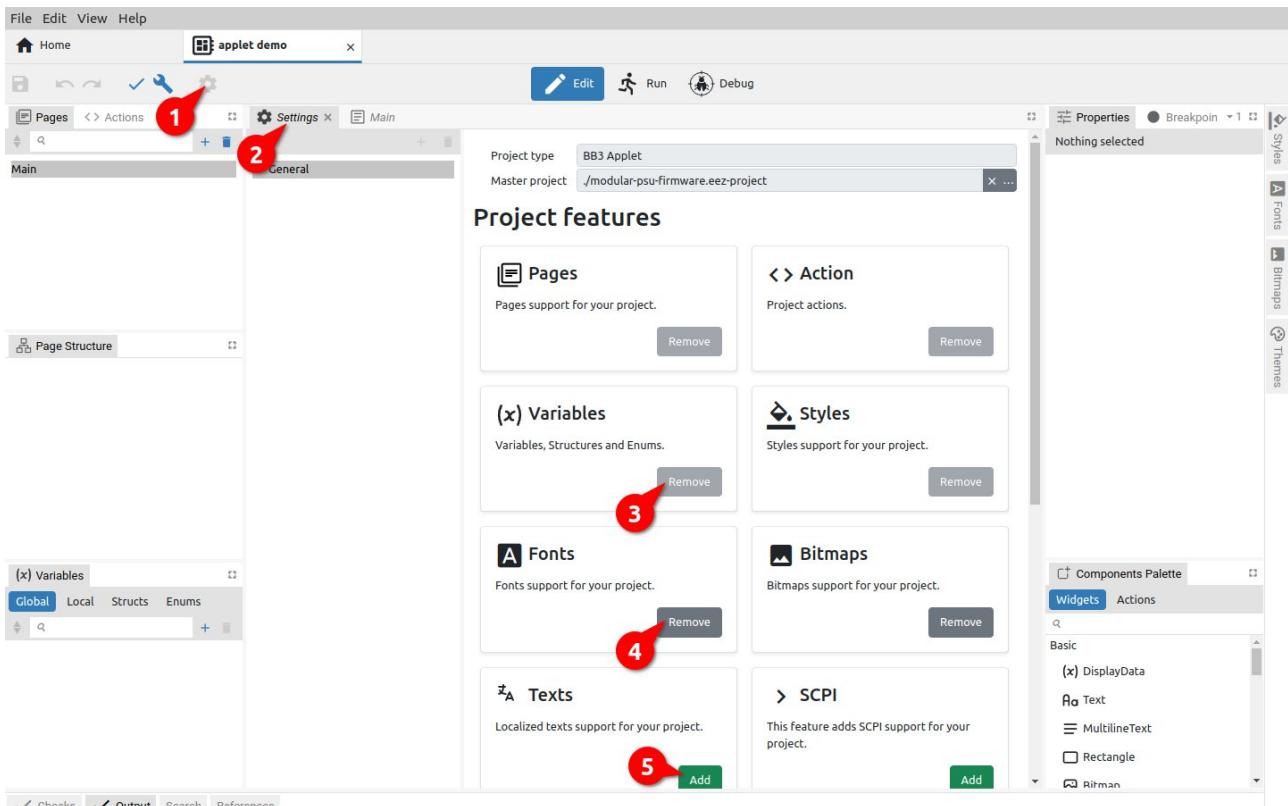


Fig. 7: Newly created project settings

P2. Project editor overview

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

P2.1. Project editor workspace

Fig. 15 shows a typical arrangement of Project editor elements. Thanks to its modern design, the Project editor offers users the freedom to rearrange them according to their own needs and taste.

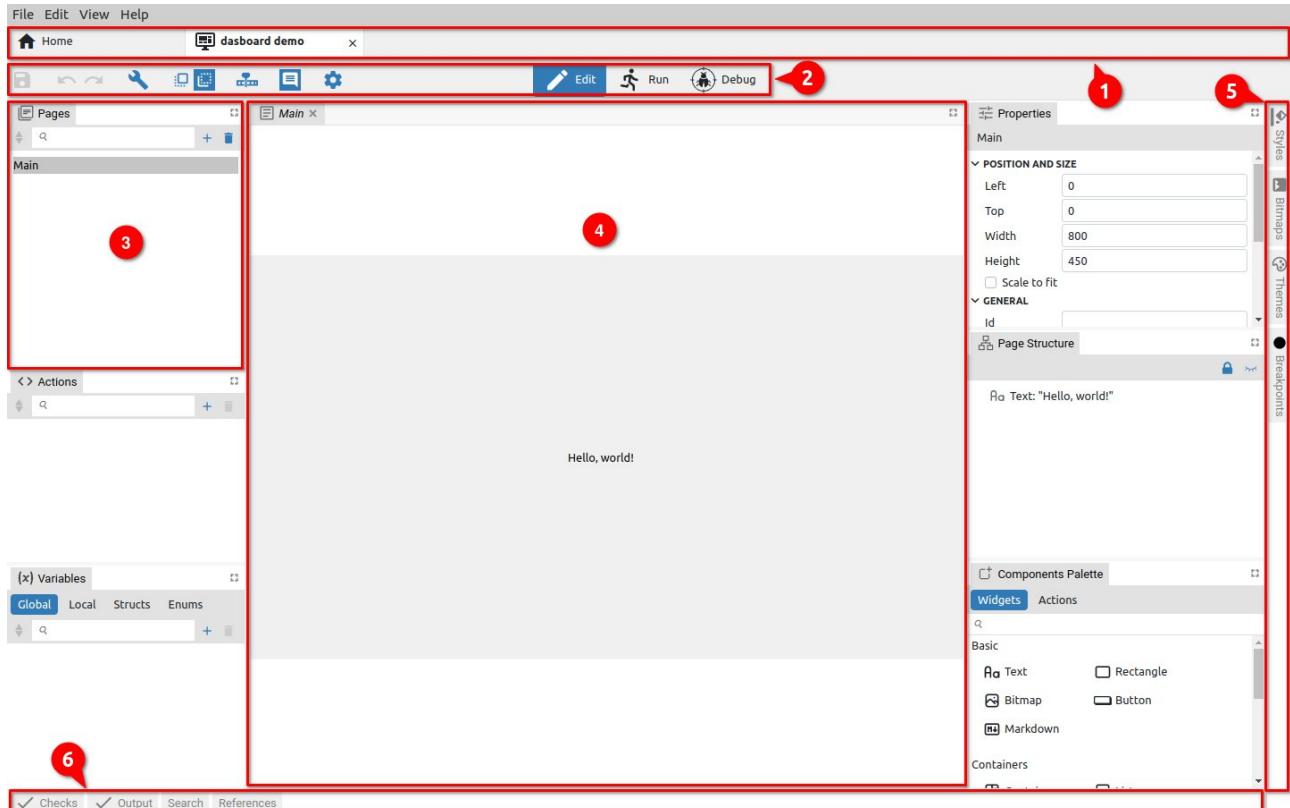


Fig. 8: Project editor sections

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

- *Toolbar* – contains icons of basic editor functions, the number of which varies depending on the type of project.
- *Panel windows* – can contain groups of project elements, components and reports e.g. *Pages*, *Actions*, *Styles*, *Fonts*, *Bitmaps*, *Variables*, *Checks*, *Output (Build results)*, *Search* and *References*. Panels can be grouped within a tabs 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 tabs. Tabs 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. 15 are explained below.

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

4 Editor tabset

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

5 Right border tabset

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

6 Bottom border tabset

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

P2.2. Display of the page in the editor

In Fig. 9 shows how it is possible to work with multiple editors. To display a page in the editor, click on the desired page (1). A new editor tab will appear, with the name of the selected page in italics (2). This indicates that the tab is not locked and if you choose another page from the list, it will replace the currently displayed one. If we want to lock the page, we will use the right click when the option *Keep Tab Open* (3) will appear. When the page is locked, its name will no longer be displayed in italics (4).

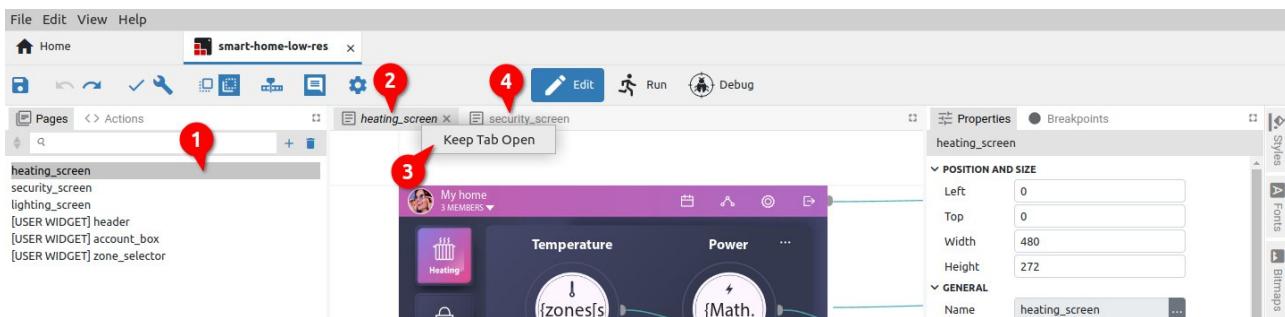


Fig. 9: Page editor tab locking

P2.3. Panel moving and docking

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

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

Below is an example of how to move the *Actions* panel to another tabset. To begin, click and hold the *Actions* tab (Fig. 10).

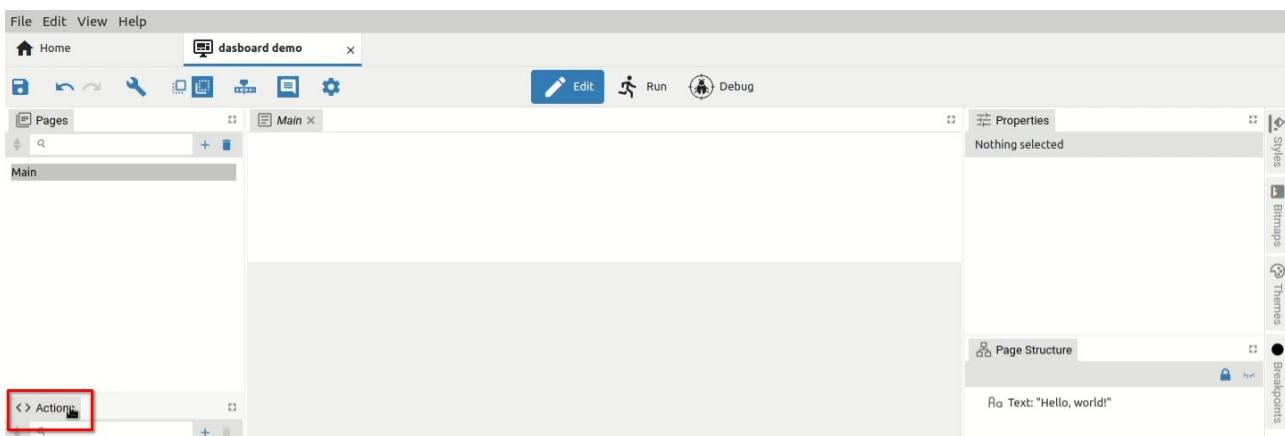


Fig. 10: Panel selection

The panel is now ready to move to another location. The cursor will change and marks will also appear on all four sides indicating the ability to dock into border tabs (Fig. 11).

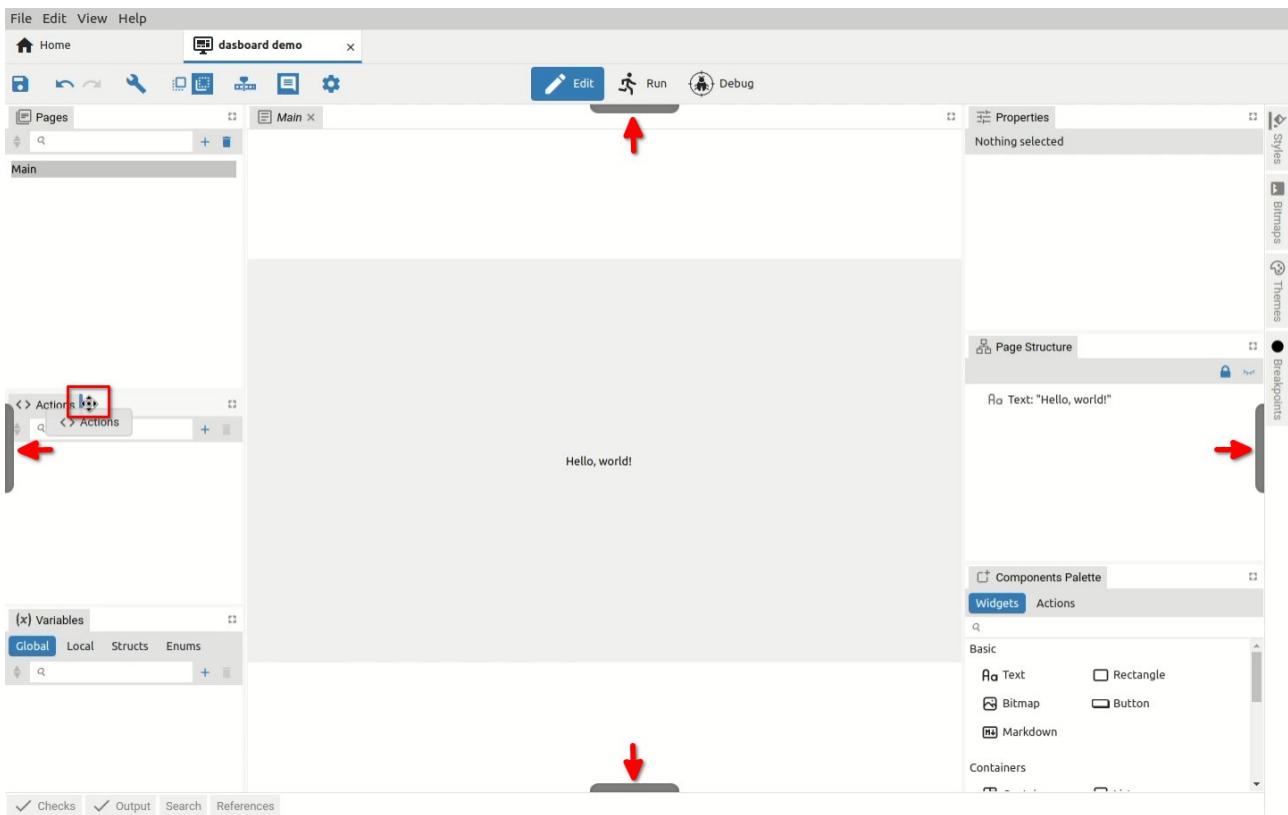


Fig. 11: Docking indicators for border tabs

Now we can choose where we want to dock the panel and whether we want it to become a new tab in the tabset or share the space occupied by an existing tabset. For example, if we want the selected panel to share horizontally the lower part of the space occupied by *Pages*, we will need to move the cursor to the lower part of the *Pages* panel when a rectangle will be displayed as in Fig. 12. Similarly, if we want the selected panel to divide vertically the right part of the space occupied by *Pages*, we will need to move the cursor to the right part of the *Pages* panel until a rectangle is displayed as in Fig. 13.

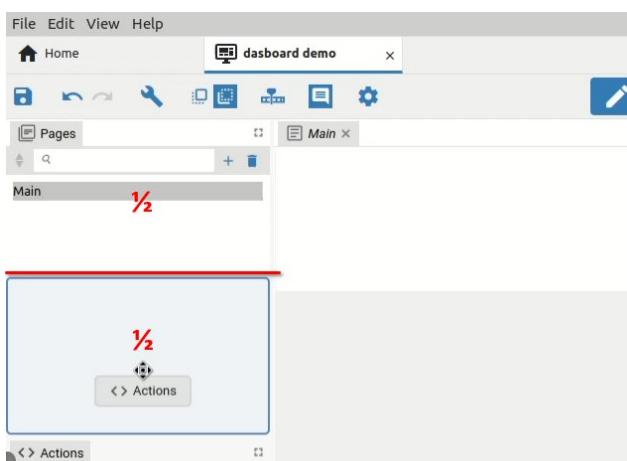


Fig. 12: Panel horizontal positioning

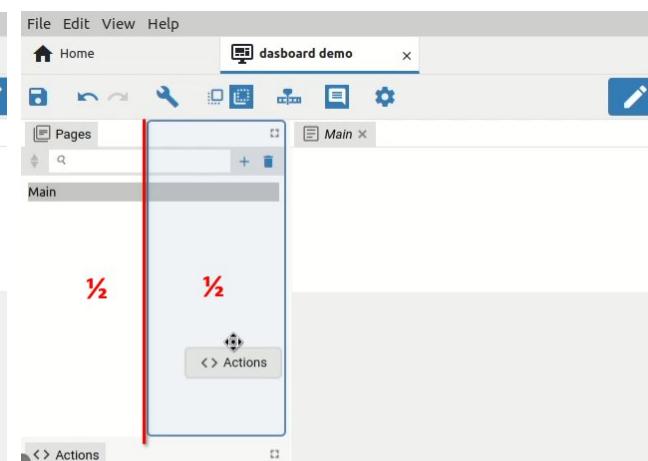


Fig. 13: Panel vertical positioning

The panel can also become a new tab within the existing tabset. This can be done in two ways: by moving the cursor next (left or right) to the existing tab in the tabset as shown in Fig. 14 or to place the cursor approximately in the middle of the existing tab so that a rectangle appears as in Fig. 8.

Note: if we move the cursor closer to the edges of the existing tab, smaller rectangles will appear indicating that the space of the existing tab will be split as shown in Fig. 12 and Fig. 13.

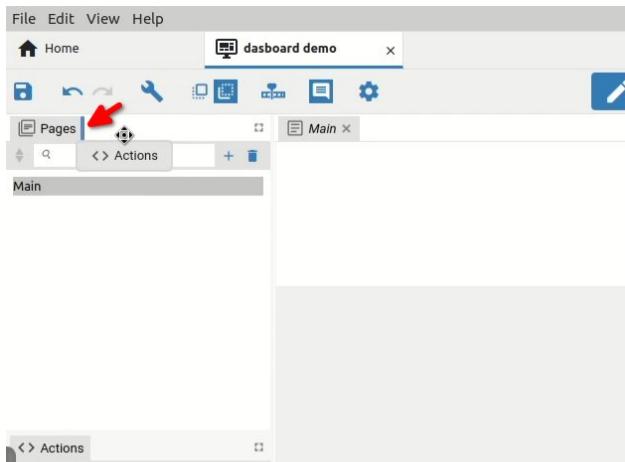


Fig. 14: Positioning in another tabset (1st method)

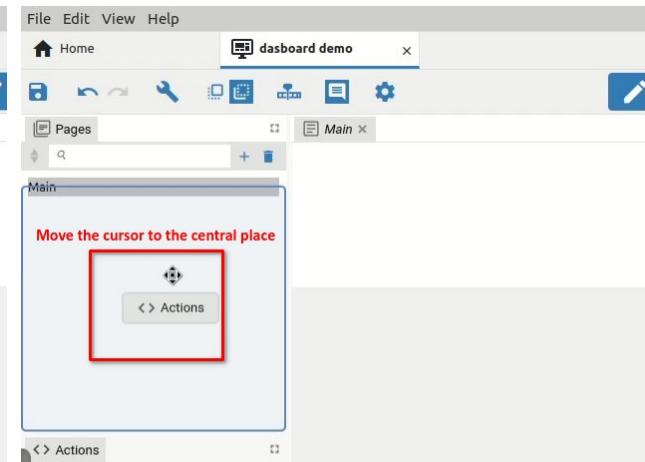


Fig. 15: Positioning in another tabset (2nd method)

Finally, when we have chosen where we want the selected panel to be displayed for docking, it will be necessary to release the mouse button. In our example, Actions will become a new tab in the tabset with Pages (Fig. 16).

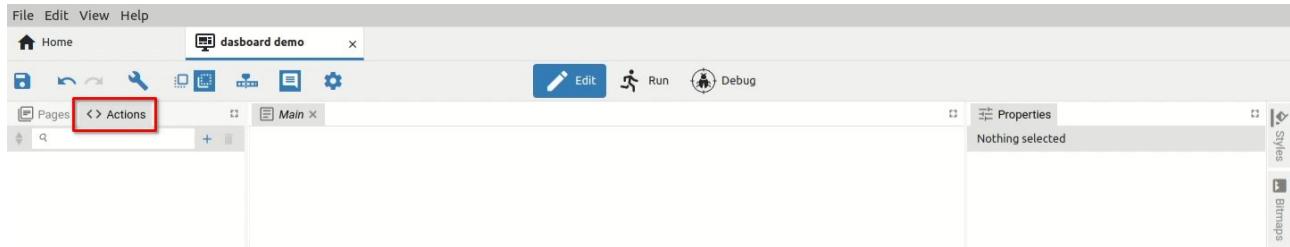


Fig. 16: Panel docking completed

P2.4. Border tabssets

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

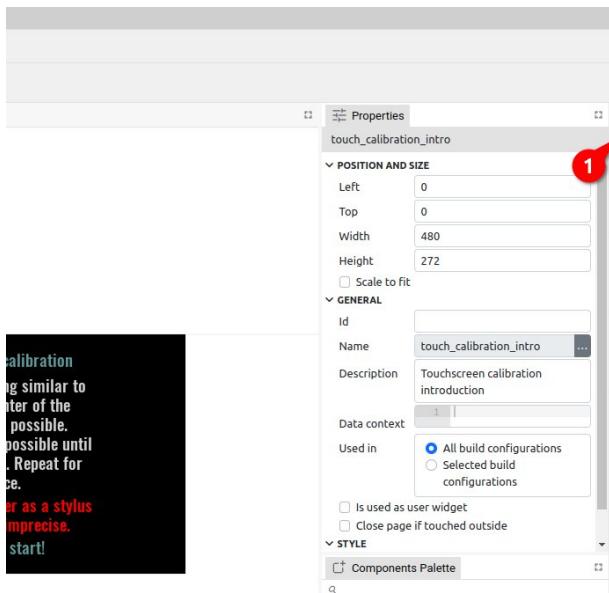


Fig. 17: Border tabs are closed

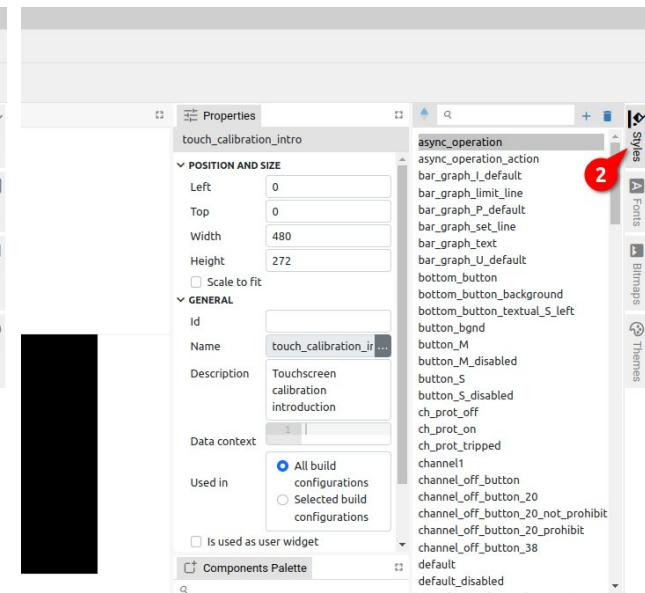


Fig. 18: The border tab is selected and opened

Panel docking is possible in Edit and Debug mode, but in Debug mode it is not possible to dock into border 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. 19).

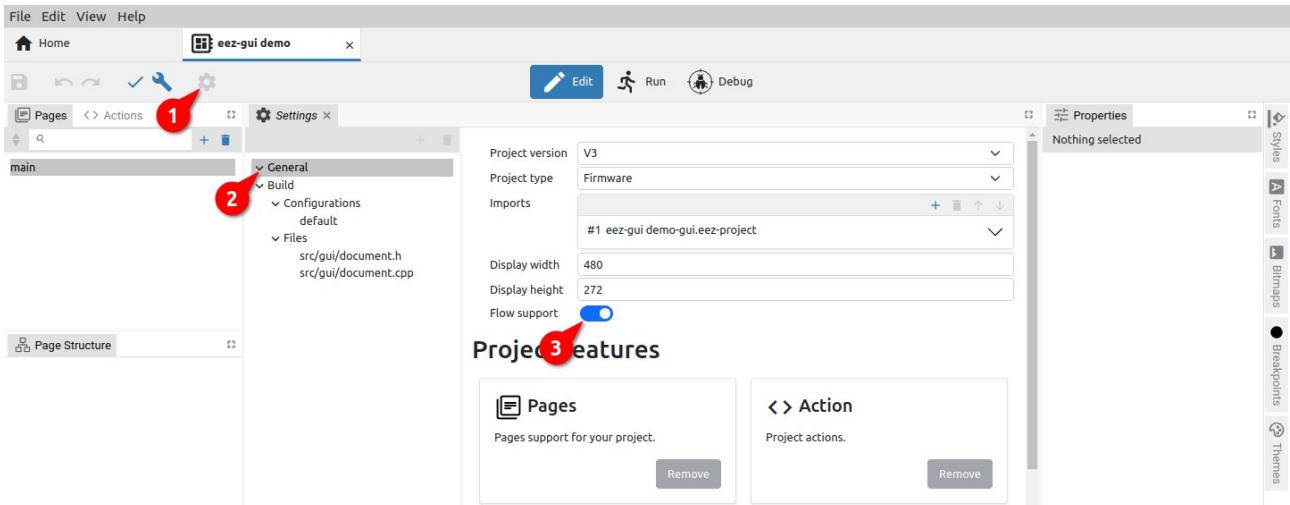


Fig. 19: Enabling EEZ Flow in project Settings

P3.1. Toolbar overview

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

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

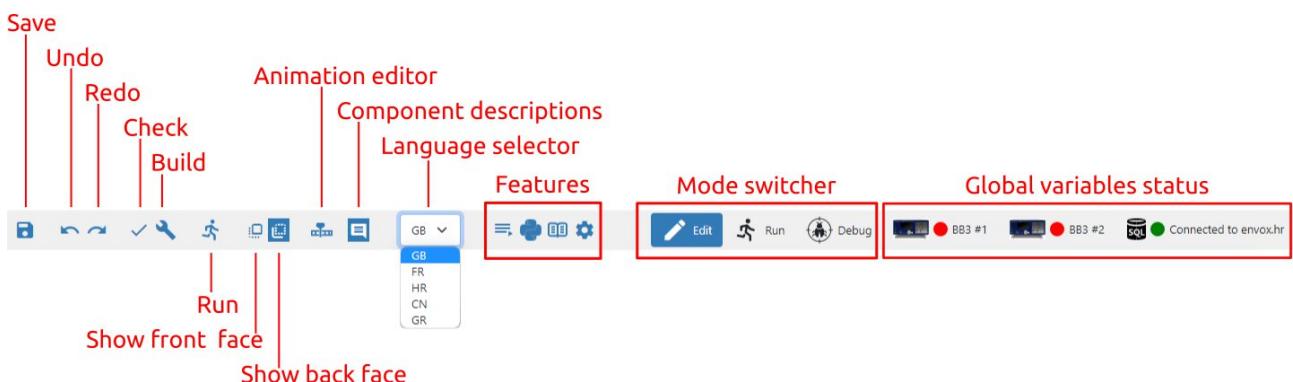


Fig. 20: Project toolbar (all options)

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

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

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

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

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

Table 1: Toolbar options in all modes

P3.2. Toolbar in Edit mode

Undo / Redo

Undo / Redo recent editor Action. If any changes have been made to the project since the last save, a * sign will appear in the project tab next to the name (Fig. 21).



Fig. 21: Indication of unsaved changes

Check

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

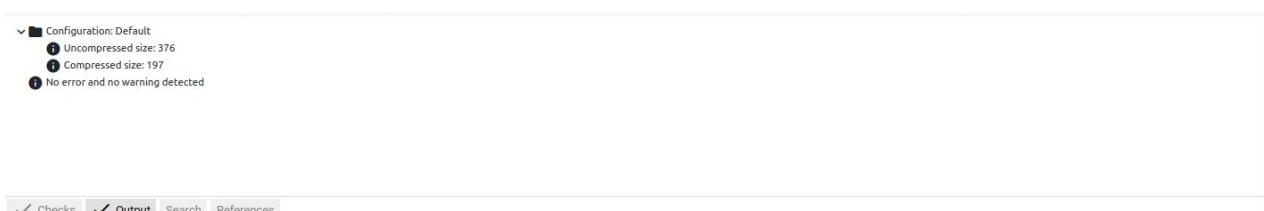


Fig. 22: Results of project checking

Build

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

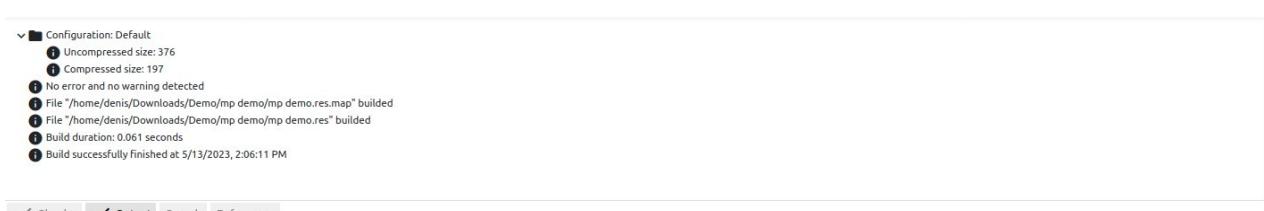


Fig. 23: Project build results

Run MicroPython Script (EEZ BB3 only)

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

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

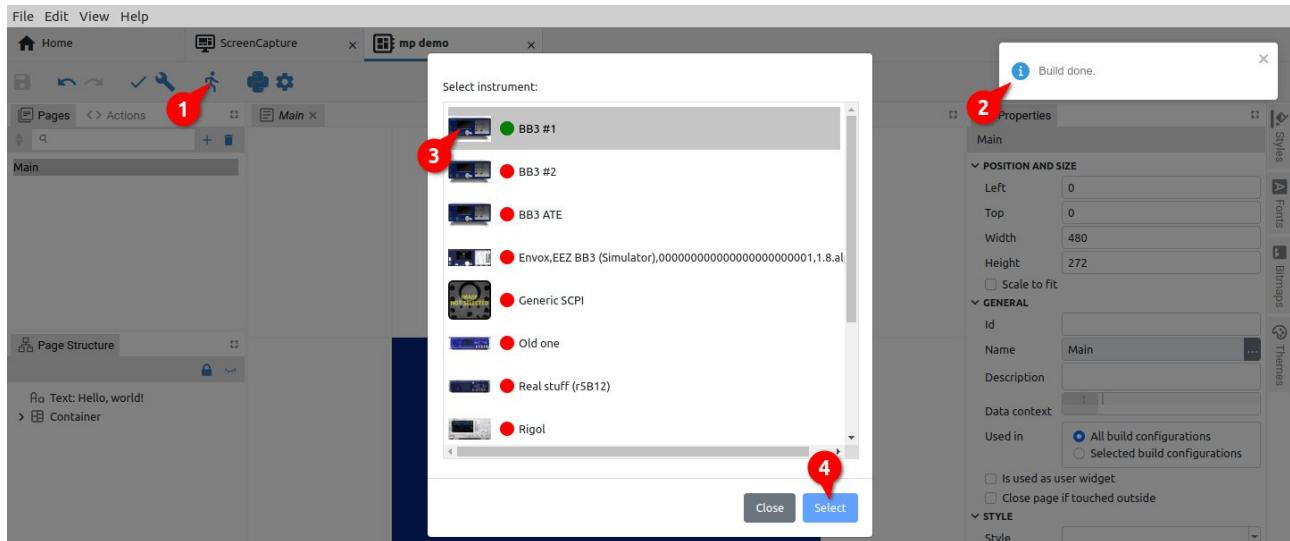


Fig. 24: Selection of target EEZ BB3 for project execution

In case there is no active connection with the selected EEZ BB3, an additional dialog box for establishing the connection will appear. For example in Fig. 25 the serial interface is selected.

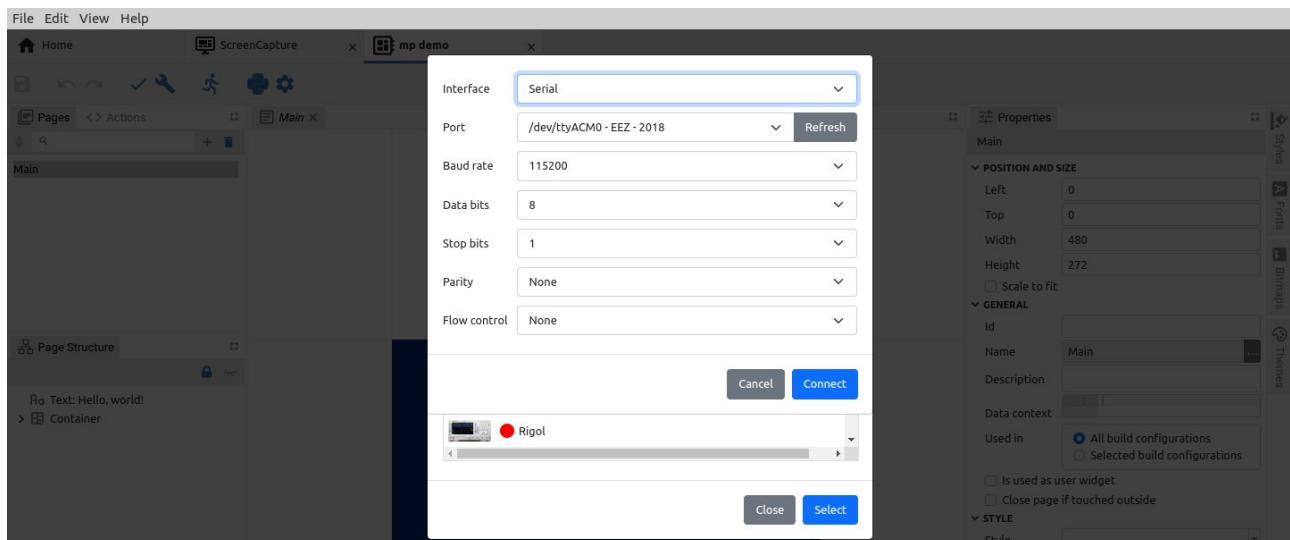


Fig. 25: Selection of target EEZ BB3 for project execution

Finally, after the project files (.py and .res) have been successfully transferred to the selected EEZ BB3, an indication will appear when the MicroPython script has been started (Fig. 26).



Fig. 26: Project execution indication

Show front face

Shows only Widgets without Action components and lines in the page editor for better readability. This button is present if EEZ Flow is enabled in the project and if the page editor is in focus (Fig. 27).

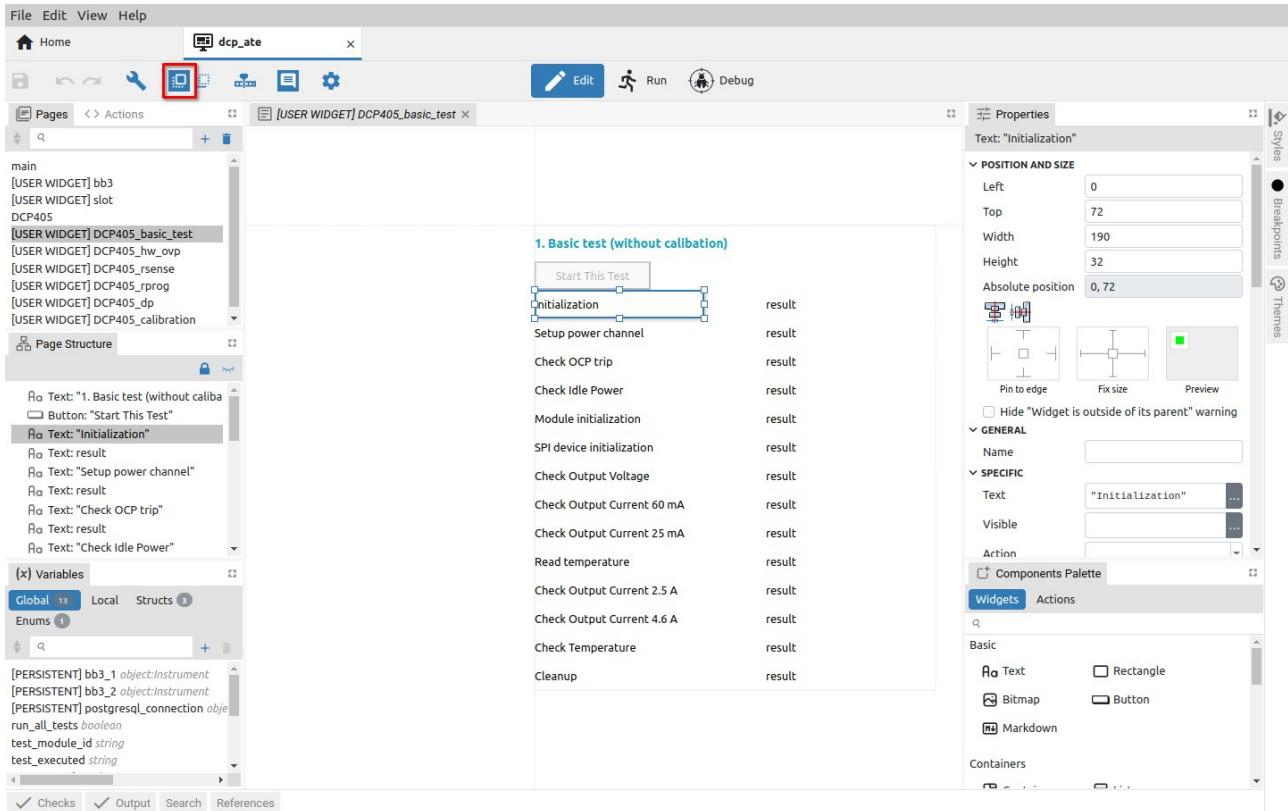


Fig. 27: Project page in front face view

Show back face

Shows all components (both Widgets and Actions) and lines in the page editor. This button is present if Flow is enabled in the project and if the page editor is in focus (Fig. 28).

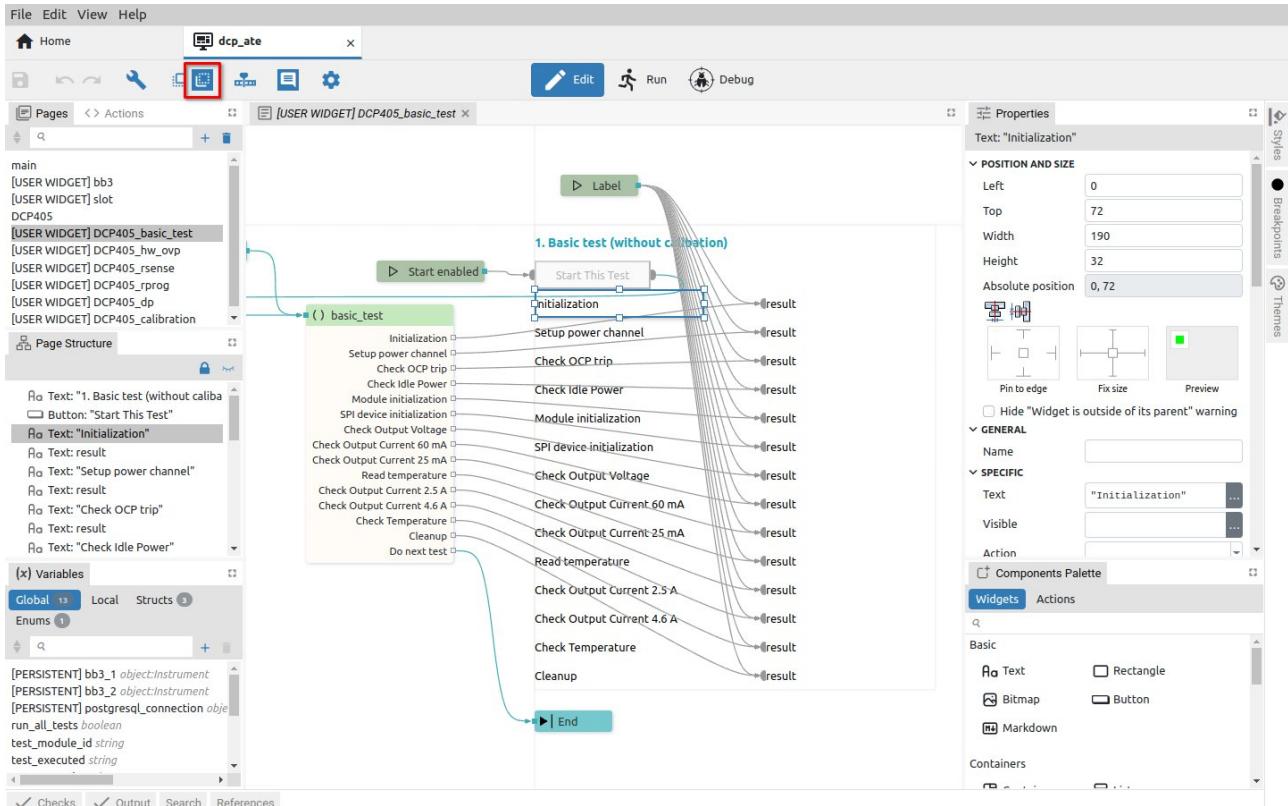


Fig. 28: Project page in back face view

Show / Hide animation timeline editor

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

An icon in the toolbar to show and hide it will appear when the page editor is in focus. EEZ Flow should also be enabled in the general settings of the project (Fig. 17).

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

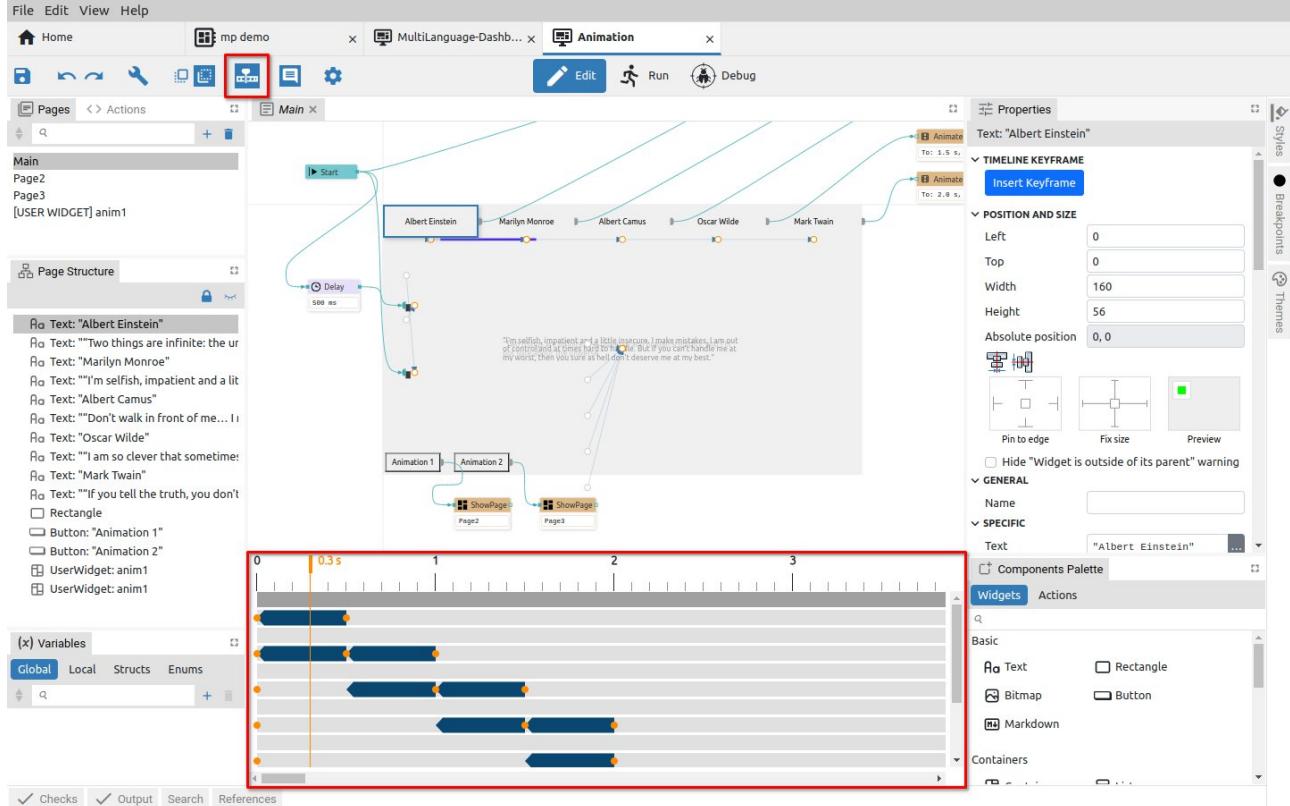


Fig. 29: Project animation timeline editor

Show / Hide component descriptions

Each component has a *Description* property. With this option, we choose whether the description will be seen under the component or not (Fig. 30).

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

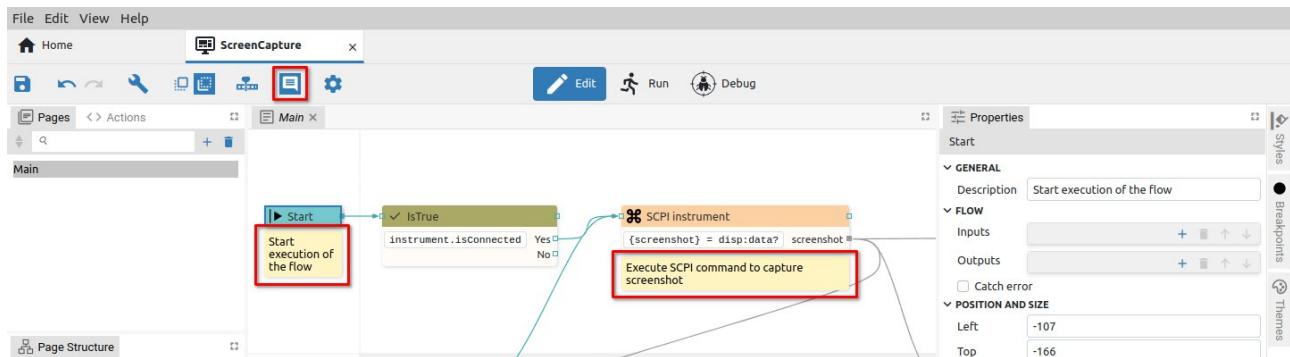


Fig. 30: Component descriptions are visible

Language selector

This option is present in the toolbar if the *Texts* feature is selected in the project general settings and if at least one language is defined. Then the *Texts* tab will appear in the left border tabset (2), whose panel (3) contains definitions of multilingual text strings, used languages and translation statistics. The texts displayed in the page editor will be displayed in the language selected in the language selector of the toolbar (1). In the example in Fig. 24 French (FR) is selected.

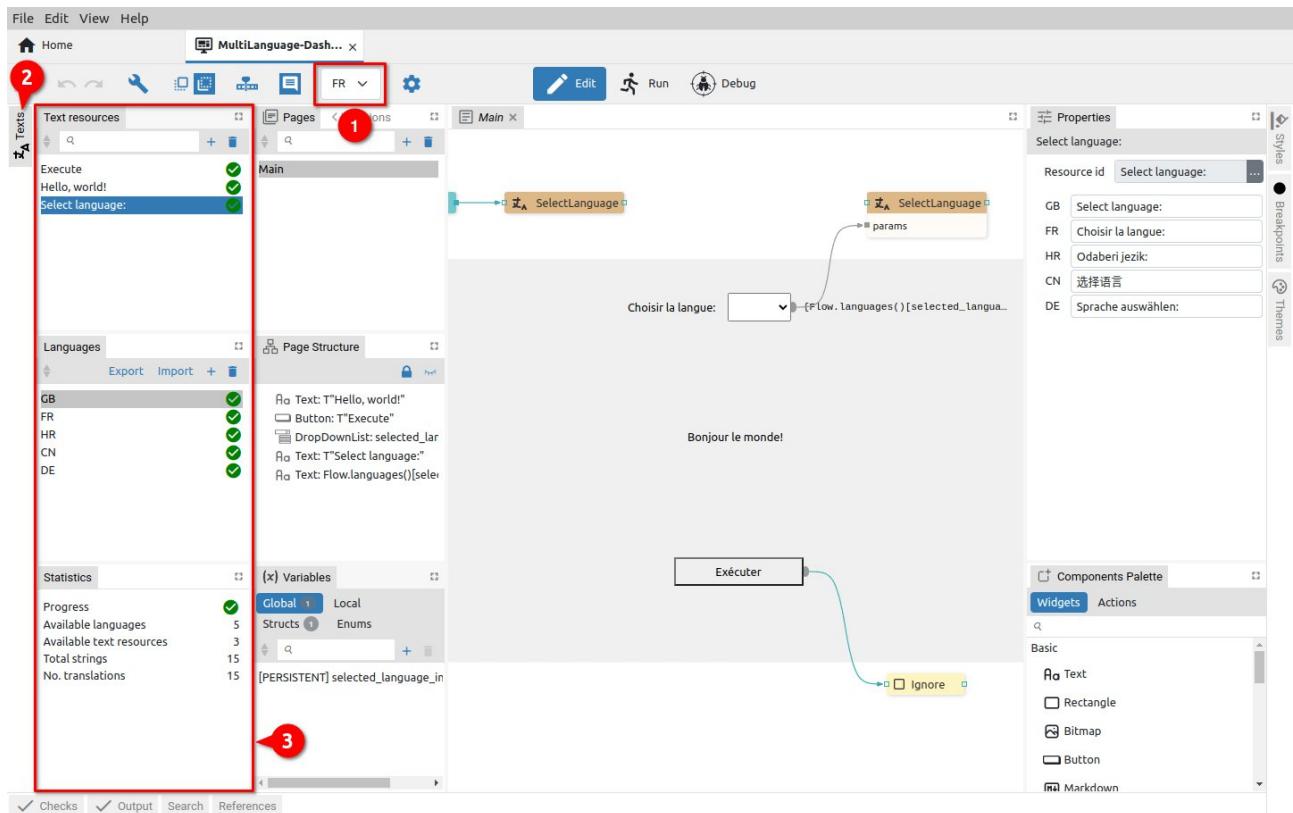


Fig. 31: Project language selector

P3.3. Feature buttons

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

P4. Project editor panels

P4.1. Panel items

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

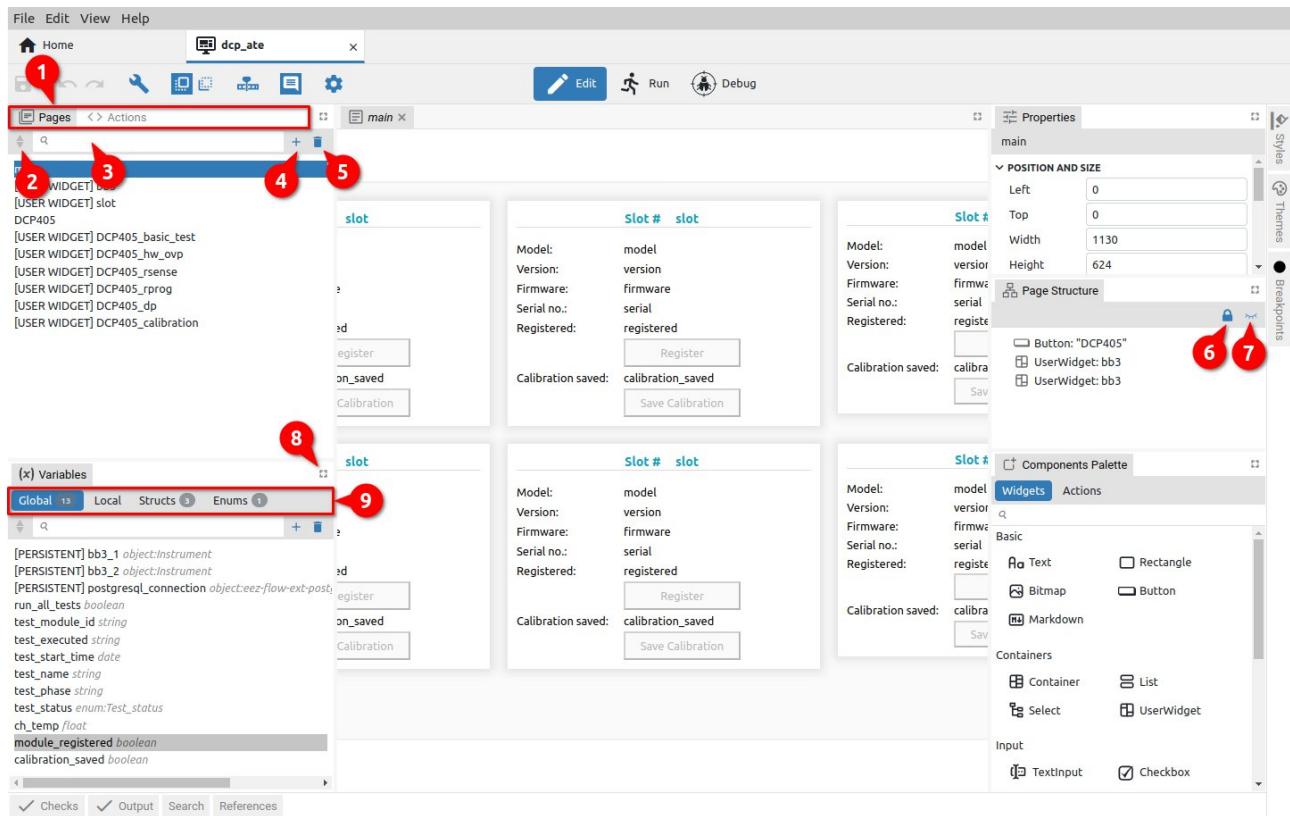


Fig. 32: Panel items

#	Item
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).

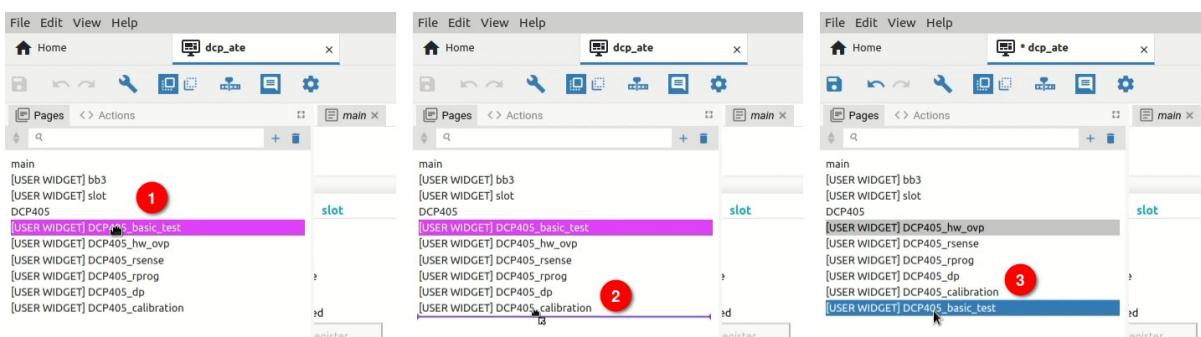


Fig. 33: Changing item position

IMPORTANT: make sure that the first page in the list is the one you want to be attached first when starting the project. The name of the page can be arbitrary (main in the example in Fig. 33).

- | | |
|----------------------|--|
| 3 <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. 34 shows the dialog box for adding a new page. |

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



Fig. 34: Adding a new item (Page)

- | | |
|----------------------------------|--|
| 5 <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 subtabs within a tabset. |

P4.2. Right-click menu

Right-click opens a context menu that generally contains options as in Fig. 35. The right-click menu for Widgets (Fig. 36) has a few more options and will be described in Chapter P7.2.2.

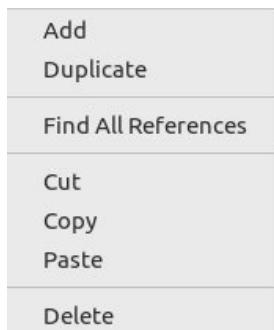


Fig. 35: Right-click menu (common)

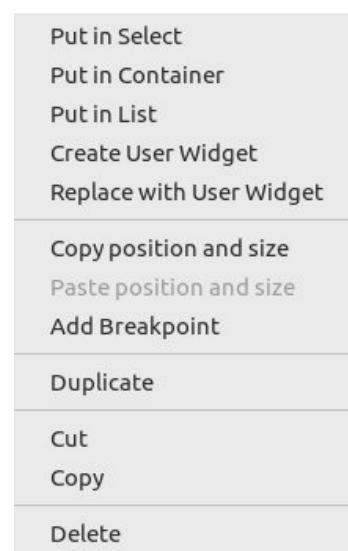


Fig. 36: Right-click menu (Widgets)

Option	Description
Add	Adding a new item. See the description in the previous subsection.
Duplicate	Duplication of items in the list. The name of the duplicated item will be given a numerical suffix: for example, <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.

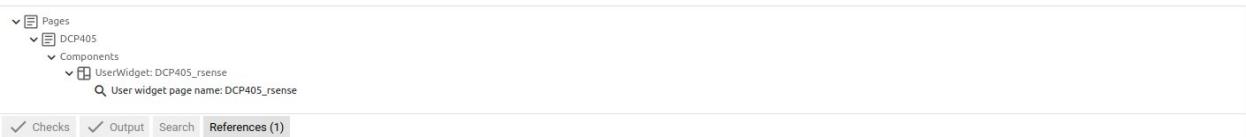


Fig. 37: Displaying the results of the Find all references operation

Cut	Cut (remove) item from list and copy it to clipboard.
Copy	Copy item to clipboard.
Paste	Adding items to the list from the clipboard. This option is hidden if the clipboard is empty.
Delete	Deleting the selected element from the panel. A deleted item can be restored with <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. 37).

P4.3.1. Search and Replace

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

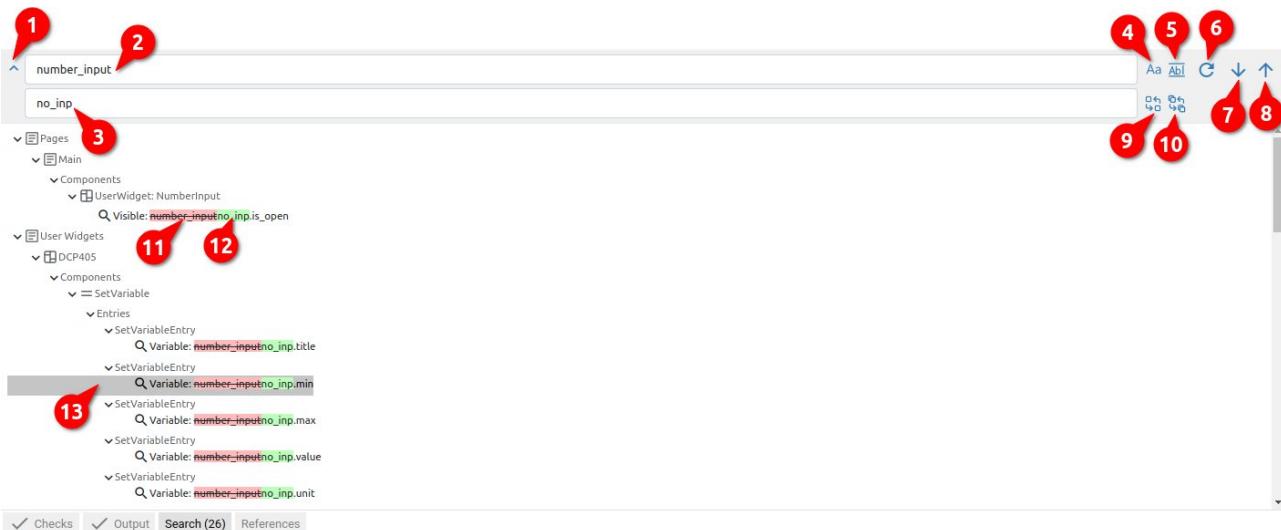


Fig. 38: Search and Replace panel

#	Item	Description
1	<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. 39).

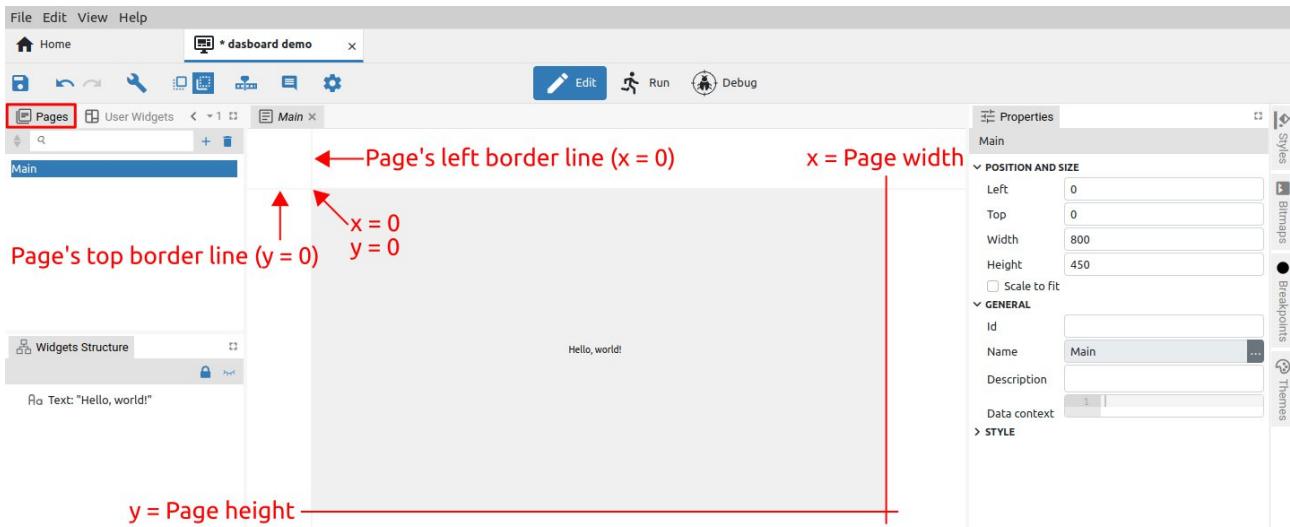


Fig. 39: Display of the page in the editor

P5.1.2. User Actions

The User Actions editor allows editing the selected Action from the User Actions panel (Fig. 39).

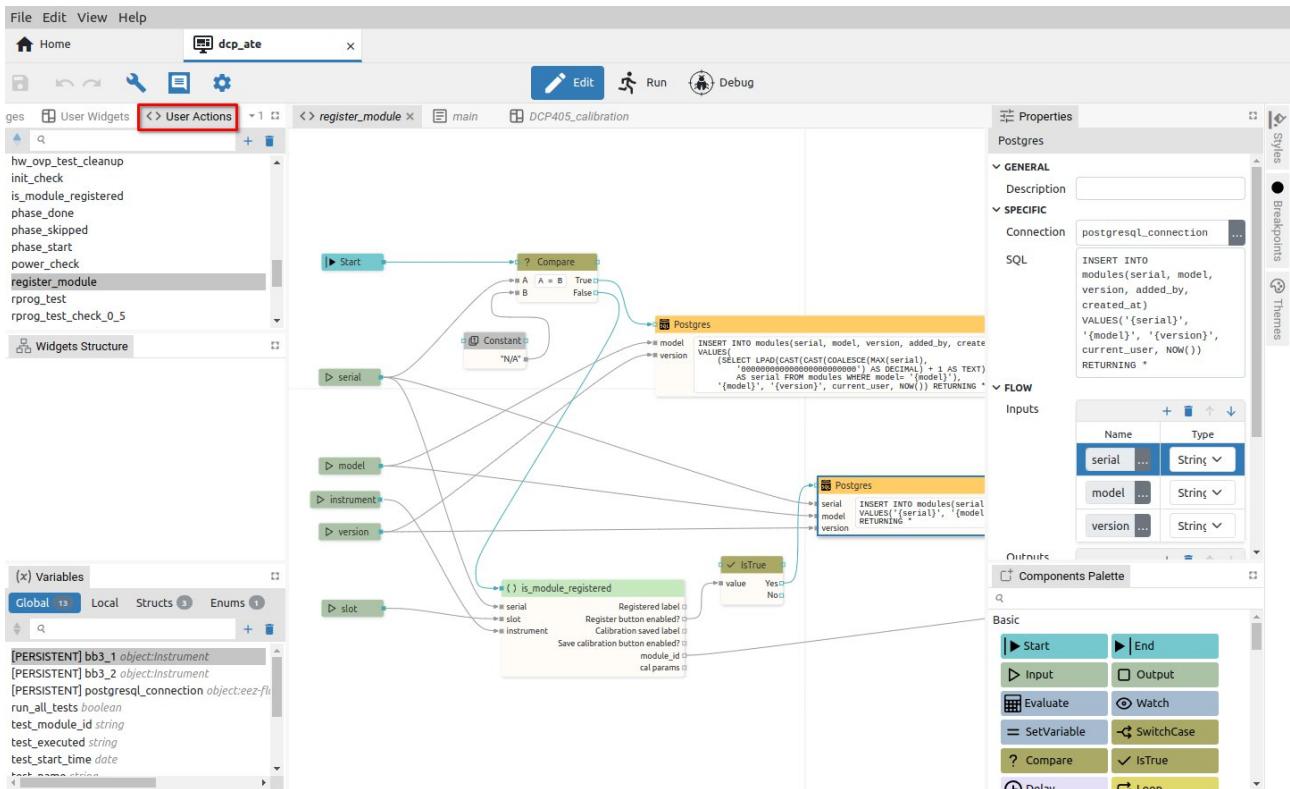


Fig. 40: User Actions editor page

P5.1.3. User Widgets

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

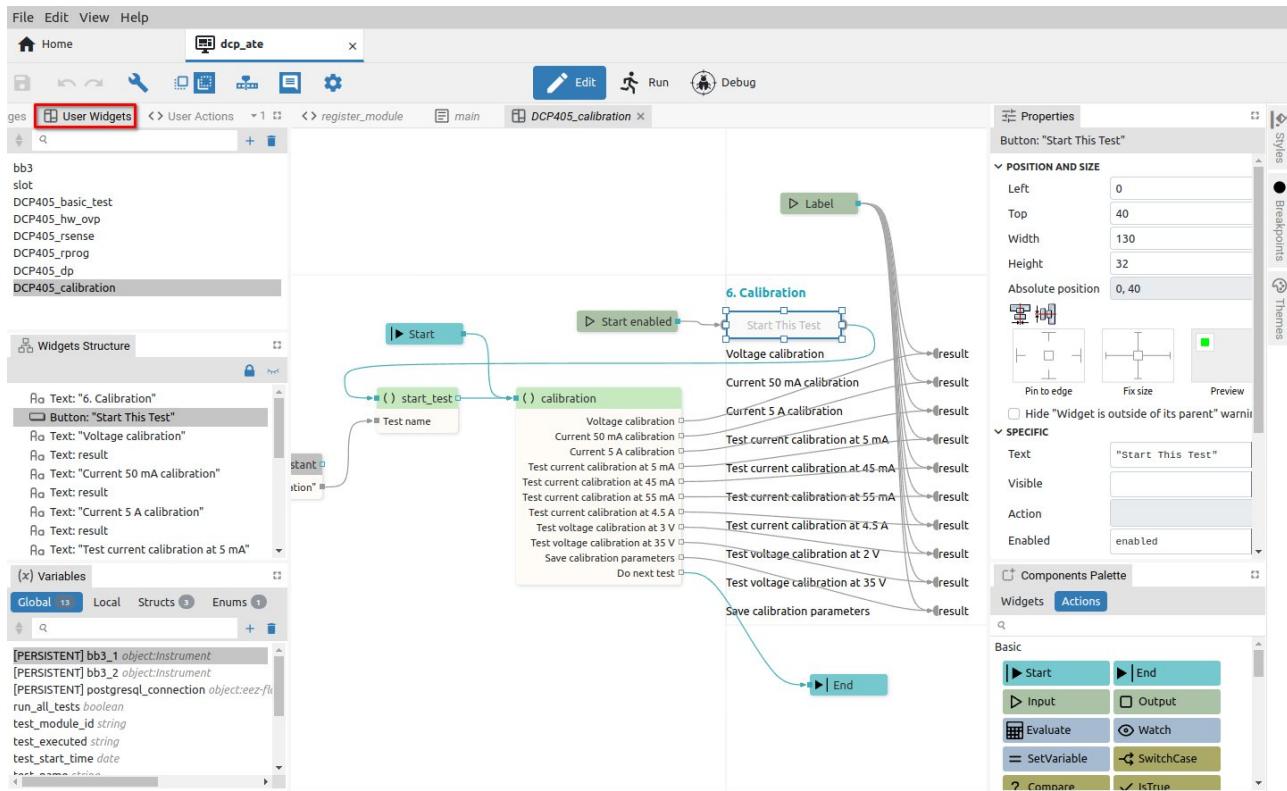


Fig. 41: User Widgets editor page

P5.1.4. Font editor

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

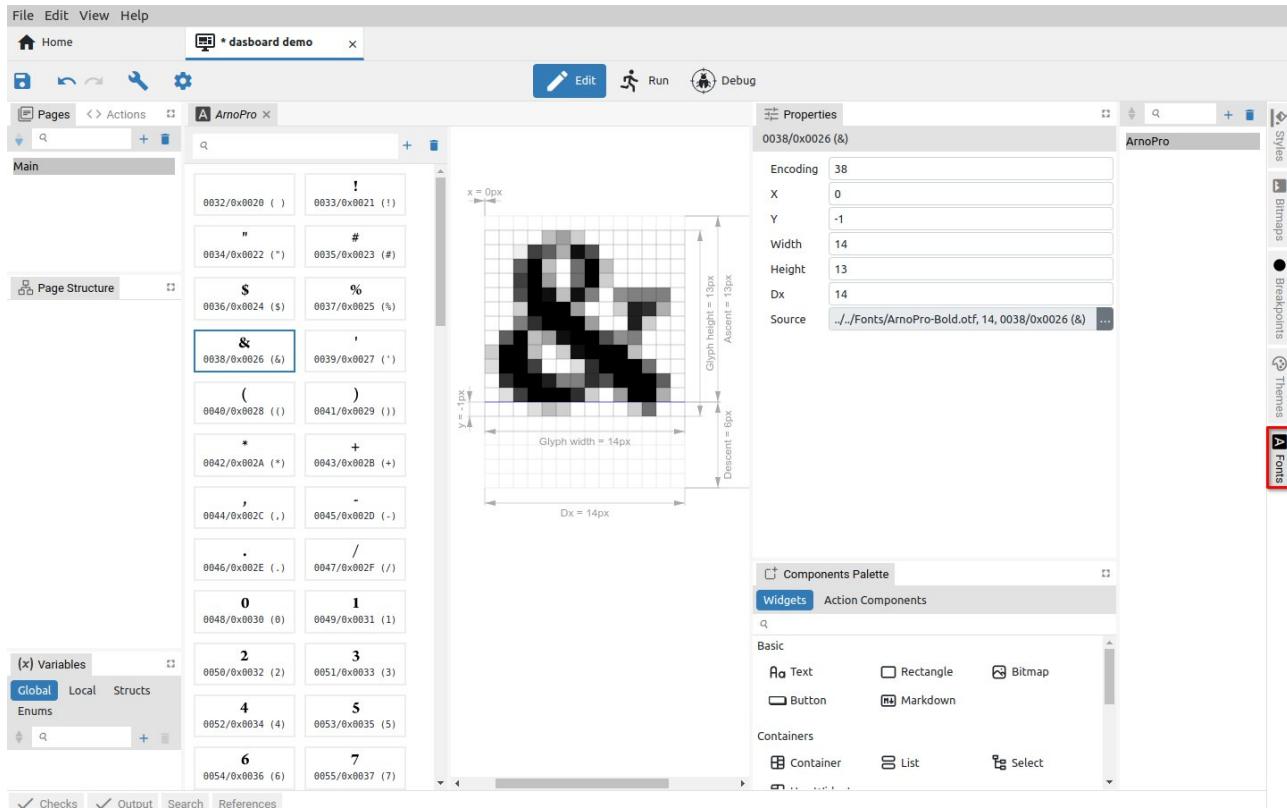


Fig. 42: Font editor page

P5.1.5. Shortcuts (EEZ-GUI only)

An *EEZ-GUI* project that includes Instrument Extension definitions (*IEXT defs* feature) can also have the *Shortcuts* feature enabled. In this case, the *Shortcuts* icon appears in the toolbar, which is used to display the page for defining shortcuts in the editor tabset (Fig. 43).

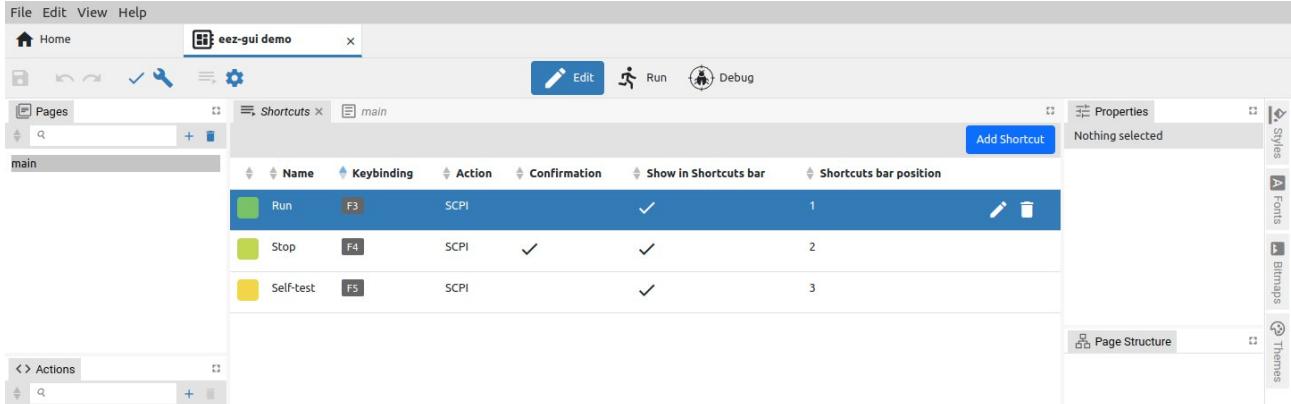


Fig. 43: Project shortcuts page

P5.1.6. MicroPython (EEZ BB3 only)

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

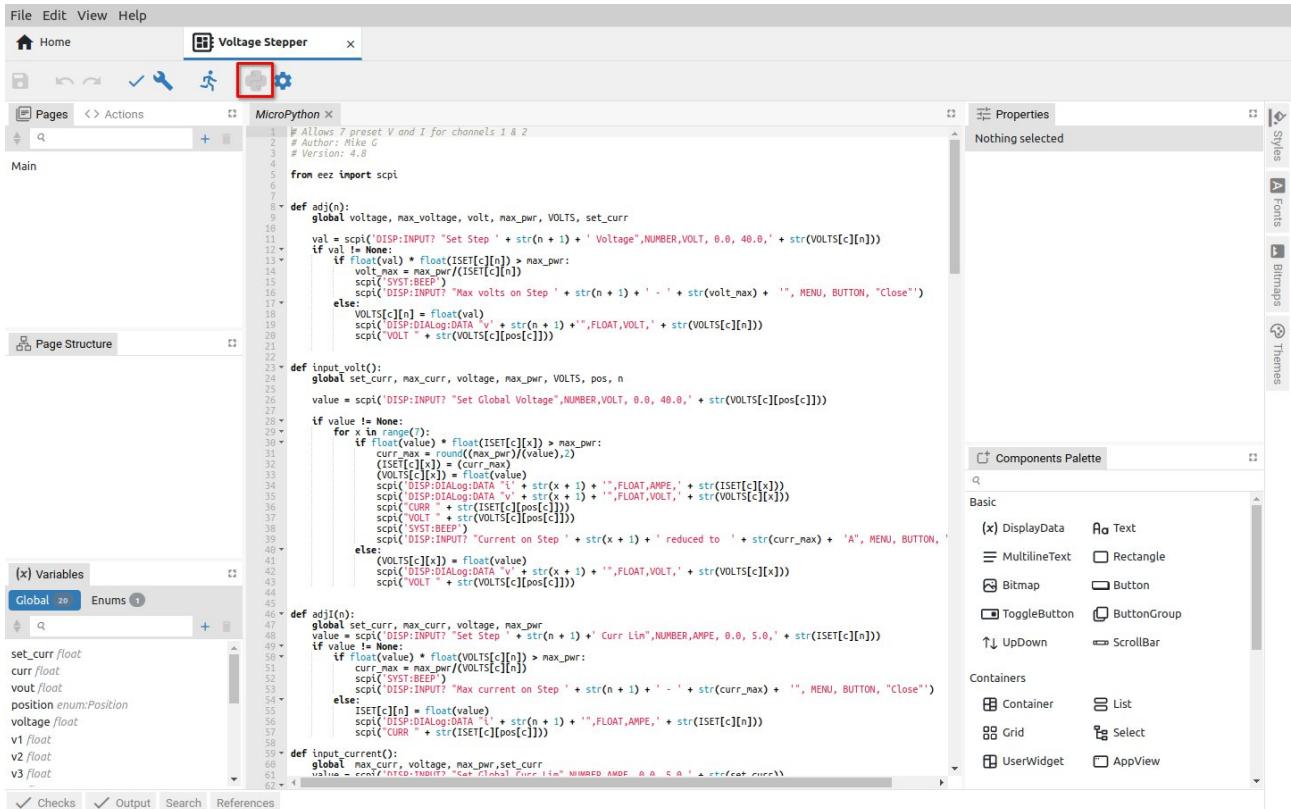


Fig. 44: MicroPython editor page

P5.1.7. Readme

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

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

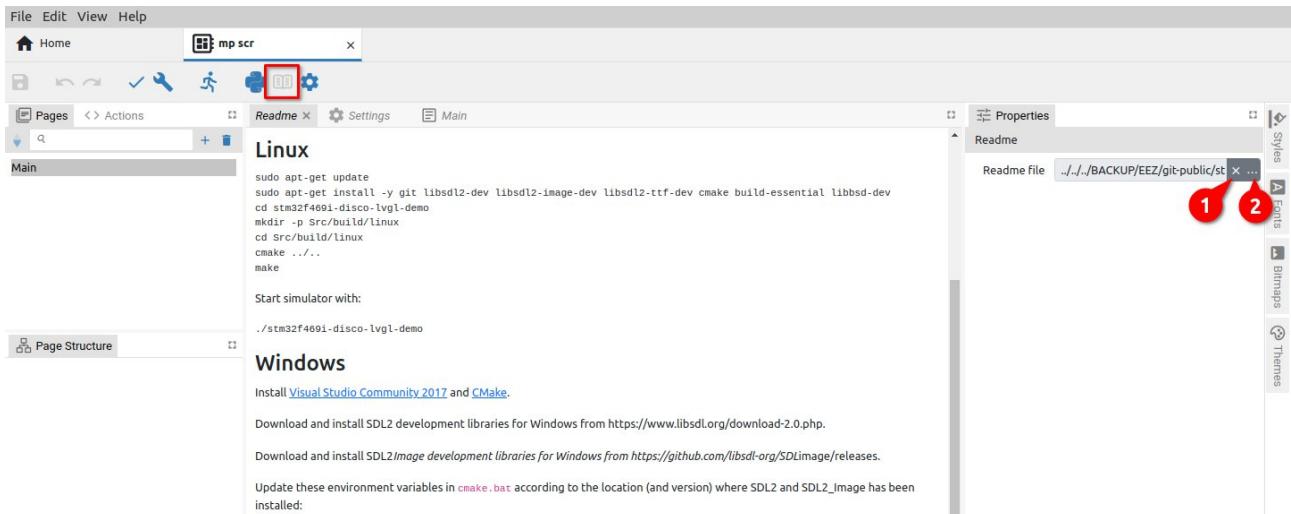


Fig. 45: Project Readme file

P5.1.8. Settings

The Settings page is used to edit the global parameters and features of the project (Fig. 45).

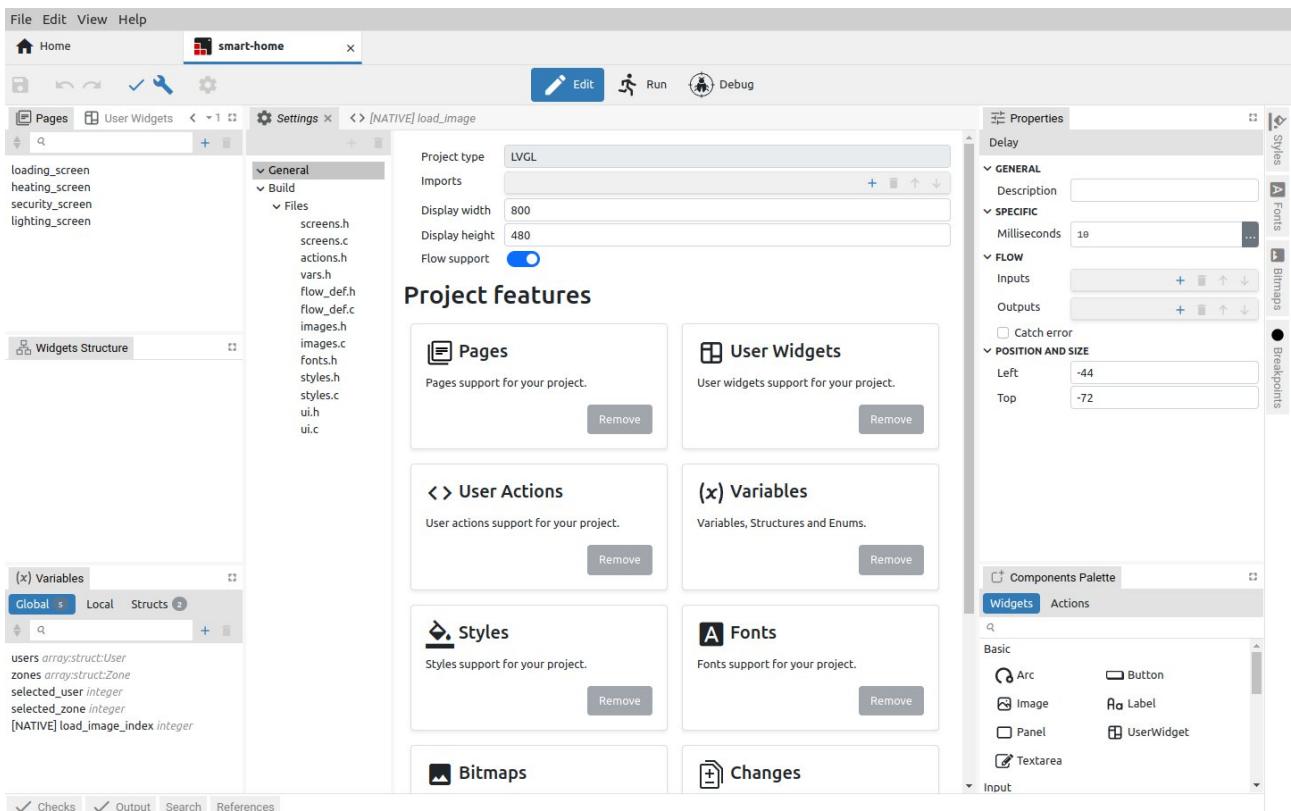


Fig. 46: Project settings page

P5.2. Viewers

P5.2.1. Page viewer

In *Run* mode, it is possible to see only the viewer of the currently active page (Fig. 42). In *Debug* mode, pages and Actions cannot be edited, so editors effectively become viewers (Fig. 40).

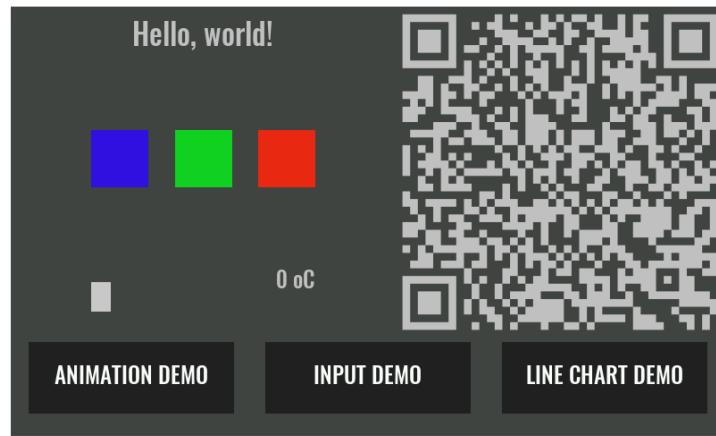


Fig. 47: Page view in Run mode

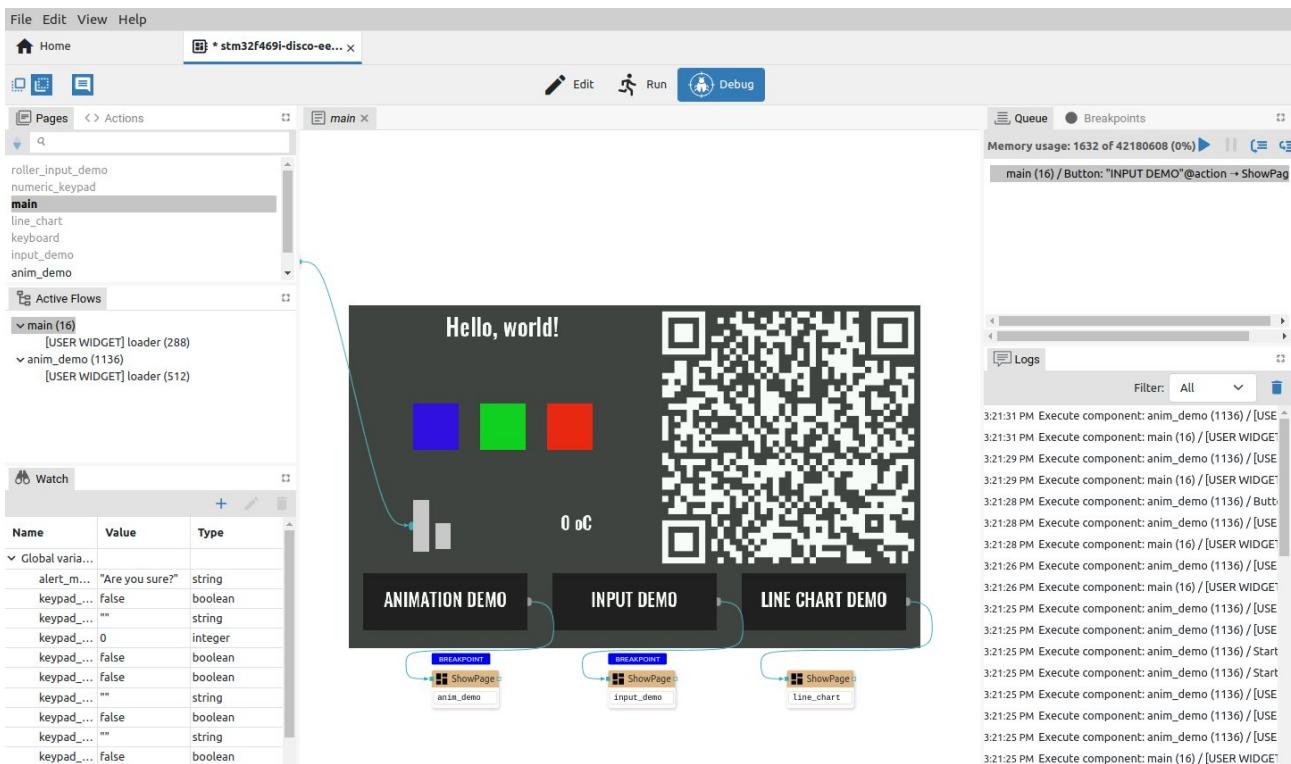


Fig. 48: Page view in Debug mode

P5.2.2. Action viewer

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

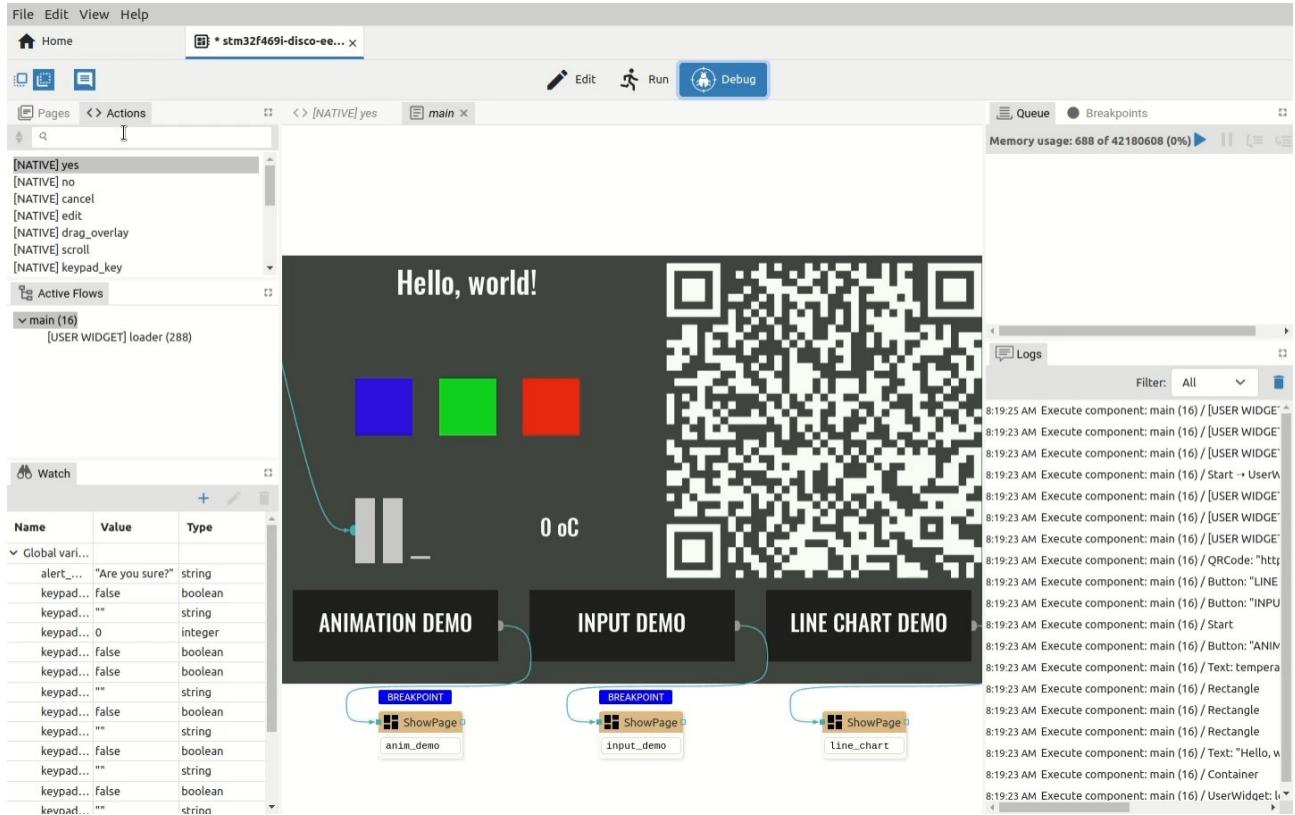


Fig. 49: Action viewer

P6. EEZ Flow

P6.1. EEZ Flow basic concepts

EEZ Flow is used to add programming logic to a project which enables programming using 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. 50).

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

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

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

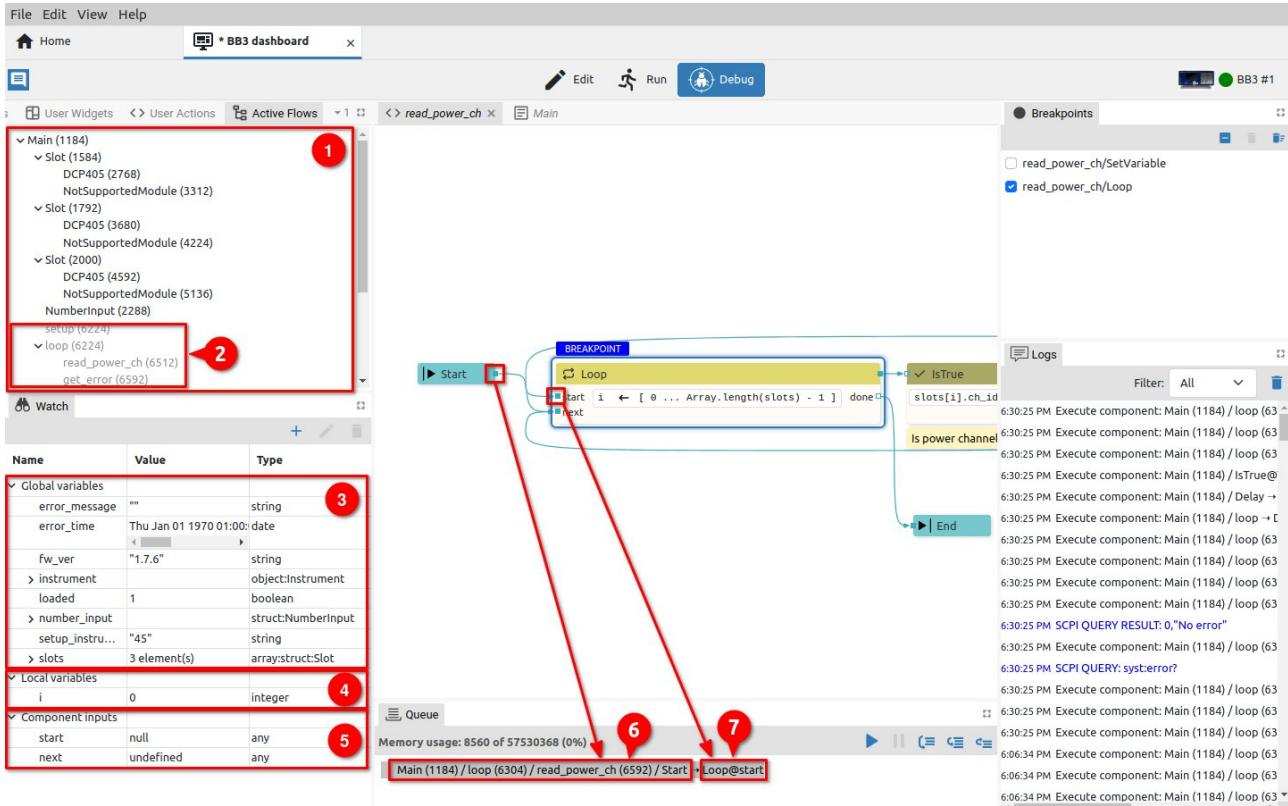


Fig. 50: Flow execution monitoring

When the project is executed in *Debug* mode, we can monitor in the *Active Flows* panel which Flows are active (1) and, as well as those that have finished execution, are no longer active, i.e. they are no longer in the execution queue (2). In the example in Fig. 50 we see that the *Main* page has one active Flow and under it are all the active Flows for the Widgets located in the *Main* page: we have three *Slot* Widgets, and each of them has its own two active Flows under it (*DCP405* and *NotSupportedModule*).

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

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

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

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

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

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

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

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

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

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

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

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

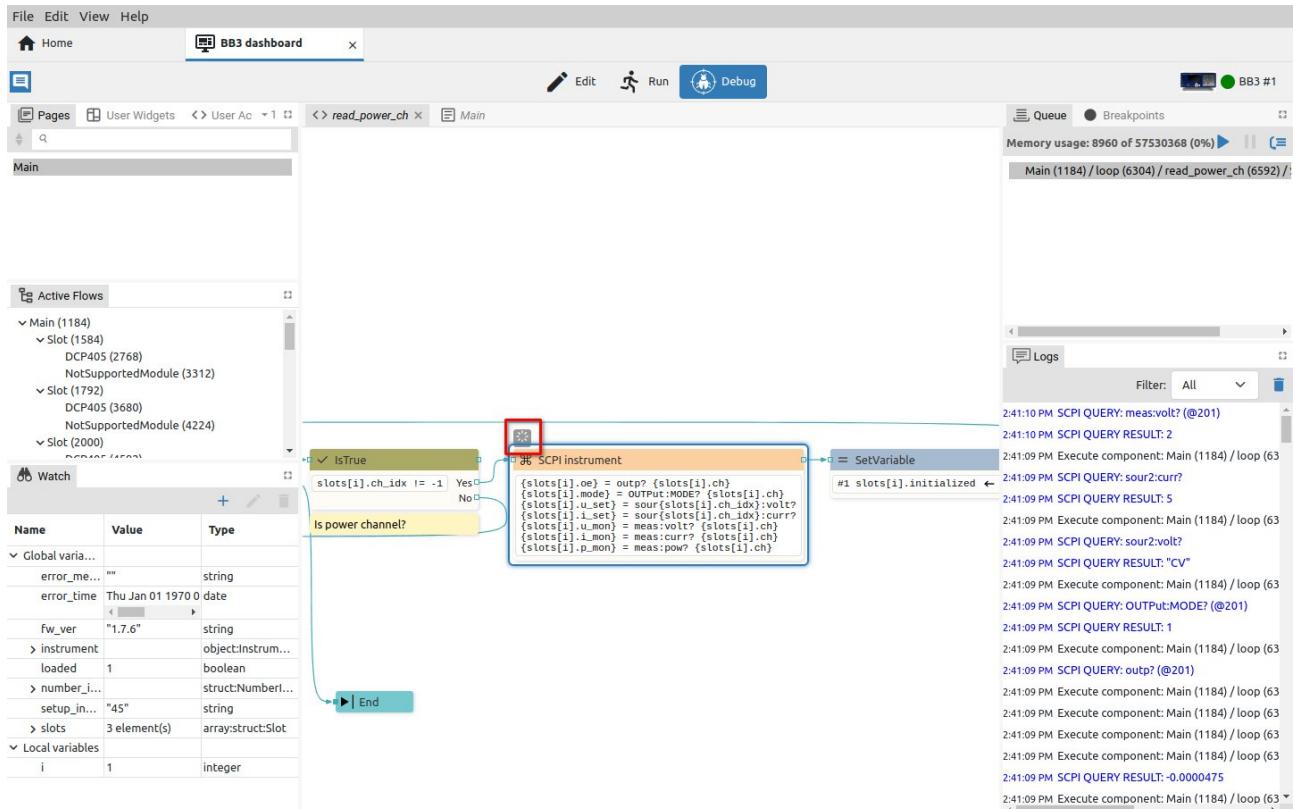


Fig. 51: Indication that the component preserves internal states

Components that preserve internal state, i.e. whose execution takes a long time, are also marked with a special icon in the debugger (Fig. 51). Example of such components: Loop, Delay, SCPI, etc. Such components, when they have done part of their work, can put themselves back in the queue. For

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

P6.3. Flow examples

Flow execution will depend on the way components are connected. In Fig. 52 shows four simple Flows that contain User Actions whose inputs are triggered in different ways. Fig. 53 also shows the final execution results when the defined string will appear upon startup (4), with a defined delay (1) (3) or will not be displayed at all due to incorrect connection (2).

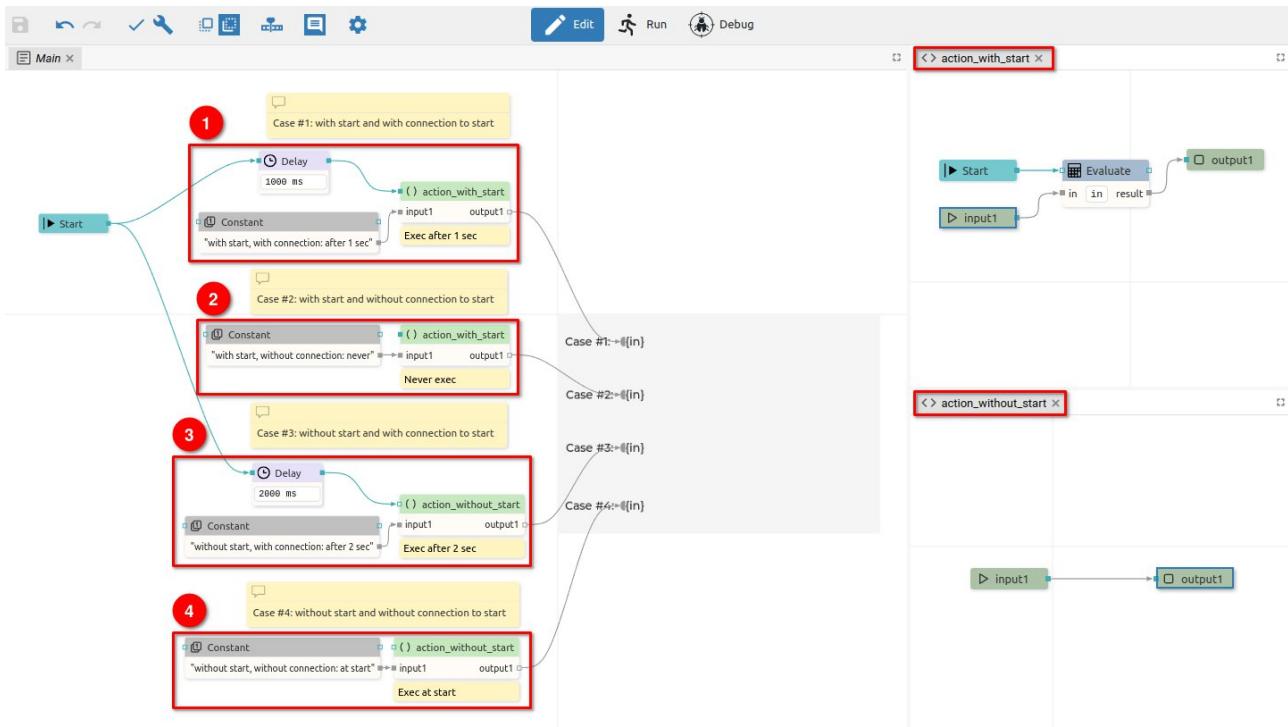


Fig. 52: Flow execution examples

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

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

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

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



Fig. 53: Execution results

P7. Project editing

Designing the graphical layout of the page is possible by simple drag & drop of one or more Widgets from the *Components palette*. The first step is to click on the Widget and hold which will change the cursor (Fig. 54)

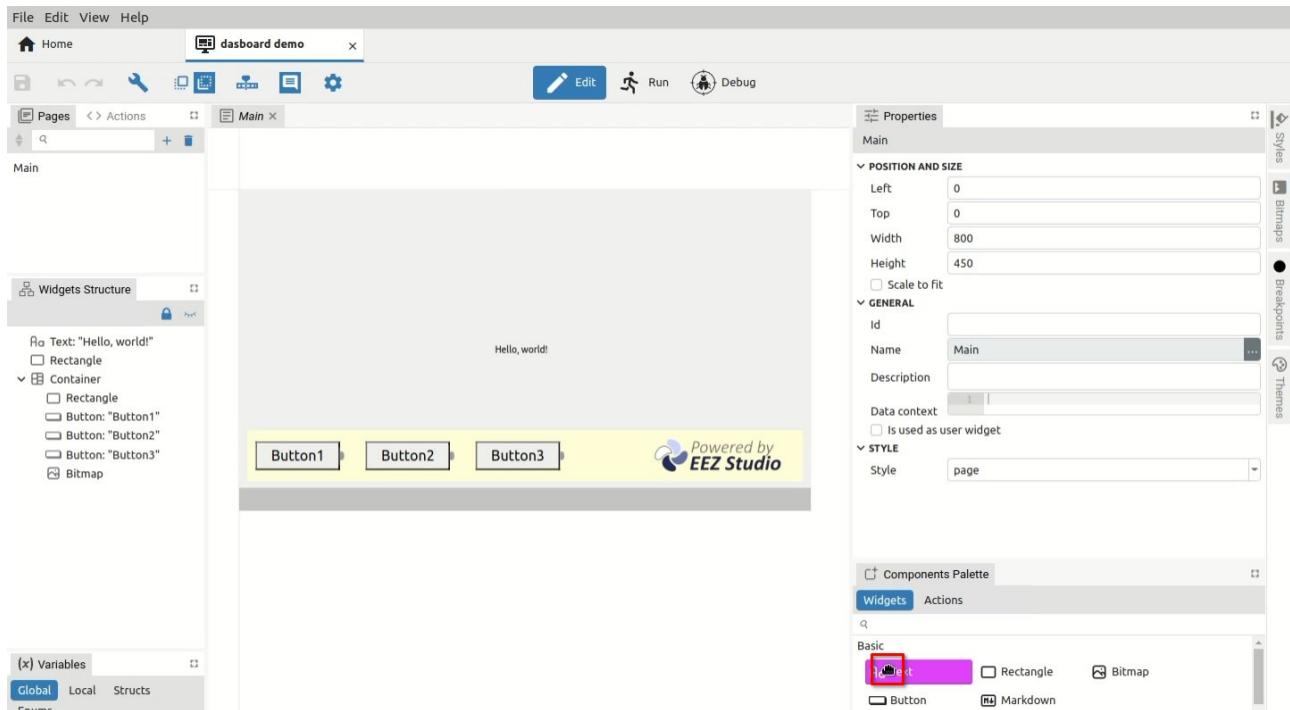


Fig. 54: Selection of Widget to add to the page

When the Widget is dragged into the editor area, auxiliary snap lines will appear immediately, which will make it easier to place the Widget in the desired place. Snap lines are displayed depending on nearby objects. If the position of the new Widget is not close enough to the others, or if it is the first Widget added to the page, the snap will be possible towards the page itself as in the examples in Fig. 55 where snap lines appear for horizontal or vertical centering.

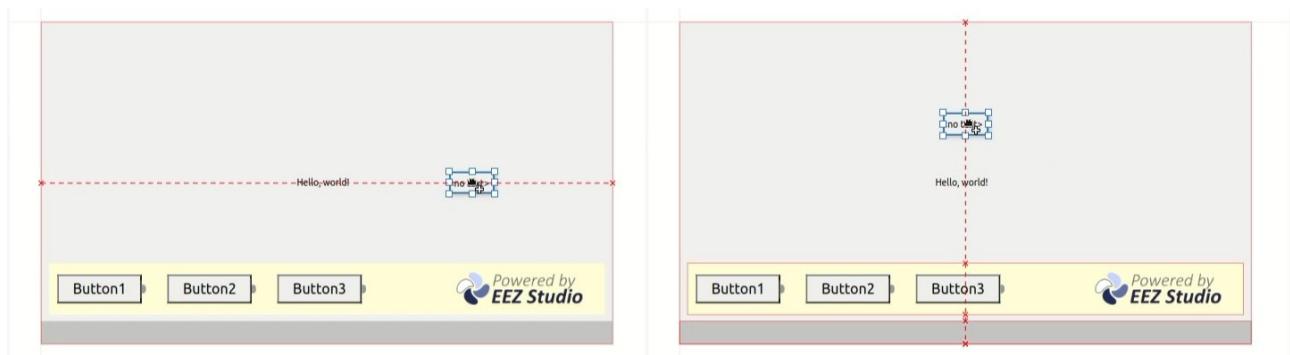


Fig. 55: Snap lines for centering in the page



Fig. 56: Snap lines for positioning versus other Widgets

Fig. 56 shows examples of snap lines to the edges of other Widgets on the page.

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

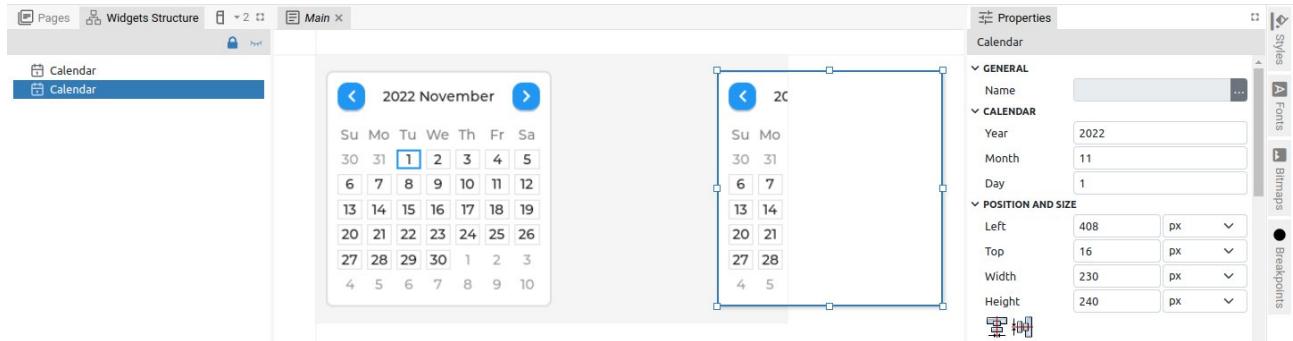


Fig. 57: Widget positioned outside the page

Please note that if the Widget in the *EEZ-GUI* and *LVGL* projects is set to protrude from the page (Fig. 57), the part that is outside the page will not be visible.

Widget positions can be freely changed and this can be done for one or more selected Widgets. It is possible to select multiple Widgets (Fig. 58): both in the page editor (1) or in the *Page Structure* panel (2). In both cases, information will appear in the *Properties* panel that multiple Widgets are selected (3). When selecting in the *Page Structure* panel, it is possible to use the SHIFT key to select a continuous sequence or CTRL to add individual Widgets to the selection.

There are two methods of multiple selection in the editor: selecting Widget by Widget while holding down the SHIFT key, and the second method is the so-called rubber band selection shown in Fig. 59 when selecting the area that will include the Widgets we want to select.



Fig. 58: Multiple Widgets selection

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

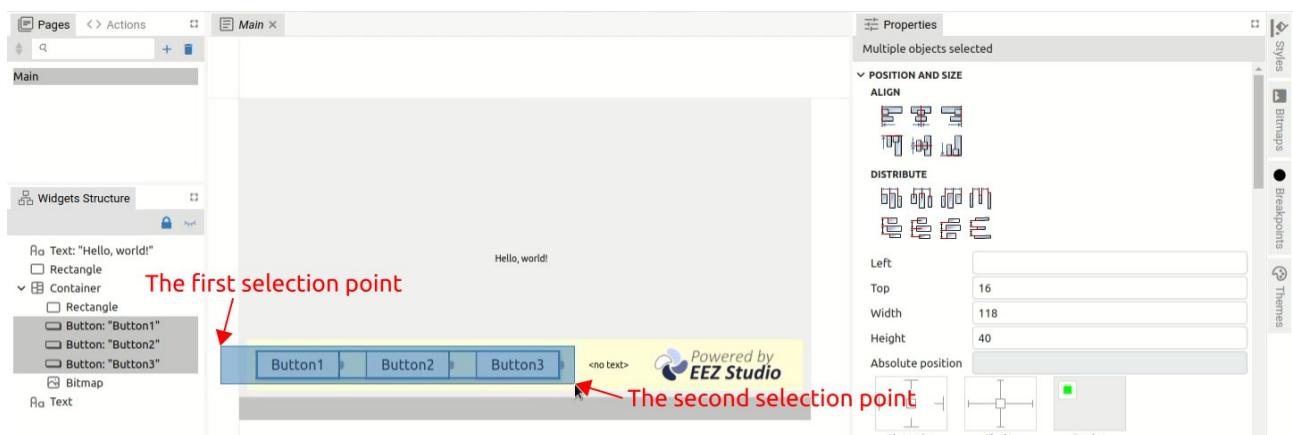


Fig. 59: "Rubber band" selection

As shown in Fig. 58, it is possible to change *Properties* for multiple Widgets. Specific to multiple selection is the *Position and size* section when the *Align* subsection gets more options, and a complete *Distribute* subsection appears.

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

Align

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

Distribute

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

The page in the editor can be resized or set to the default size (1:1) or scrolled horizontally and vertically within the editor. For these operations, the mouse wheel is used in combination with the SHIFT and CTRL keys, as shown in Fig. 60.

Operation	Description
<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.

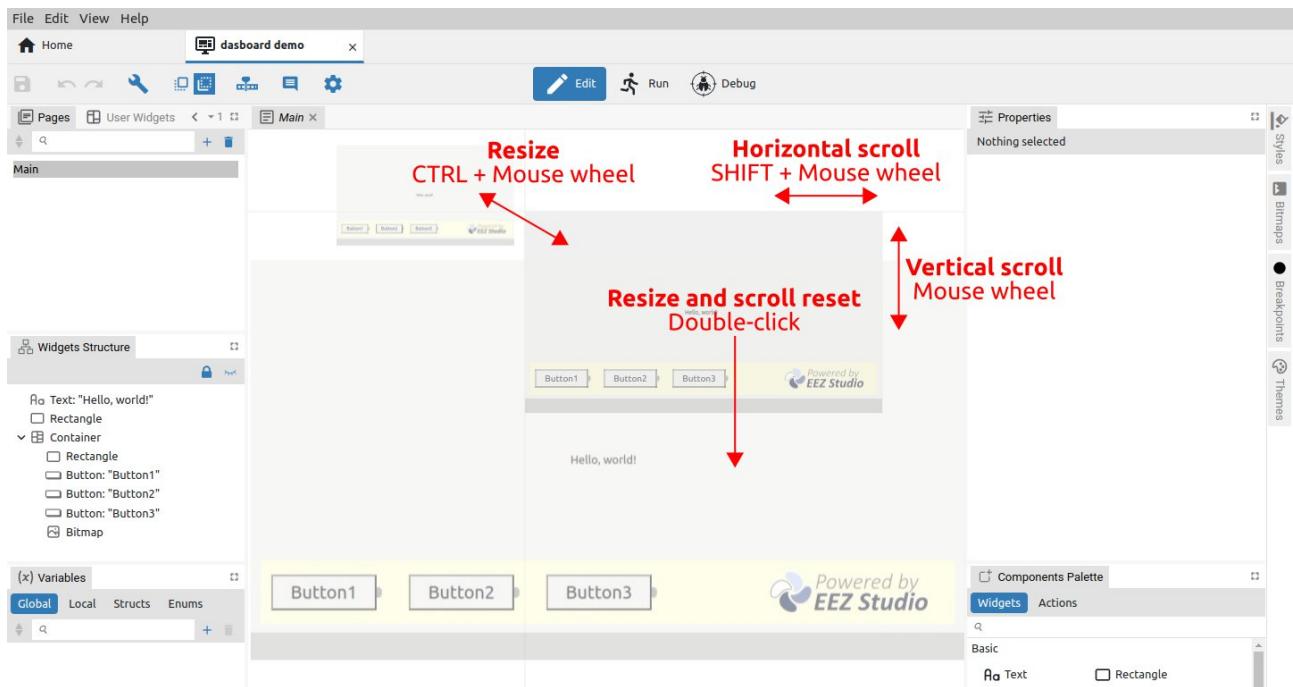


Fig. 60: Page resize and scroll

P7.1.1. Connecting Flow components

The connection (Flow line) between Flow components (Widgets and Actions) is used to define the flow of execution. In Fig. 55 shows how the output of one Action is connected to the input of another Action. When we position ourselves on the output (1), the color of its background will change, and if we continue with the mouse drag, the Flow line (2) will appear. When the cursor reaches the input of the second component, the Flow line will change color to green (3). Now we can release the mouse when the connection between the two components will be established (4). In case of moving one of the components, the Flow line will move with it.

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

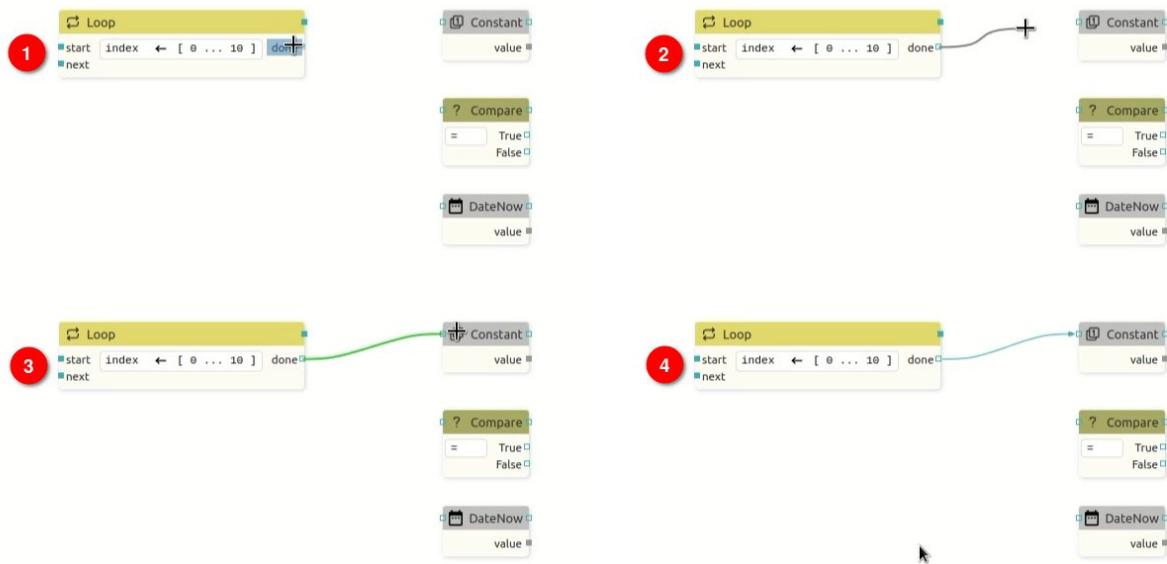


Fig. 61: Connecting the output to the input of the Widget

Adding a Flow line is also possible by starting from the input of one component to the output of another. In Fig. 62 shows how to connect the input of one Action to the output of another.

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

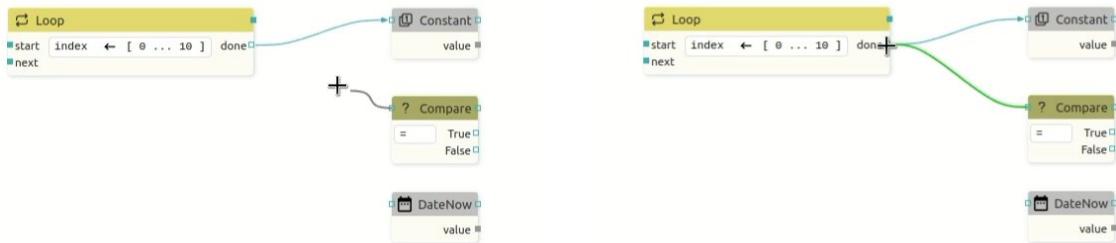


Fig. 62: Connecting the input to the output of the Widget

In case we have multiple Flow lines that end at a single input or output, it is possible to move them all to another input or output if necessary. Example in Fig. 63 shows the multiple Flow line moving from one output to another. First, we need to get to the output when the color of the background and all affected flow lines (1) will change. Then we need to drag the mouse while holding SHIFT, and a copy of the selected lines will appear, and their end can now be moved as desired (2). When we reach a new output, the color of the flow lines changes to green (3) and when the mouse button is released, a new connections will be drawn.

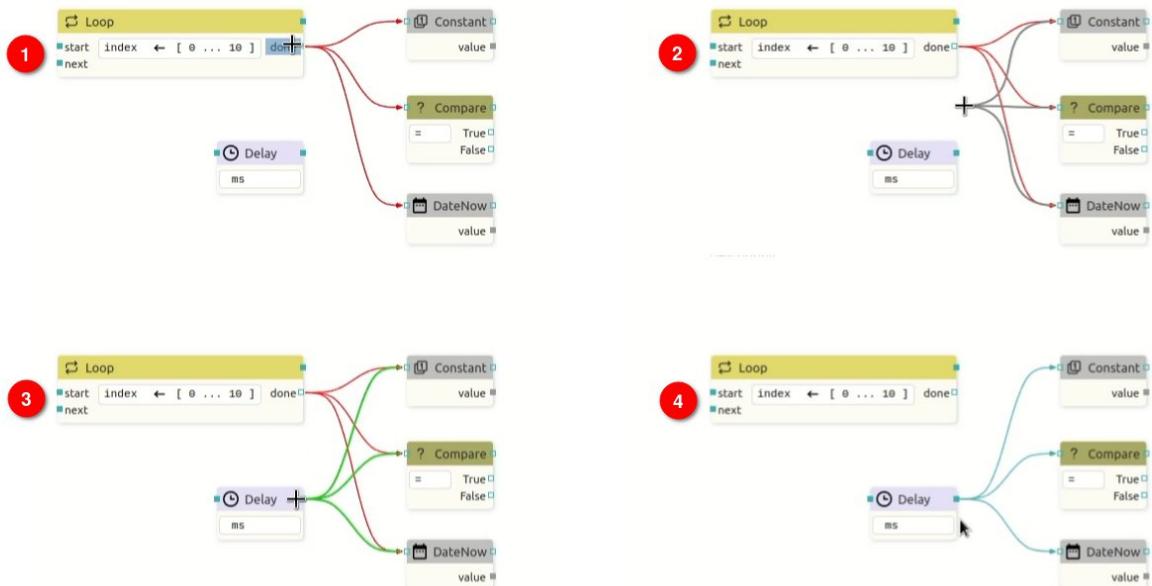


Fig. 63: Moving multiple connections

P7.1.2. Copy & Paste between two projects

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

P7.2. Working with Widgets

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

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

- *EEZ-GUI (Native)* – Widgets created for the purposes of creating the EEZ BB3 embedded GUI for the STM32 family of MCUs that support graphics (Fig. 64)
- *LVGL* – Widgets from the open-source embedded graphics library LVGL. They can only be used in a project of type LVGL. (Fig. 65)

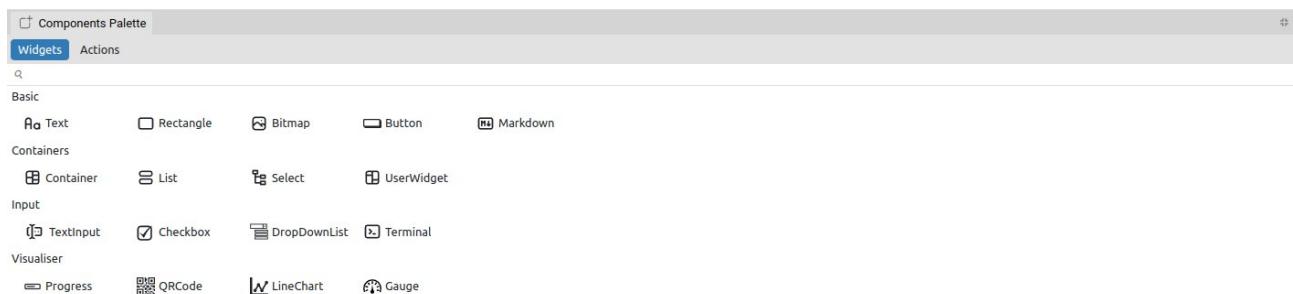


Fig. 64: Widgets palette for the EEZ-GUI project

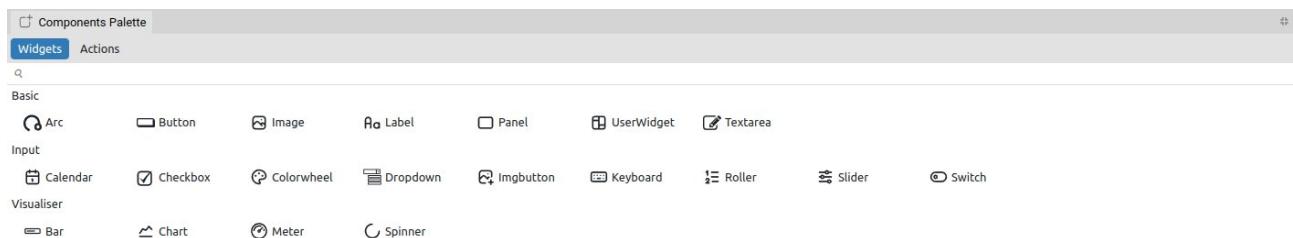


Fig. 65: Widgets palette for the LVGL project

P7.2.1. Widget component's items

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

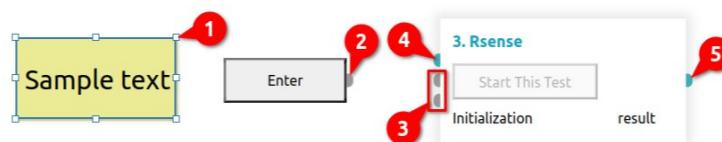


Fig. 66: Widget components items

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

5 Data Output

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

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

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

Table 2: User Widget's pin types

P7.2.2. Creating a User Widget

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

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

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

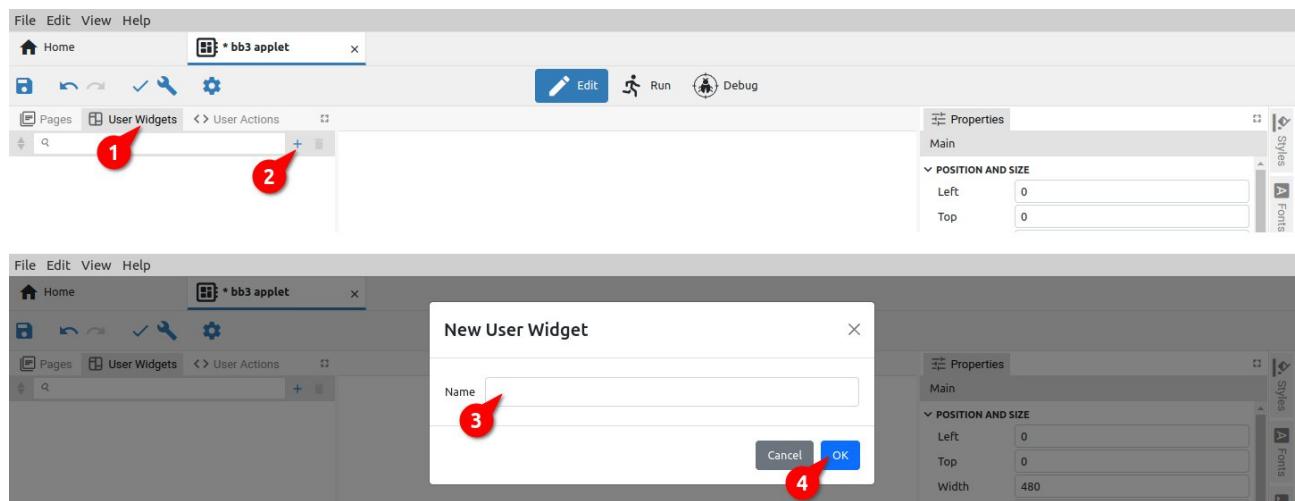


Fig. 67: Creating a new User Widget from the User Widgets panel

In the first case, select the *User Widgets* panel (1) and then the *Add* option (2), when a dialog box for entering the name of the User Widget will appear (Fig. 67).

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

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

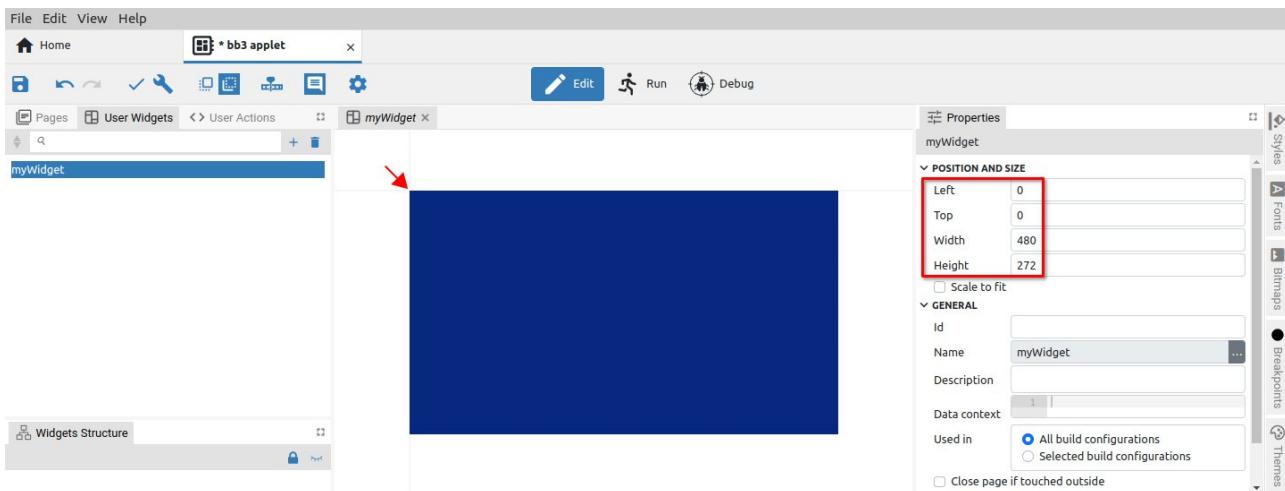


Fig. 68: Page editor of the newly created User Widget

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

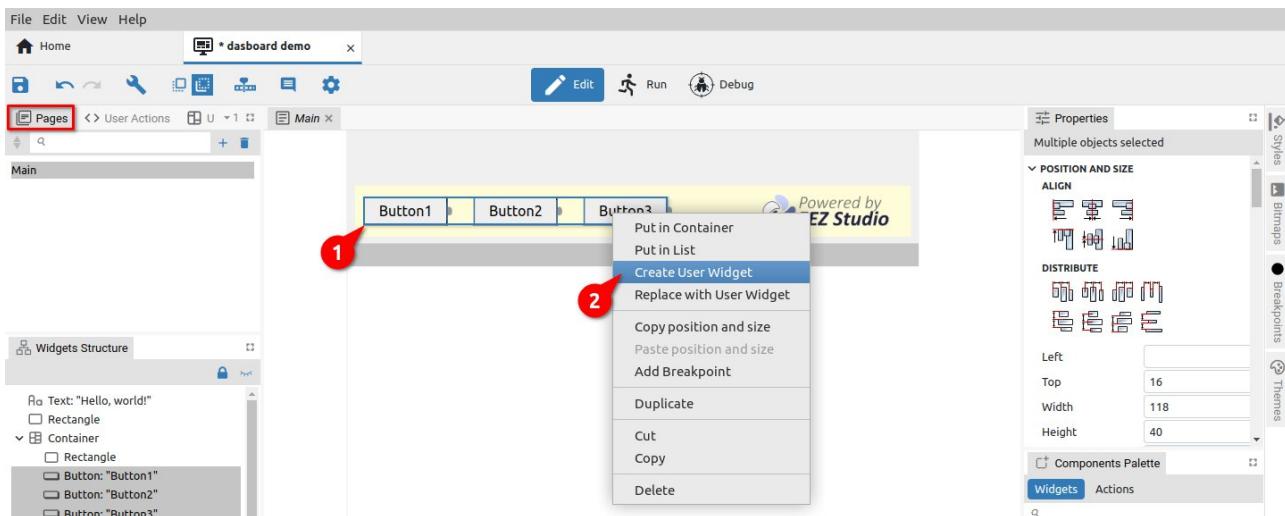


Fig. 69: Creating a new User Widget using the right-menu option

After successfully adding a new User Widget, it can be edited in the page editor (Fig. 70). Unlike the previous case, this Widget will have the dimensions of the original selection and the first component will be positioned at the starting point.

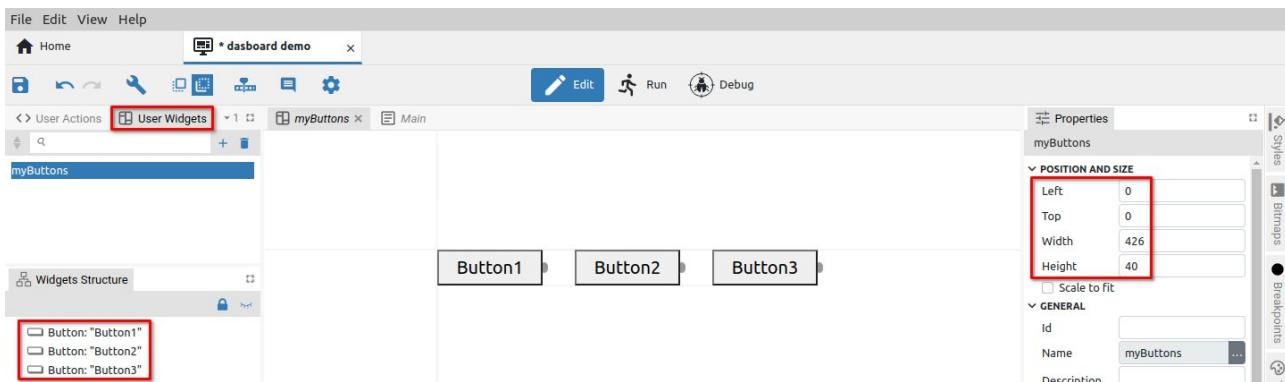


Fig. 70: The newly created User Widget in the editor

P7.3. Working with Actions

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

The Actions that come with EEZ Studio (i.e. built-in Actions) are located in the *Actions* subtab of *Components Palette* and are added to the editor with drag & drop and grouped for easy finding. The number of implemented Actions depends on the type of project. In Fig. 71 shows the Actions for the Dashboard project, and Fig. 72 for the LVGL project.

The Action can also be implemented in the EEZ Studio extension. An example of such an Action is *Postgres*, which is shown in the *eez-Flow-ext-postgres* group (Fig. 71).

EEZ Studio also allows defining User Actions. To edit them, we use the User Actions editor. All User Actions are also listed at the bottom of the Actions subtab (Fig. 72), from where they can be added to the project with drag and drop as any Action or Widget (1).

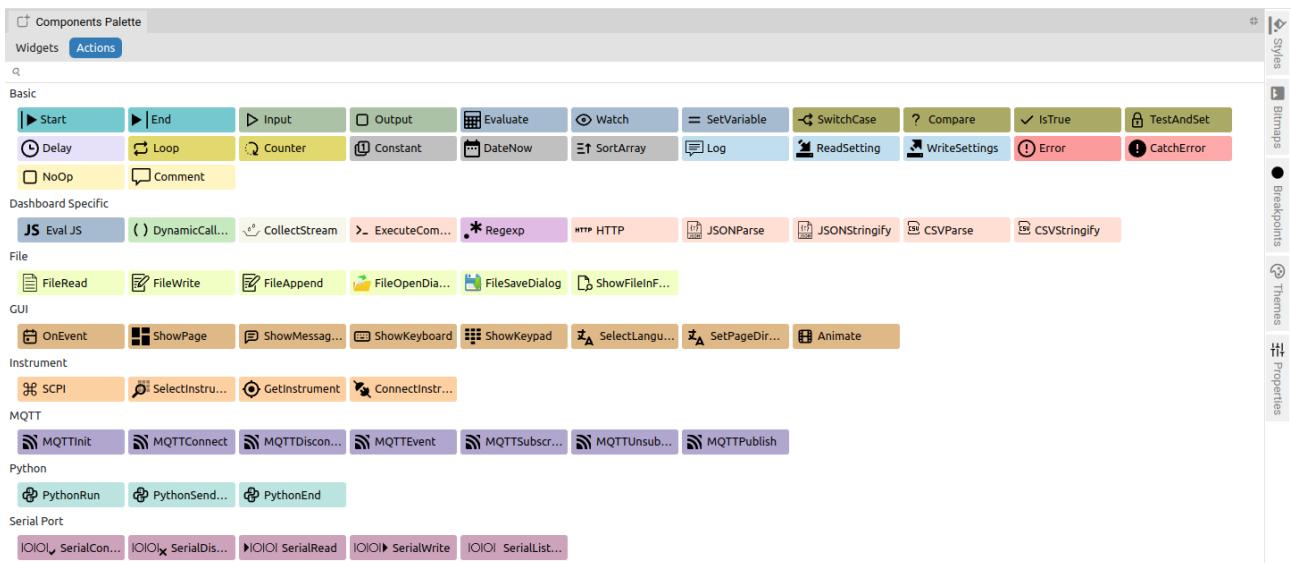


Fig. 71: Actions palette for the Dashboard project

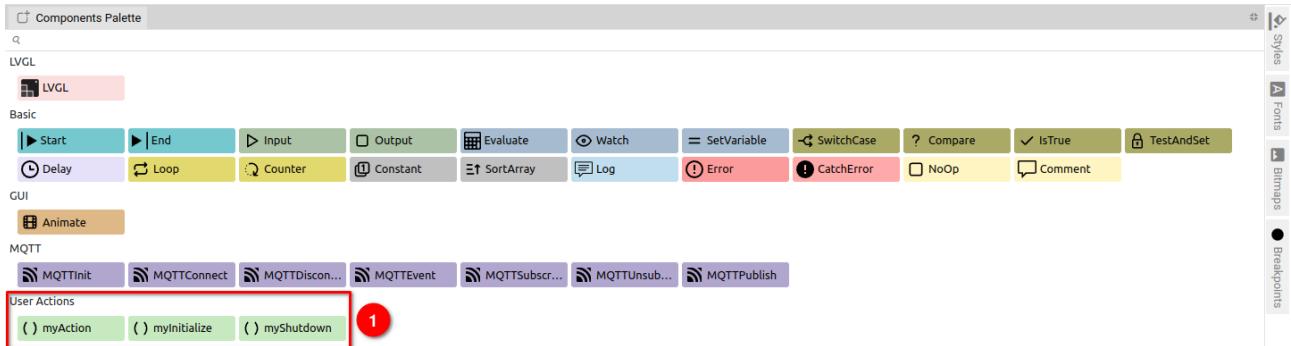


Fig. 72: Actions palette for the LVGL project

P7.3.1. Action component's items

Action component items are shown in Fig. 68 and described below.

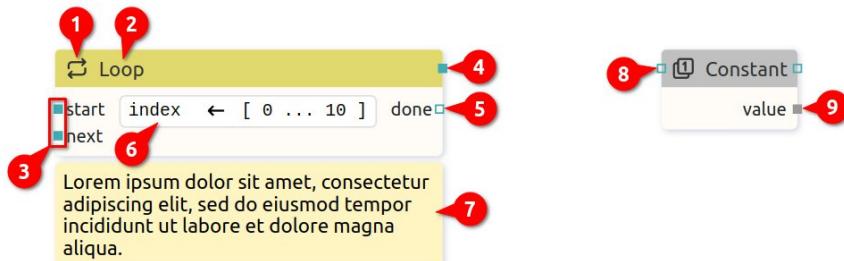


Fig. 73: Action components items

#	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.

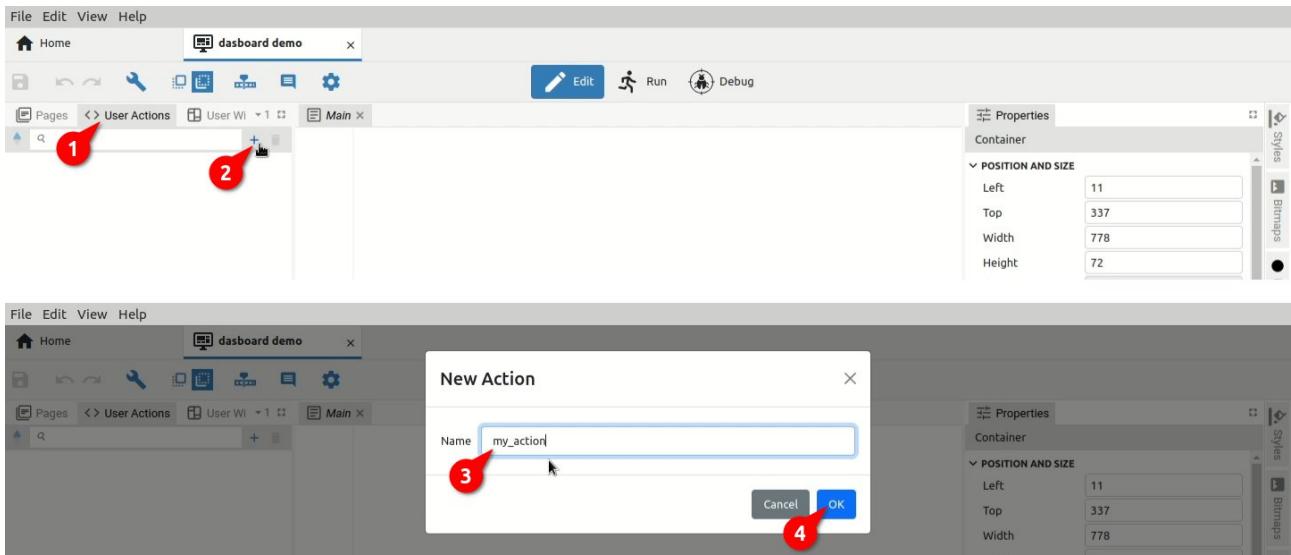


Fig. 74: Creating a new User Action

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

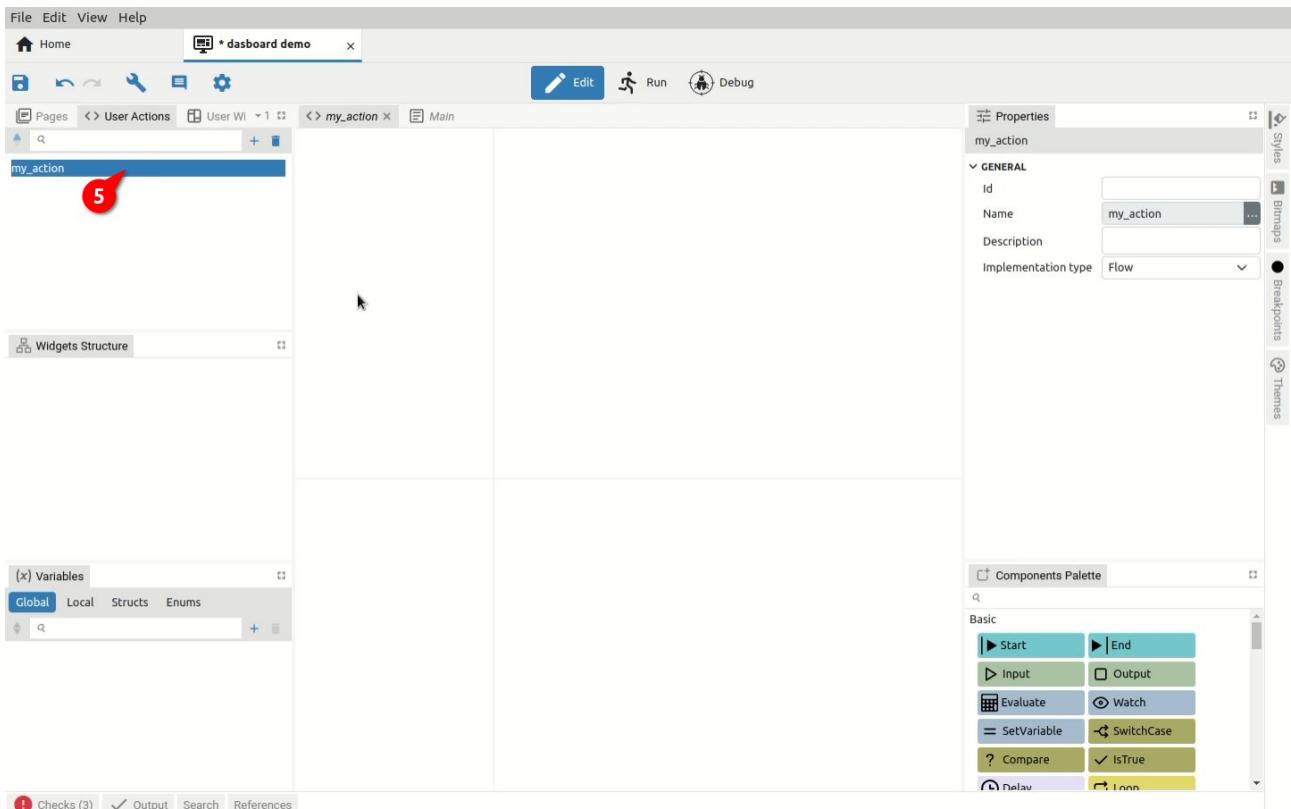


Fig. 75: Flow editor of the newly created User Action

Fig. 61 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.

EEZ Studio Reference Guide

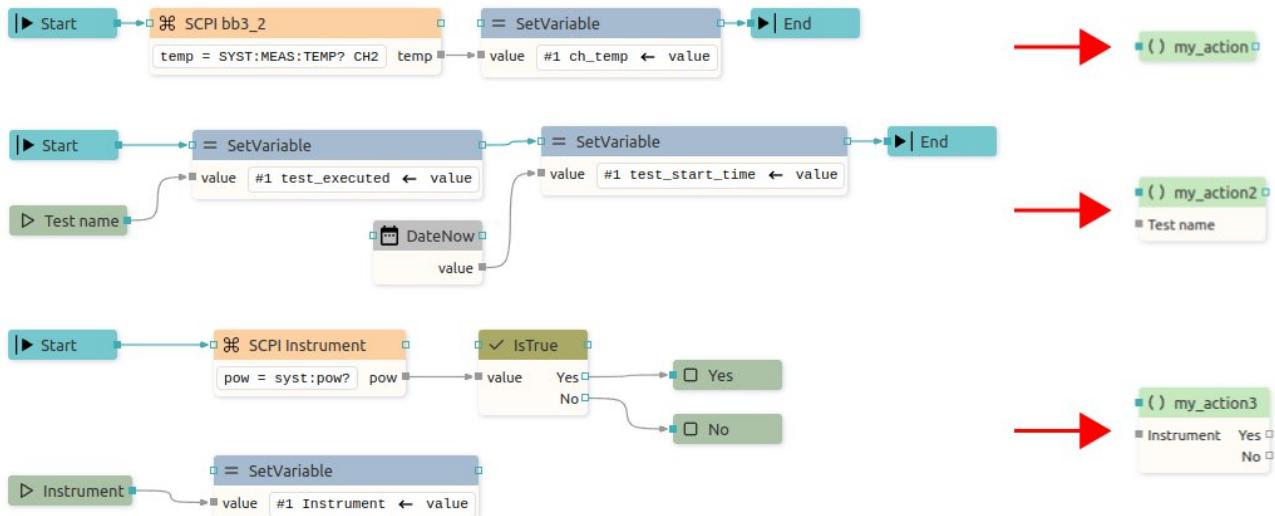


Fig. 76: User Action examples

P8. Variables

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

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

To add variables, use the Variables panel (Fig. 77), when a dialog box will open for defining the basic parameters of the variable (*Name*, *Type* and *Default value*).

To edit the parameters of the variable selected in the *Variables* panel, use the *Properties* panel. In Fig. 78, Fig. 79 and Fig. 80 shows *Properties* panels for different types of variables from different types of projects.

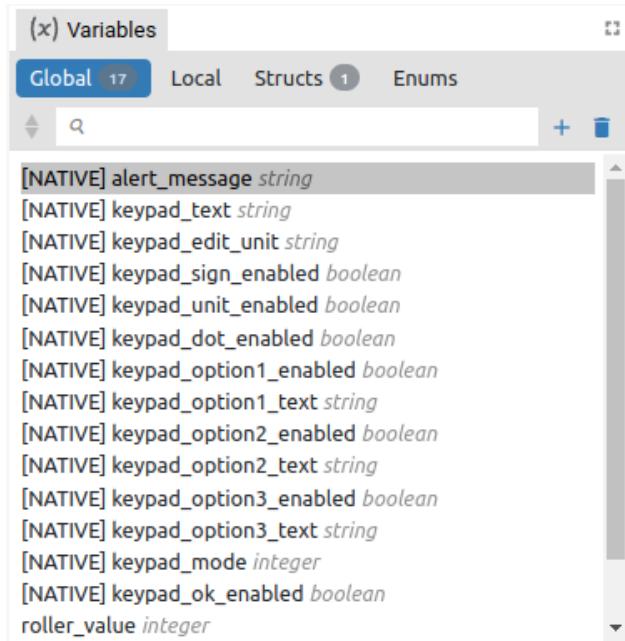


Fig. 77: Variables panel

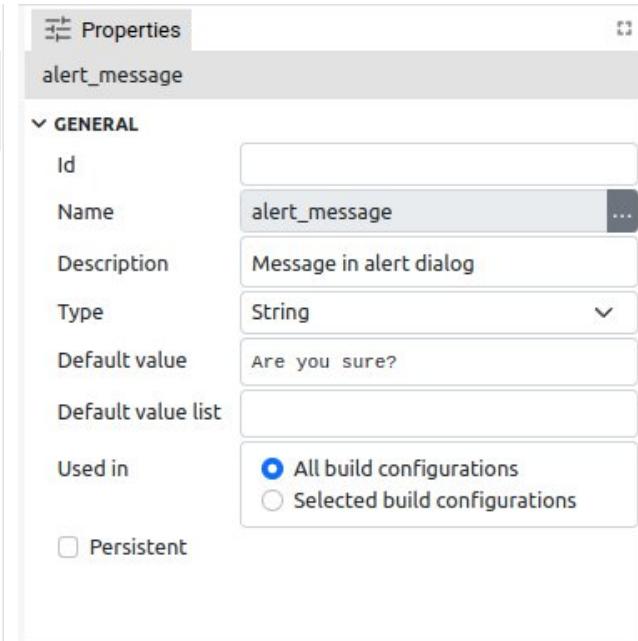


Fig. 78: Variable Properties panel (EEZ-GUI)

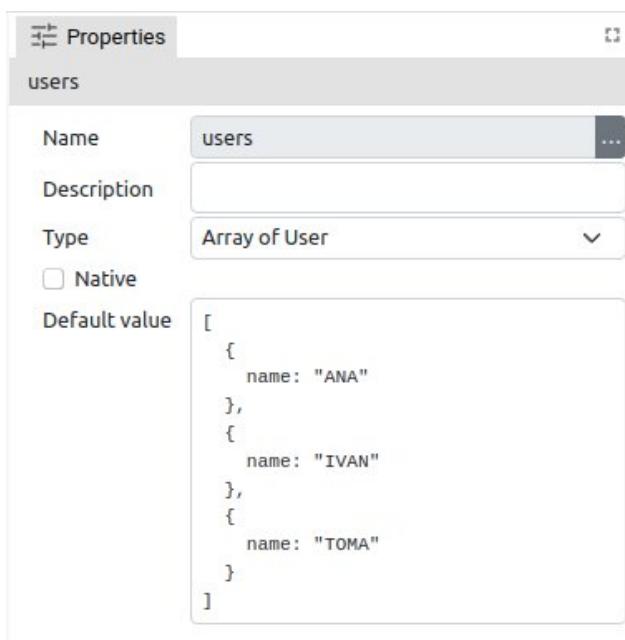


Fig. 79: Variable Properties panel (LVGL)

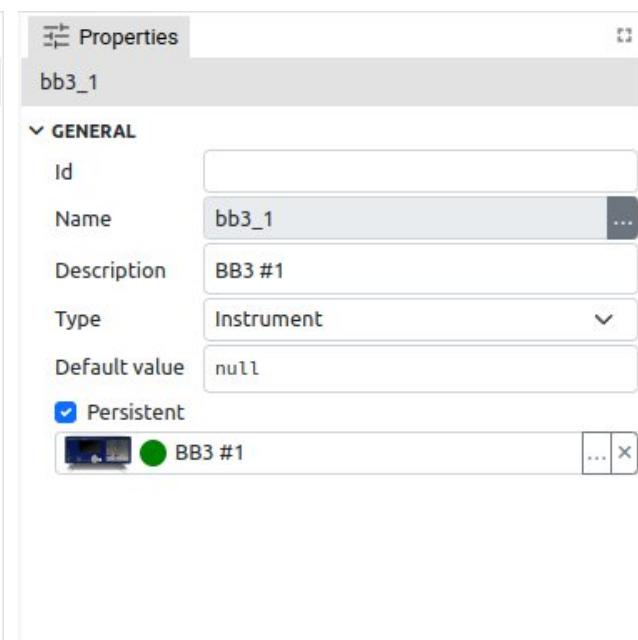


Fig. 80: Variable Properties panel (Dashboard)

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

Expression examples:

<code>`var[i].member1`</code>	<code>'var'</code> is array which contains structs, which has member <code>'member1'</code> <code>'i'</code> is integer variable evaluates to <code>'member1'</code> value in the i-th element
<code>`var == State.START var == State.EMPTY`</code>	<code>'var'</code> is of type enum:State, and State enum has two members: START and EMPTY evaluates to True if <code>var</code> contains data that is either <code>State.START</code> or <code>State.EMPTY</code>

P8.2.7. Literals

Type	Description / Example
<i>Integer</i>	<code>'42'</code>
<i>Float or double</i>	<code>'3.14'</code>
<i>String</i>	<code>'"Hello world!"'</code>
<i>Translated string</i>	<code>T"text_resource_id"</code> (prefix T is mandatory)
<i>Boolean</i>	<code>'true'</code> or <code>'false'</code>
<i>JSON</i>	See https://www.json.org/json-en.html

P8.2.8. Binary Operators

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

Addition +

Rules:

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

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

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

Subtraction -

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

Multiplication *

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

Division /

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

Remainder %

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

Left shift <<

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

Right shift >>

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

Binary AND &

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

Binary OR |

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

Binary XOR ^

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

P8.2.9. 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

P8.2.10. Unary operators**Type Description**

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

P8.2.11. Conditional (ternary) operator

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

P8.3. Functions

P8.3.1. System

System.getTick

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

Parameters

None

Return value

Value in milliseconds. Return type is `Integer`.

P8.3.2. Flow

Flow.index

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

Parameters

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

Return value

Created struct value. Return type is `Struct`.

Flow.makeArrayValue

Creates a new value of type array.

Parameters

Name	Type	Description
value	JSON	Array value.

Return value

Created array value. Return type is `Array`.

Flow.languages

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

Parameters

None

Return value

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

Flow.translate

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

Parameters

Name	Type	Description
textResourceID	String	Text resource ID.

Return value

Translated string. Return type is `String`.

Flow.parseInteger

Parse integer value given as string.

Parameters

Name	Type	Description
str	String	Input string.

Return value

Parsed integer value. Return type is `Integer`.

Flow.parseFloat

Parse float value given as string.

Parameters

Name	Type	Description
str	String	Input string.

Return value

Parsed float value. Return type is `Float`.

Flow.parseDouble

Parse double value give as string.

Parameters

Name	Type	Description
<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
date	Date	Input date.

Return value

Year. Return type is Integer.

Date.getMonth

Get month from date (1 to 12).

Parameters

Name	Type	Description
date	Date	Input date.

Return value

Month. Return type is Integer.

Date.getDay

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

Parameters

Name	Type	Description
date	Date	Input date.

Return value

Day. Return type is Integer.

Date.getHours

Get hours from date (0 to 23).

Parameters

Name	Type	Description
date	Date	Input date.

Return value

Hours. Return type is Integer.

Date.getMinutes

Get minutes from date (0 to 59).

Parameters

Name	Type	Description
date	Date	Input date.

Return value

Minutes. Return type is Integer.

Date.getSeconds

Get seconds from date (0 to 59).

Parameters

Name	Type	Description
date	Date	Input date.

Return value

Seconds. Return type is Integer.

Date.getMilliseconds

Get milliseconds from date (0 to 999).

Parameters

Name	Type	Description
date	Date	Input date.

Return value

Milliseconds. Return type is Integer.

Date.make

Make a date from arguments.

Parameters

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

Return value

Constructed date. Return type is Date.

P8.3.4. Math**Math.sin**

Returns the sine of a number in radians.

Parameters

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

Return value

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

Math.cos

Returns the cosine of a number in radians.

Parameters

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

Return value

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

Math.pow

Returns the value of a base raised to a power.

Parameters

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

Return value

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

Math.log

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

Parameters

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

Return value

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

Math.log10

Returns the base 10 logarithm of a number.

Parameters

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

Return value

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

Math.abs

Returns the absolute value of a number.

Parameters

Name	Type	Description
x	Integer Float Double	A number.

Return value

The absolute value of x. If x is negative (including -0), returns -x. Otherwise, returns x. The result is therefore always a positive number or 0. Return type is `Integer|Float|Double`.

Math.floor

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

Parameters

Name	Type	Description
x	Integer Float Double	A number.

Return value

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

Math.ceil

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

Parameters

Name	Type	Description
x	Integer Float Double	A number.

Return value

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

Math.round

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

Parameters

Name	Type	Description
x	Integer Float Double	A number.

Return value

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

Math.min

Returns the smallest of the numbers given as input parameters.

Parameters

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

Return value

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

Math.max

Returns the largest of the numbers given as input parameters.

Parameters

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

Return value

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

P8.3.5. String***String.length***

Returns the length of the string.

Parameters

Name	Type	Description
string	String	A string.

Return value

The length of a string. Return type is Integer.

String.substring

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

Parameters

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

Return value

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

String.find

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

Parameters

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

Return value

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

String.padStart

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

Parameters

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

Return value

A String of the specified targetLength with padString applied from the start. Return type is `String`.

String.split

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

Parameters

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

Return value

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

P8.3.6. Array

Array.length

The number of elements in given array.

Parameters

Name	Type	Description
array	Array	An array.

Return value

The length of an array. Return type is Integer.

Array.slice

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

Parameters

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

Return value

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

Array.allocate

Creates a new array of given size.

Parameters

Name	Type	Description
size	Array	A size number.

Return value

A new array. Return type is Array.

Array.append

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

Parameters

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

Return value

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

Array.insert

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

Parameters

Name	Type	Description

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

Return value

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

`Array.remove`

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

Parameters

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

Return value

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

`Array.clone`

Deep clone of the array.

Parameters

Name	Type	Description
array	Array	An array.

Return value

A new array. Return type is `Array`.

P8.3.7. LVGL**`LVGL.MeterTickIndex`**

See the LVGL Meter Widget description (Chapter 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. 81). Expressions can be entered manually or using the *Expression Builder*.

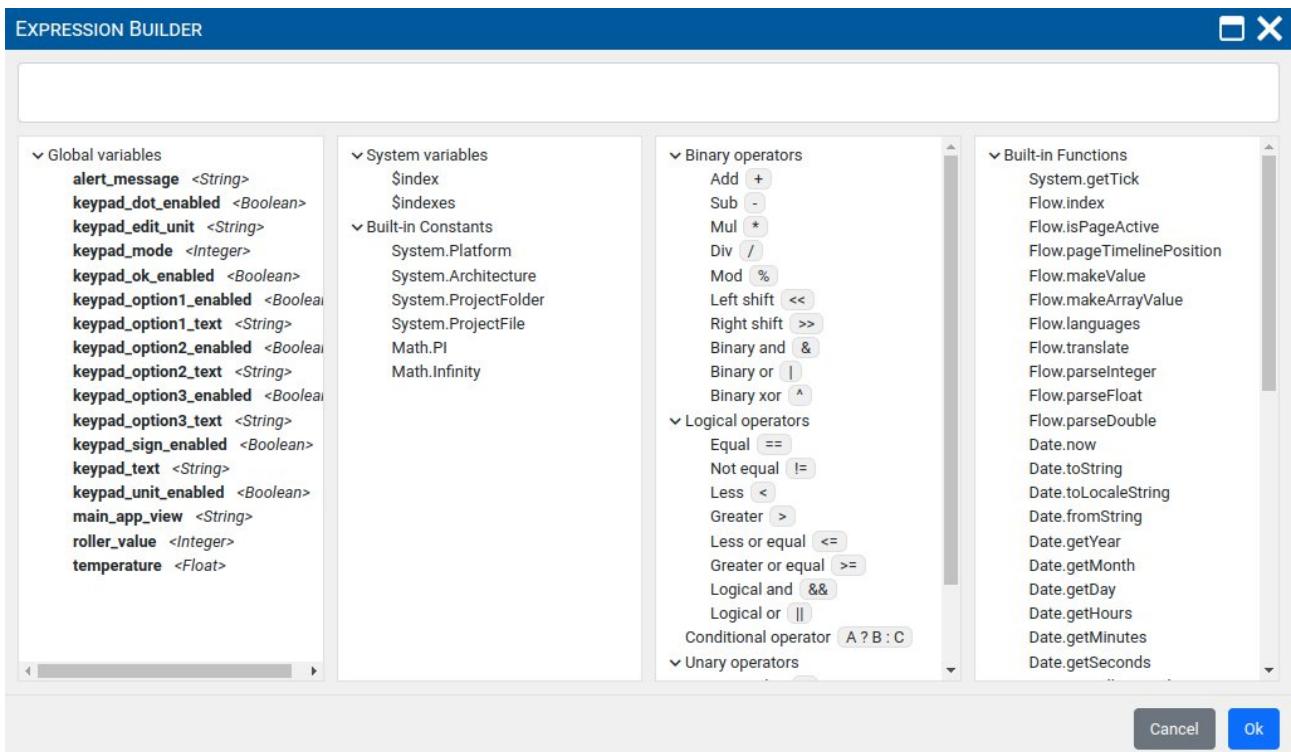


Fig. 81: Expression builder

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. 82 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.

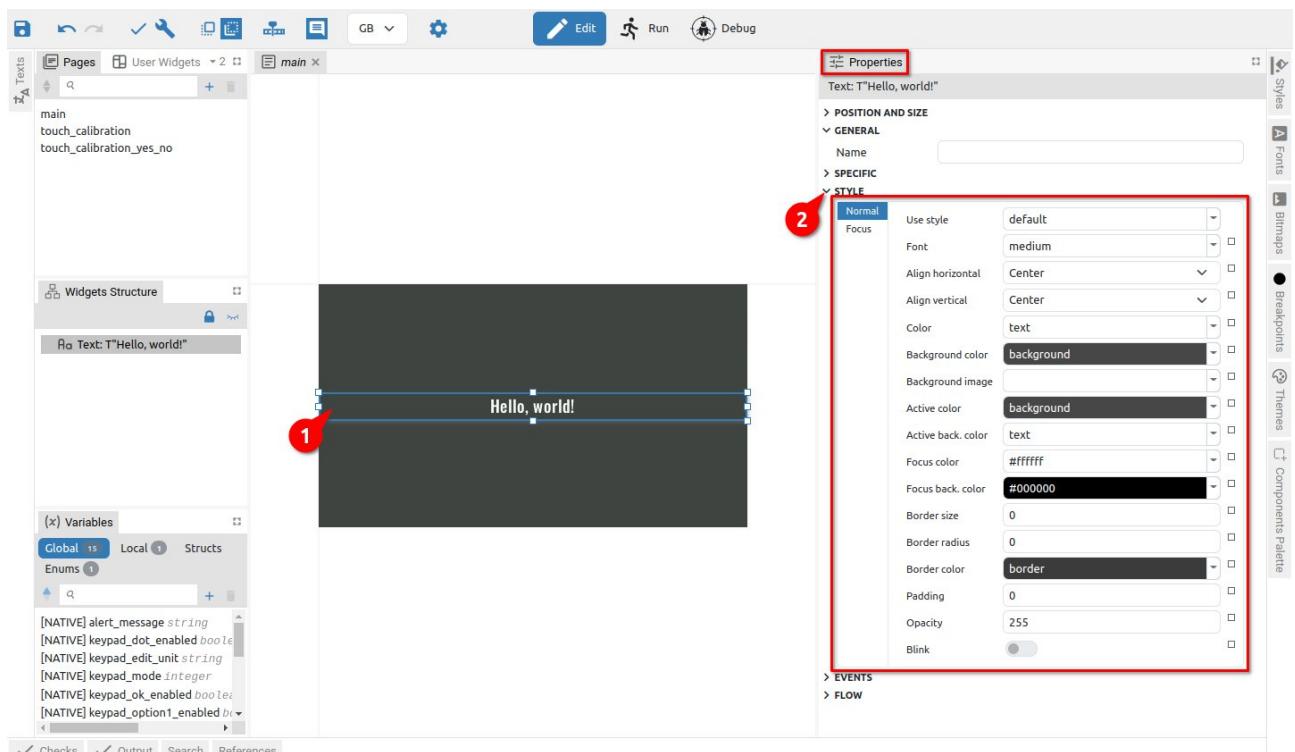


Fig. 82: Selected widget Style section for EEZ-GUI project type

The number of attributes and the appearance of the *Style* section depends on the project type in which the widget is used. In Fig. 82 the *Style* section for the *EEZ-GUI* project type is shown. Fig. 83 shows an example of the *Style* for the *Button* widget in the *Dashboard* project, and Fig. 84 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.

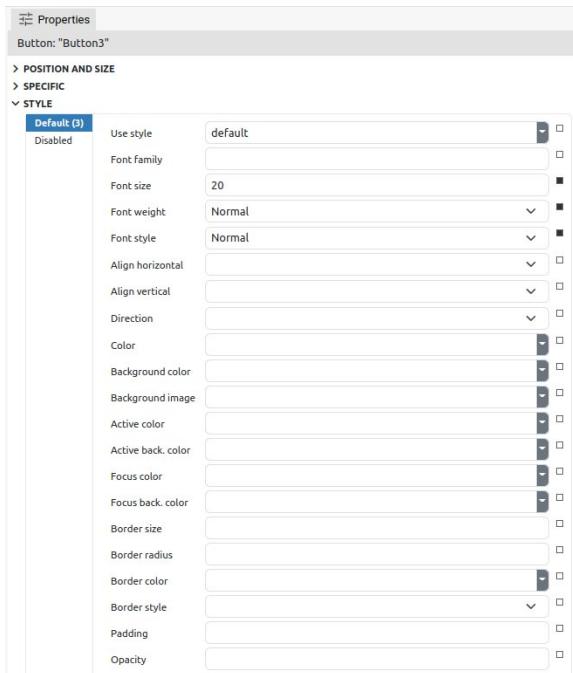


Fig. 83: Style properties (Dashboard project)

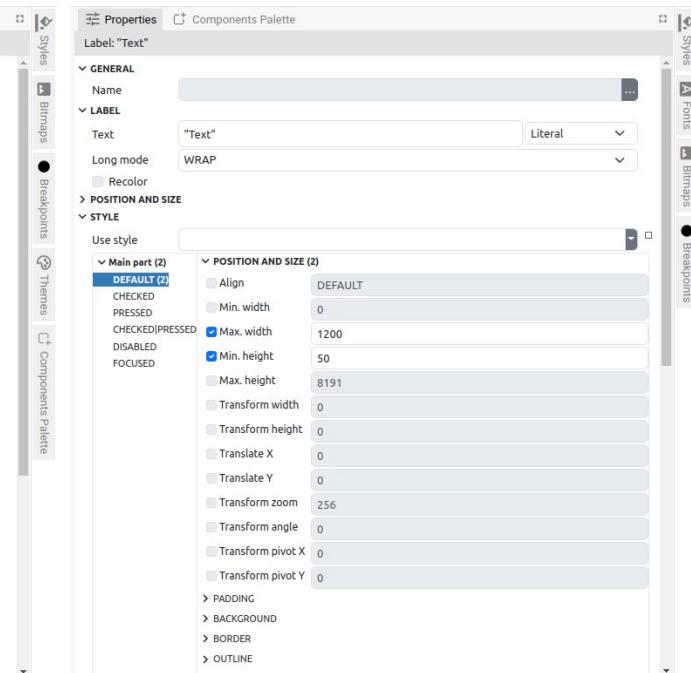


Fig. 84: Style properties (LVGL project)

P9.3. Project Styles

As shown in Fig. 82, 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. 86 the preview section for the *LVGL* project is shown (6).

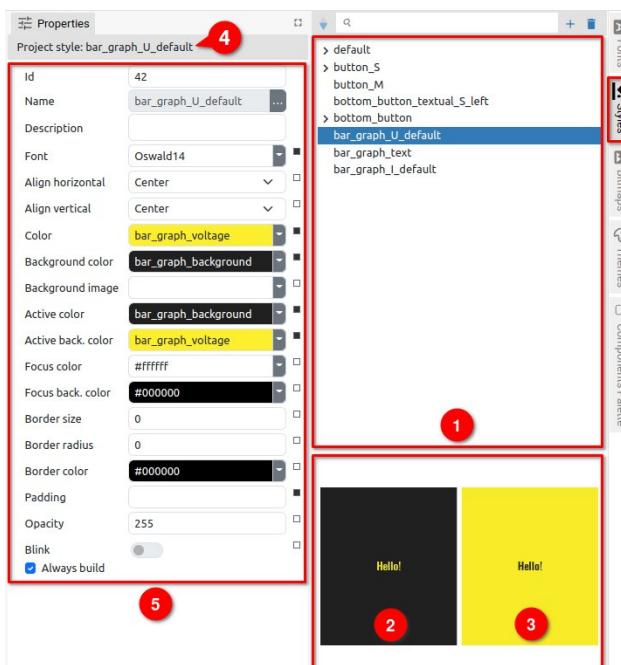


Fig. 85: Project Style Panel and Properties

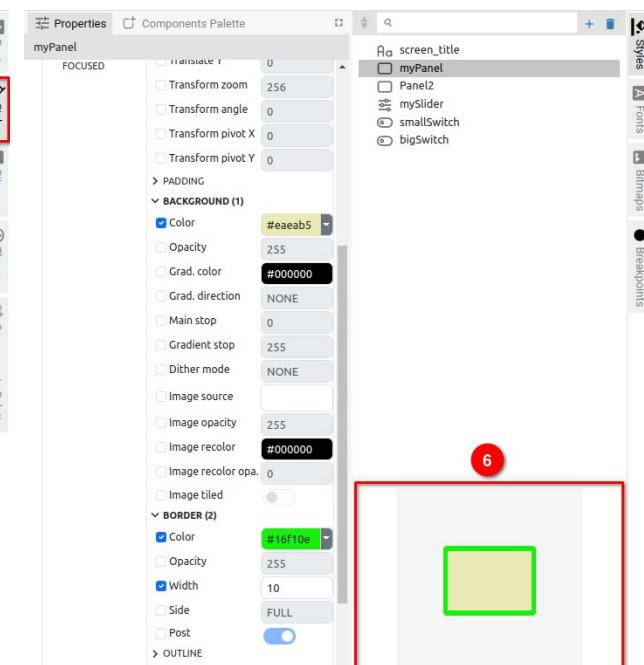


Fig. 86: LVGL Project style

P9.3.1. Creating a new Style

When creating a new Project style, it will be necessary to define the *Name* (Fig. 87) that must be unique. In the *LVGL* project, it will be necessary to choose for which Widget type it will be applied (Fig. 88).

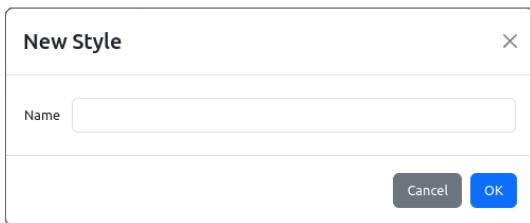


Fig. 87: Adding a new Style

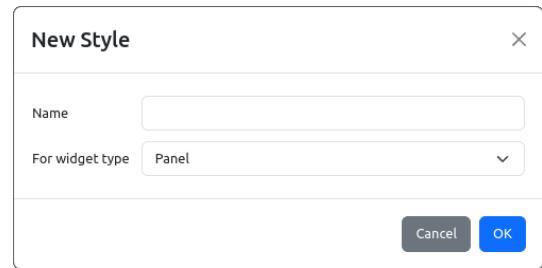


Fig. 88: Adding a new Style (LVGL)

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. 89).

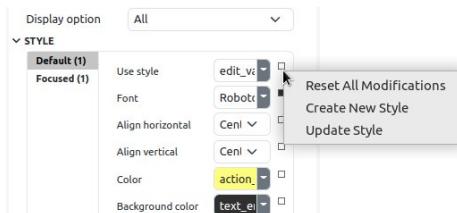


Fig. 89: Add new Style from currently selected Widget

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. 87 or Fig. 88). 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. 90 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*.

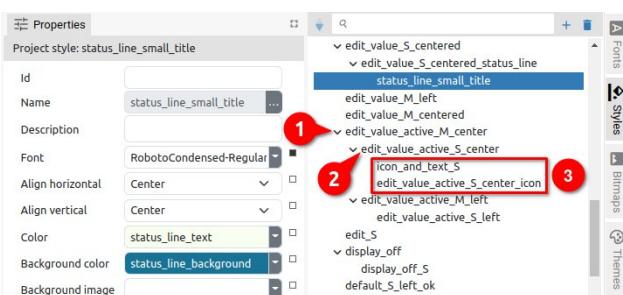


Fig. 90: Styles hierarchy

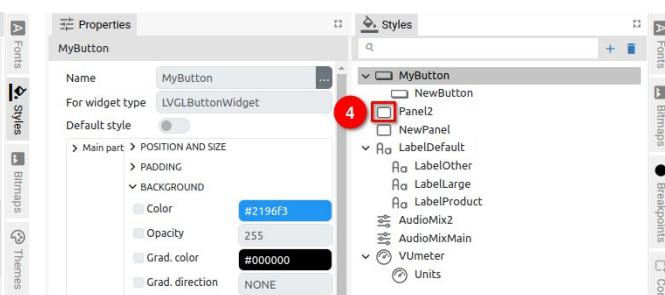


Fig. 91: Styles hierarchy in LVGL project

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. 91), 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. 92: 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).

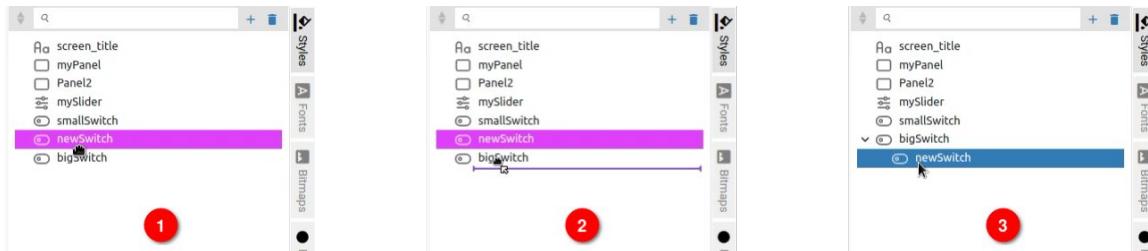


Fig. 92: Set child Style position

Resetting the child position is also carried out with drag & drop as shown in Fig. 110.

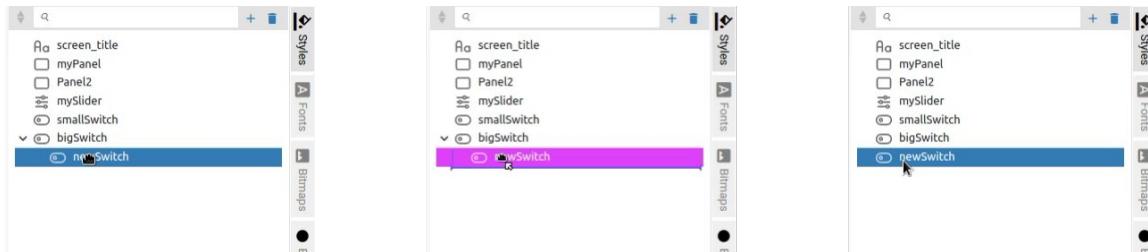


Fig. 93: Reset child Style position

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.

Fig. 94: Selecting a Style in the Style panel

Fig. 95: Setting the Style attribute color

In Fig. 94 and Fig. 95 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.

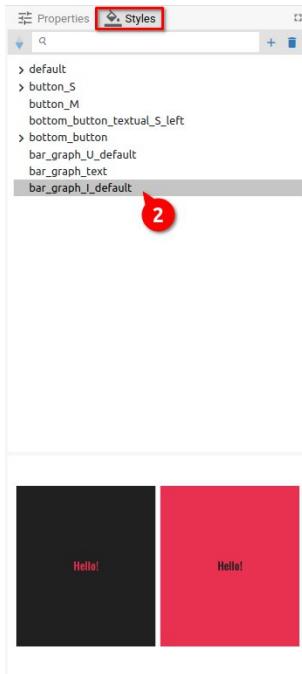


Fig. 96: Selecting a Style in the Style panel

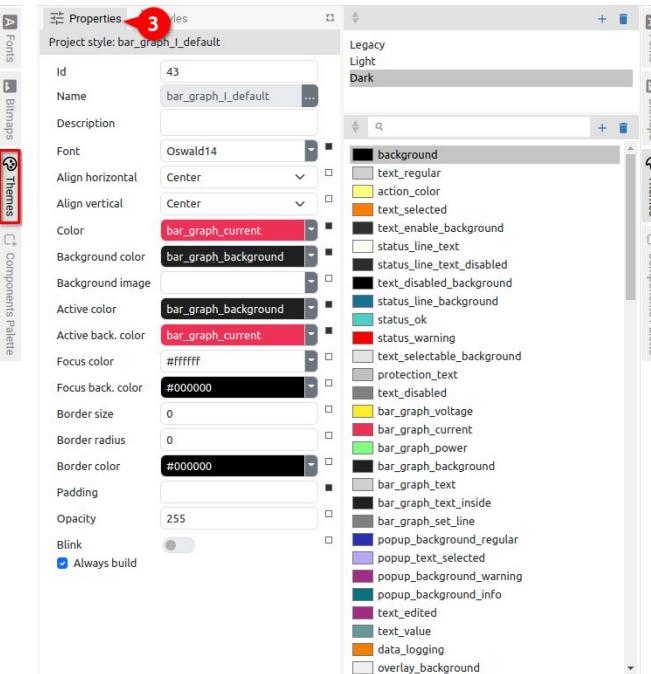


Fig. 97: Displaying the properties of the selected Style

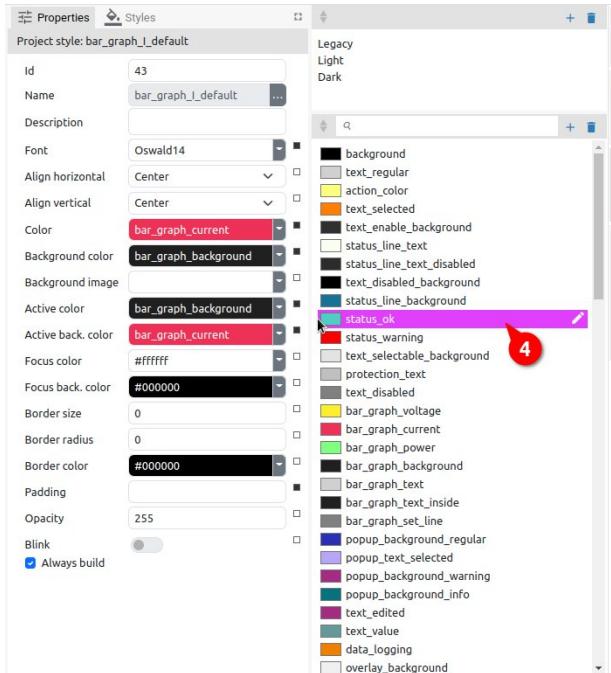


Fig. 98: Selecting a color from the Color theme

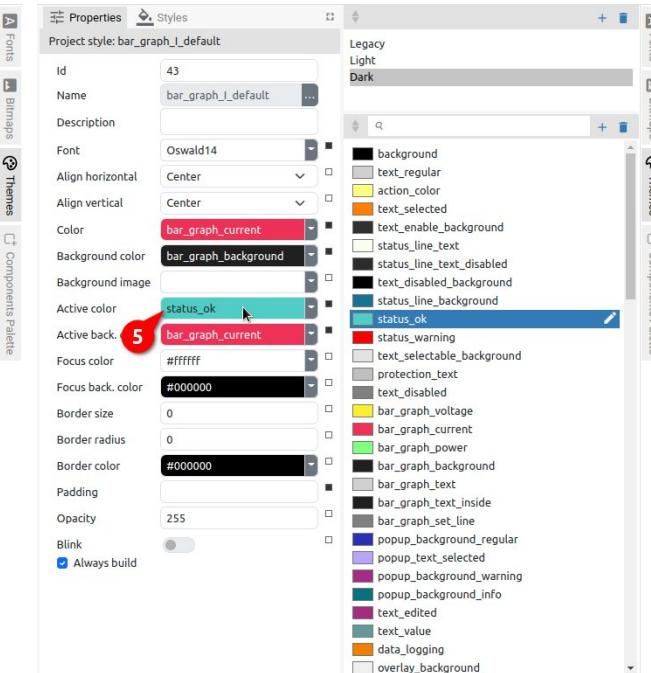


Fig. 99: Changing the color of the Style attribute

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. 100 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.

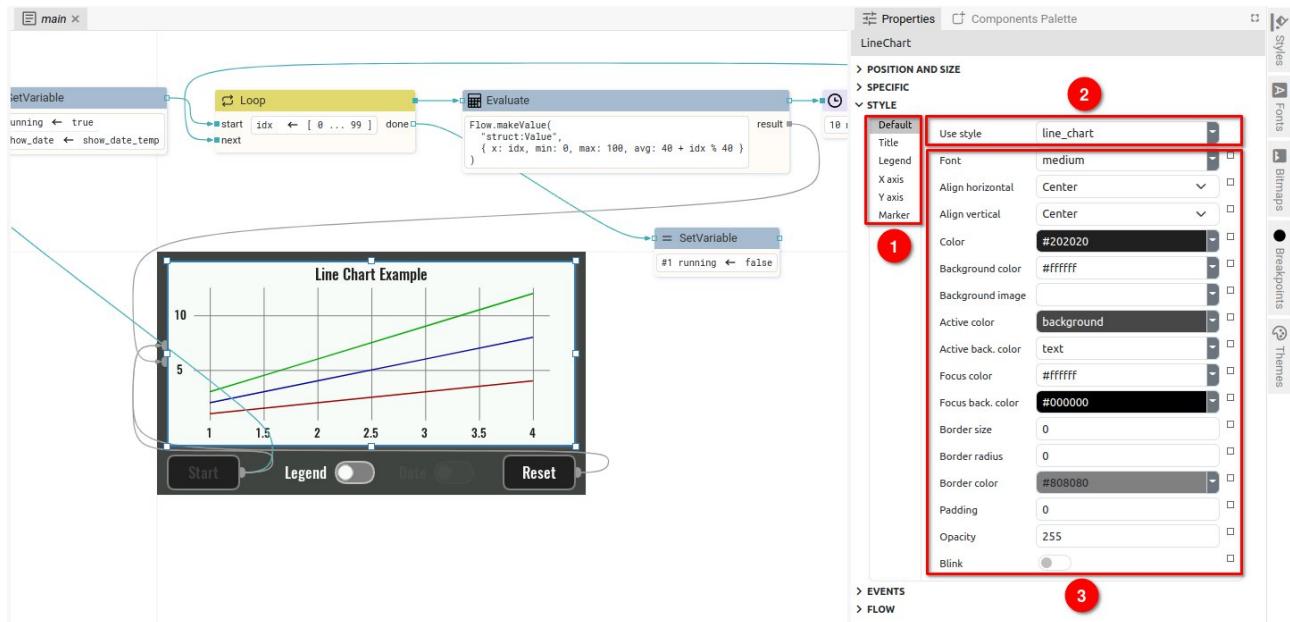


Fig. 100: EEZ-GUI project local Style attributes

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
Use style	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.
Font	The font for the texts displayed inside this Widget.
Align horizontal	Horizontal text alignment.
Align vertical	Vertical text alignment.
Color	The color of the text.
Background color	The background color of the Widget.
Background image	The background image of the Widget.
Active color	The color of the text when Widget is active. For example: <ul style="list-style-type: none"> • Button widget is active when it is clicked • when Blink property is enabled in Text widget it switches periodically between Normal and Active state.
Active back. color	The background color of the Widget when it is active.

Focus color**Focus back. color****Border size****Border radius**

The color of the text when Widget is in focus.

The background color of the Widget when it is in focus.

The line width used to draw the border.

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 *Normal* and *Active* 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.

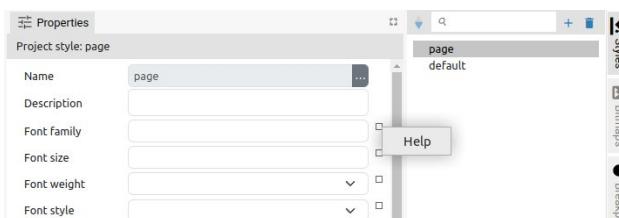


Fig. 101: Style attribute help

For each Style attribute, there is a *Help* link (Fig. 101) 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. 102). 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. 102: 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. 103 `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).

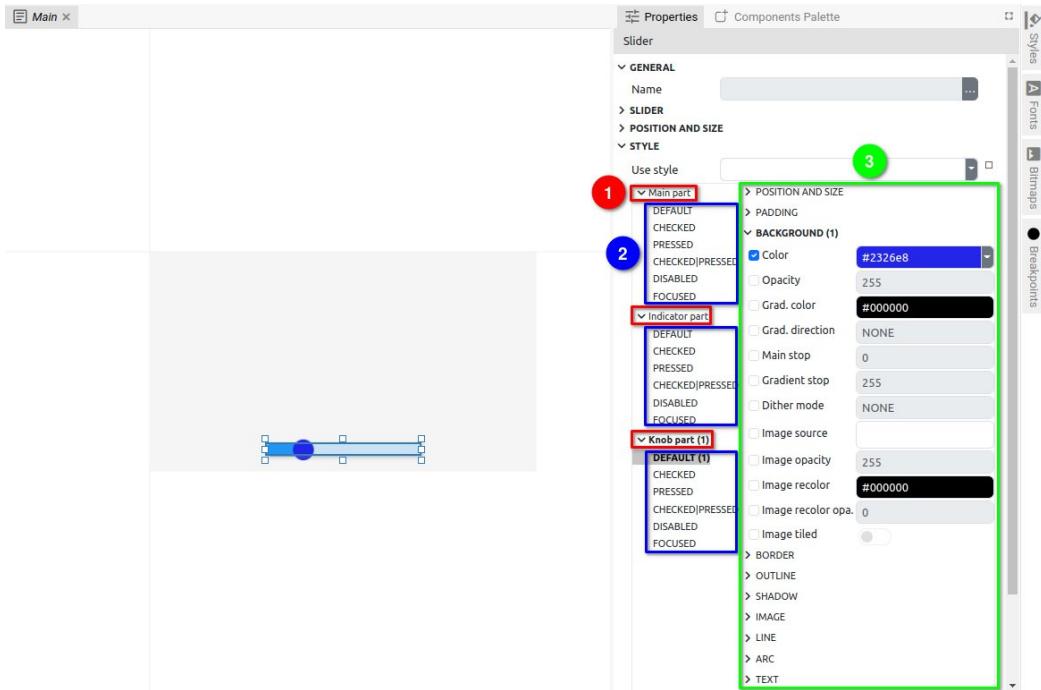


Fig. 103: LVGL project local Style attributes

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

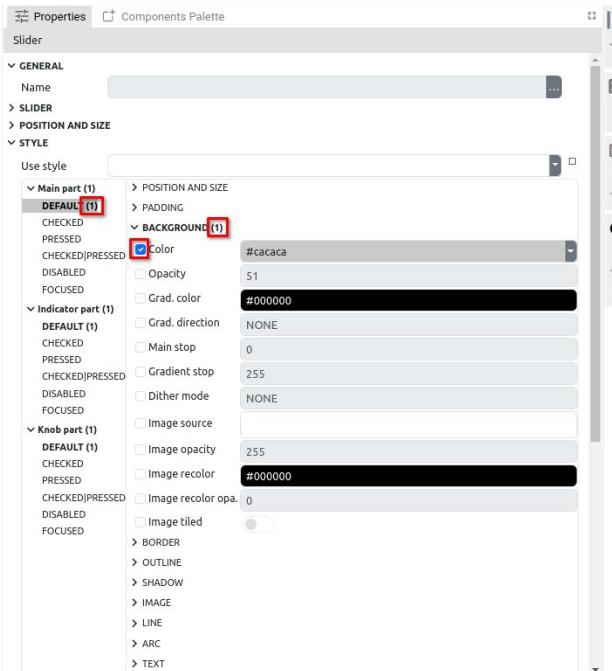


Fig. 104: Indication of modified attributes

In the category name, state name and part name there is an indication how many properties are changed (Fig. 104).

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. 109). 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. 85).

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

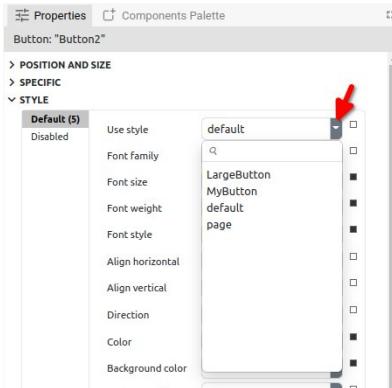


Fig. 105: Set a style from the Project style list

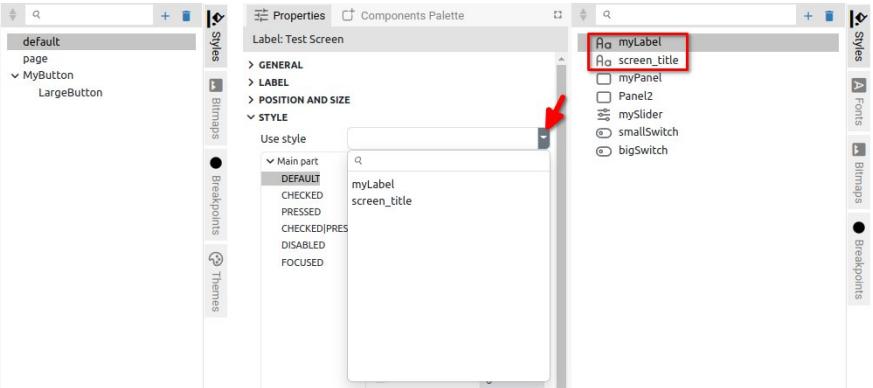


Fig. 106: Set a style from the Project style list (LVGL)

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. 99) 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. 98).

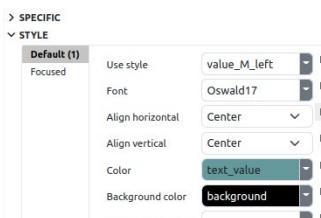


Fig. 107: Modified Style attribute indication

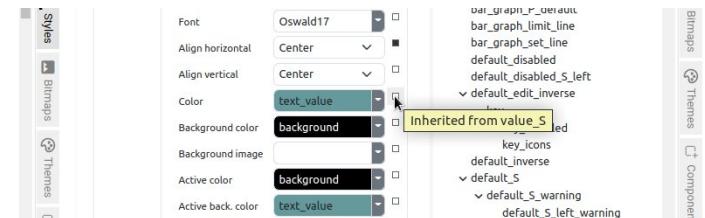


Fig. 108: Inherited Style attribute indication

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. 97). 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 96.

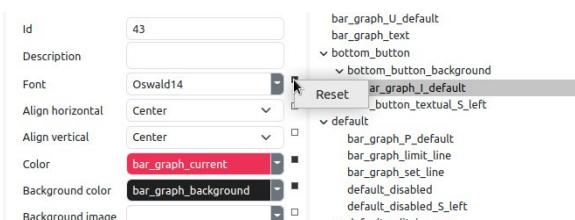


Fig. 109: Reset local style modification

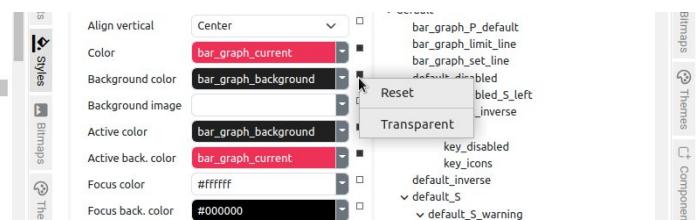


Fig. 110: Reset local style modification (Background)

P10. Bitmaps

In Fig. 111 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. 111: 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. 112 shows the dialog for *EEZ-GUI*, Fig. 113 shows the *Dashboard* and Fig. 114 shows the *LVGL* project.

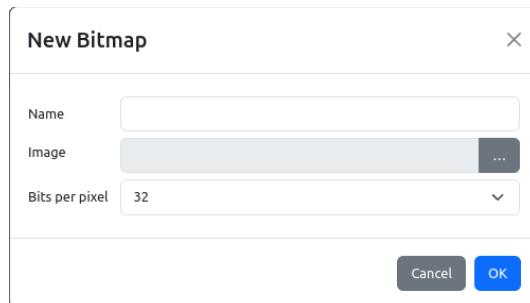


Fig. 112: Add new bitmap in EEZ-GUI project

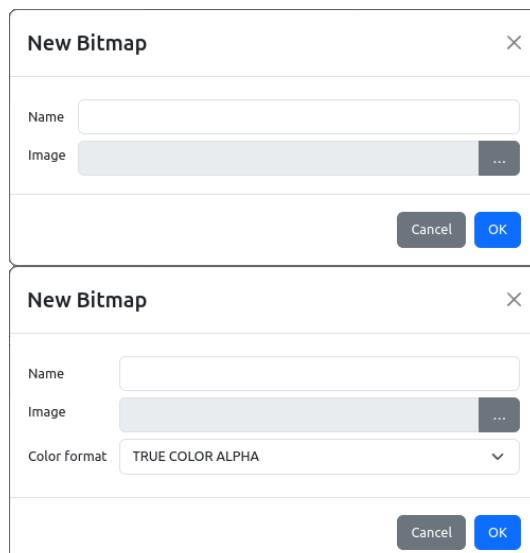


Fig. 114: Add new bitmap in LVGL project

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

P10.2. Bitmap properties

Fig. 115 shows the properties for the bitmap from the *Bitmap* Panel for the EEZ-GUI project. The parameters are described in the following table.

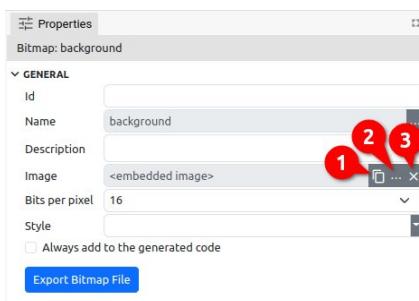


Fig. 115: Bitmap properties (EEZ-GUI)

Item	Description
<i>Id (EEZ-GUI only)</i>	<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 <input type="button" value="..."/> 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.

Color format (LVGL only)

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.

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

Export bitmap file 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. 116 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).

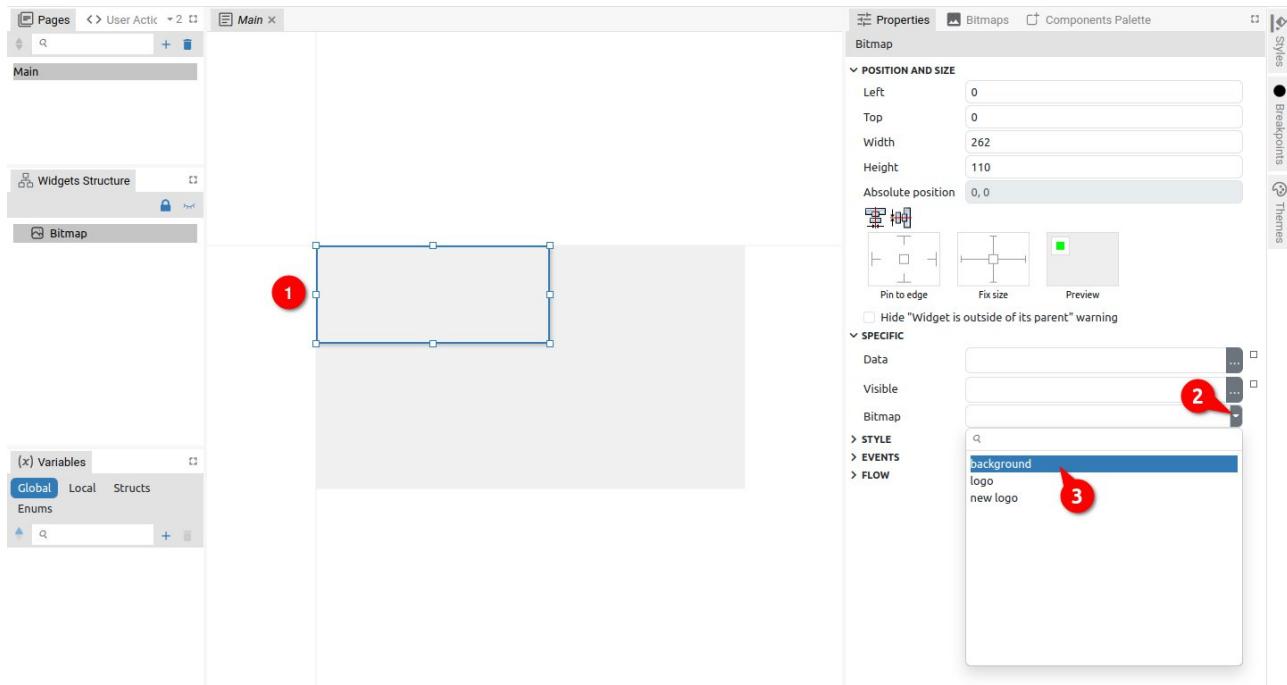


Fig. 116: Adding a bitmap to a Widget

Since the Widget's dimensions are smaller than the selected bitmap, the bitmap will exceed the Widget's borders as shown in Fig. 117. 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. 118).

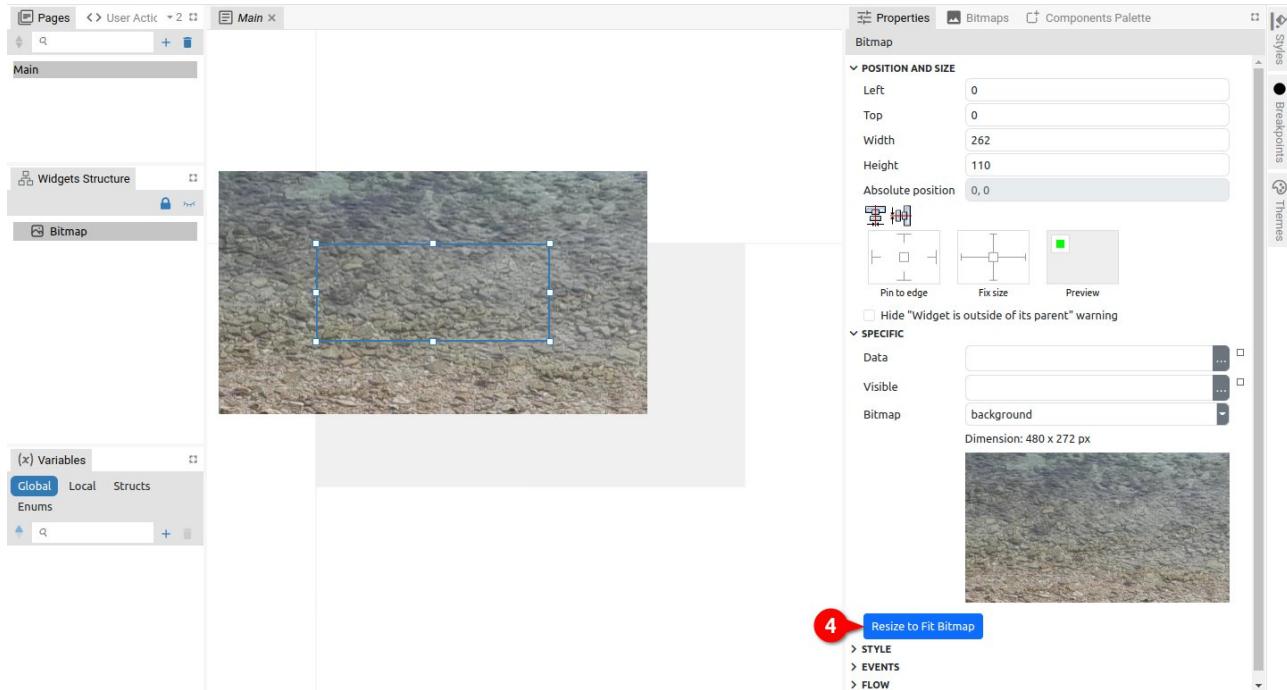


Fig. 117: Resizing the widget to fit the bitmap

Please note that this option is visible only if the current dimensions of the Widget do not match the dimensions of the bitmap (Fig. 118).

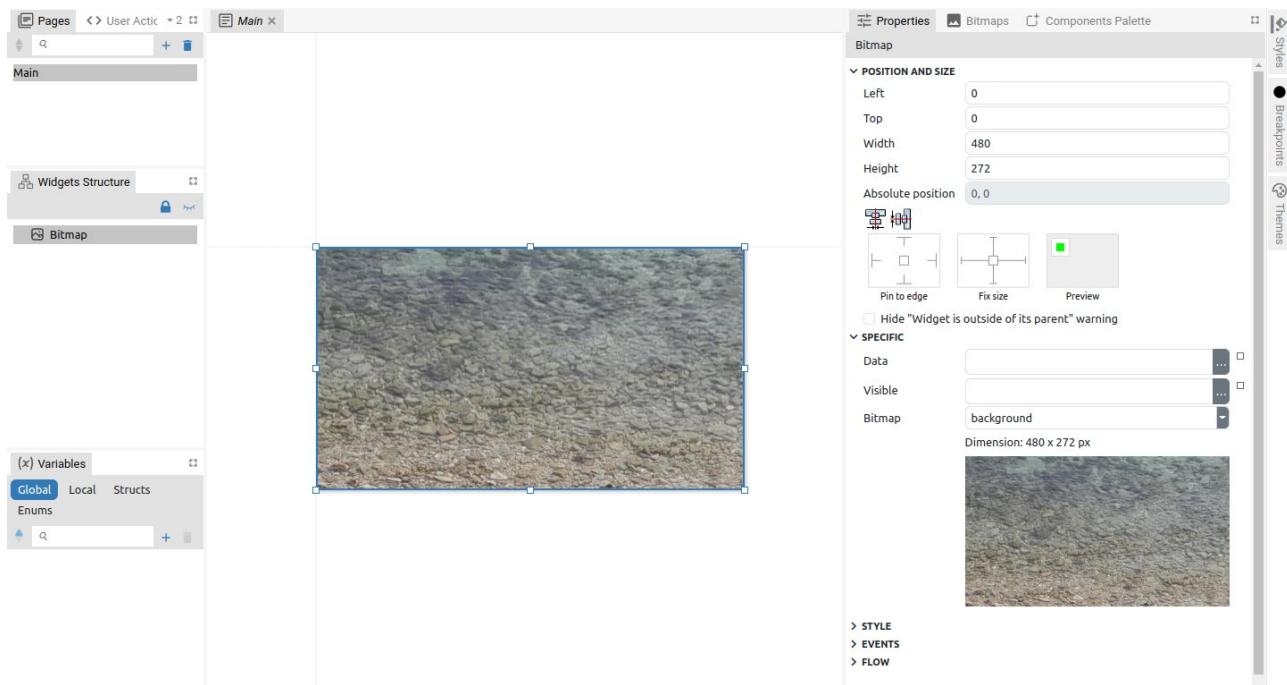


Fig. 118: Widget resized to bitmap size

P11. Fonts

The EEZ Studio project supports working with fonts. To work with fonts, it will be necessary in the project Settings (Fig. 119) 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.

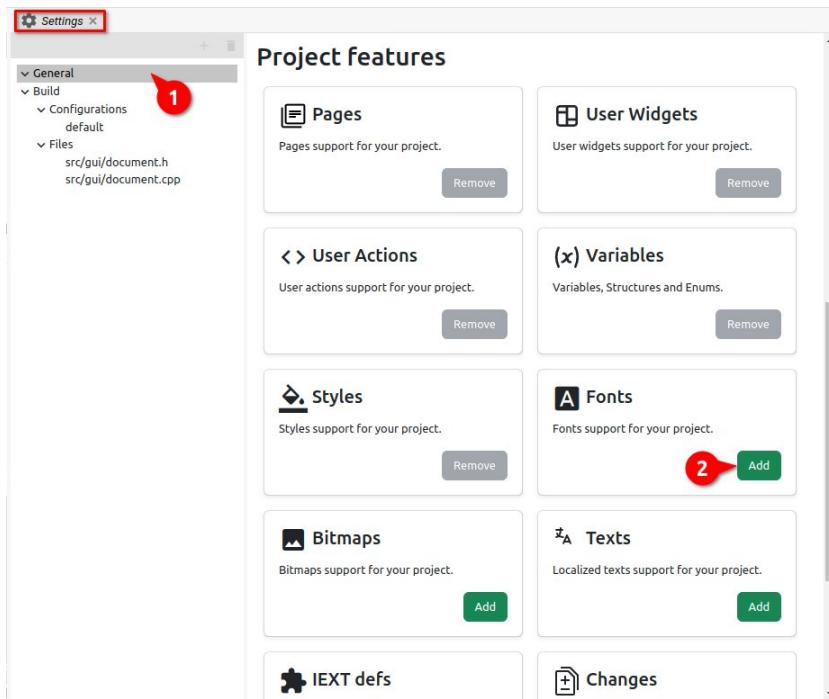


Fig. 119: Enable Fonts in project Settings

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. 120.

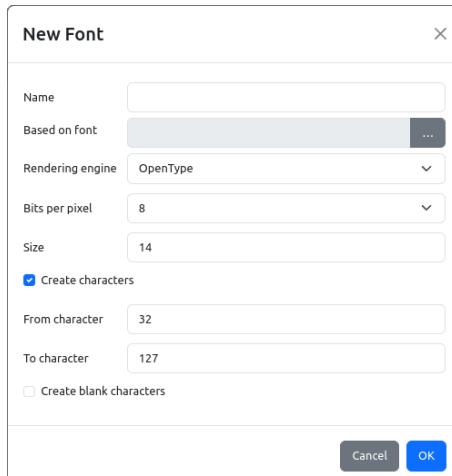


Fig. 120: Adding a new font to the EEZ-GUI project

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. 121 For the selected character, its enlarged preview will be displayed on the right.

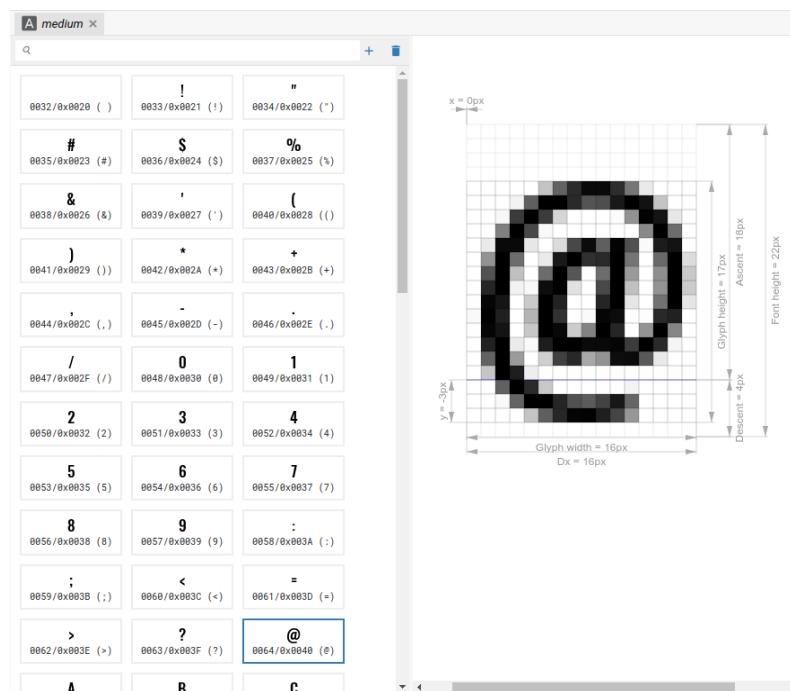


Fig. 121: Font character table (EEZ-GUI project)

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. 122.



Fig. 122: Add/delete font character

When adding a new character, a dialog opens as shown in Fig. 123.

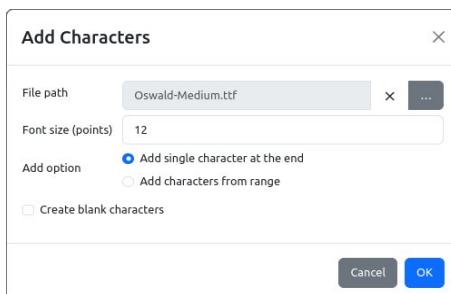


Fig. 123: Add new font character

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. 124.

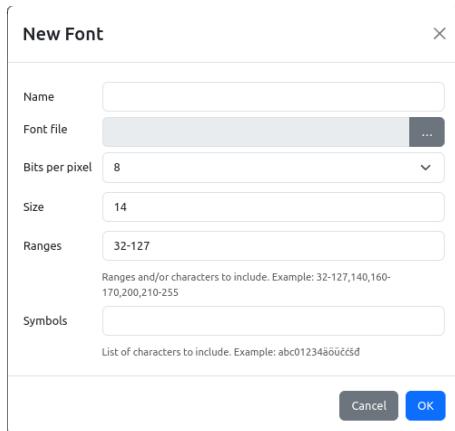


Fig. 124: Adding a new font to the LVGL project

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. 125 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.

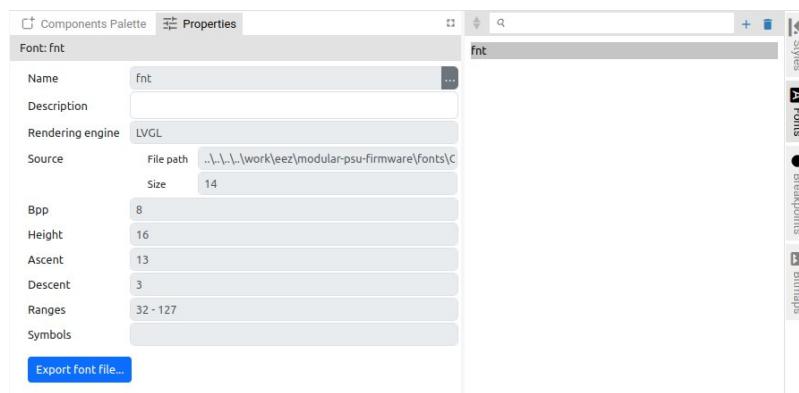


Fig. 125: Font properties

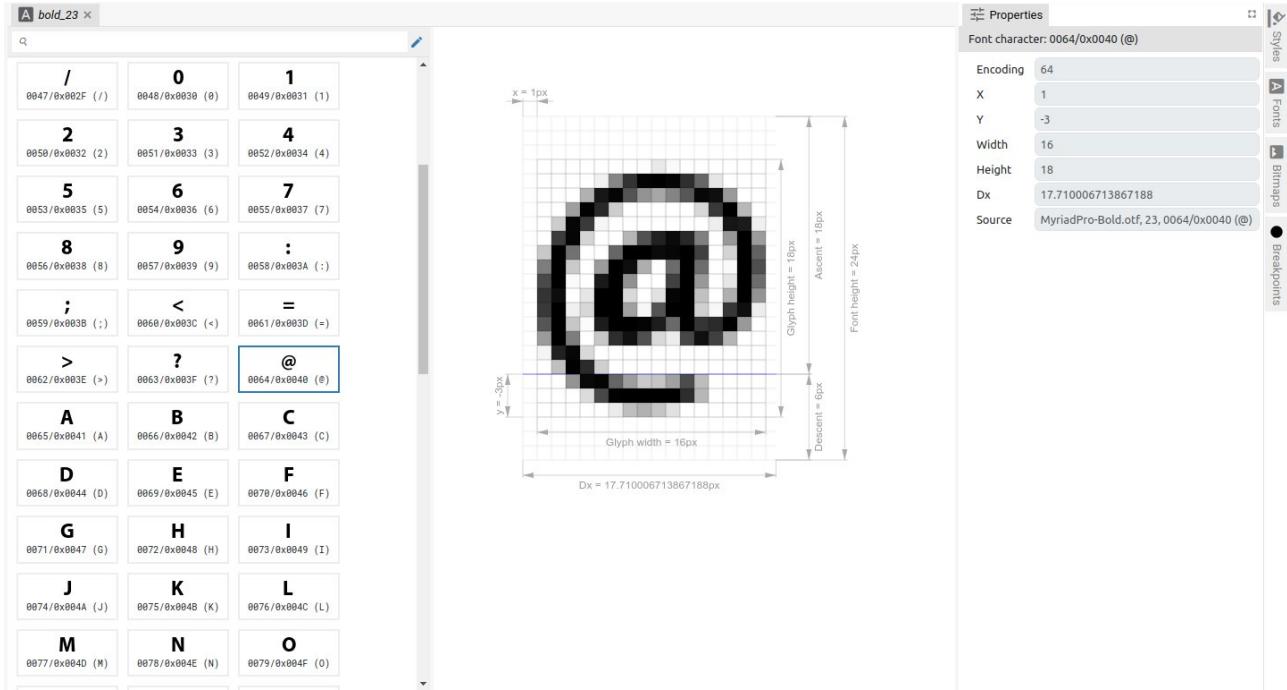


Fig. 126: Font character table (LVGL project)

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. 127.

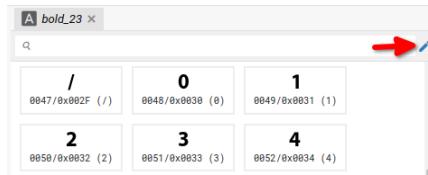


Fig. 127: Add or Remove characters option (LVGL project)

The dialog shown in Fig. 126 through which it is possible to define the Ranges and/or Symbols of the character we want to have in the font table.

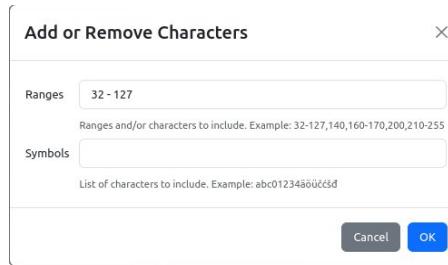


Fig. 128: Font characters editing (LVGL project)

P12. Texts

Projects that support multi-language texts can be found in *New Project Examples* (Fig. 129) 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.

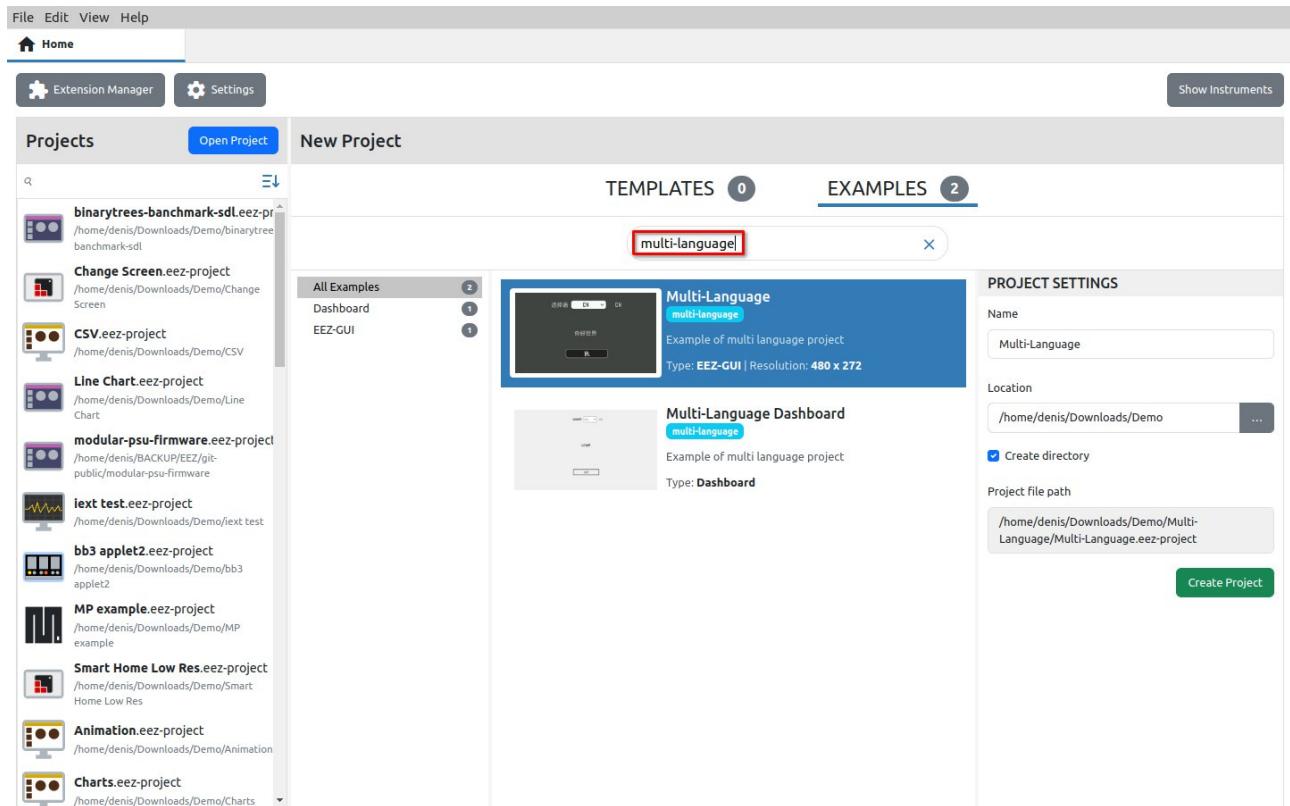


Fig. 129: Multi-language project examples

P12.1. Texts panel

Fig. 130 shows the *Texts* panel for multilingual text editing, which consists of the following three tabs: *Text resources* (1), *Languages* (2) and *Statistics* (3).

The screenshot shows the 'Texts' panel with four tabs: 'Properties' (4), 'Text resources' (1), 'Languages' (2), and 'Statistics' (3). The 'Text resources' tab is active, showing a progress bar at 50% and a 'Select language:' dropdown with options GB, FR, HR, and CN. The 'Languages' tab shows a list of languages (GB, FR, HR, CN) with progress bars at 67% for each. The 'Components Palette' on the left lists various UI components like DisplayData, MultilineText, Bitmap, etc.

Fig. 130: Editing Texts resources

The screenshot shows the 'Texts' panel with the same four tabs. The 'Text resources' tab now shows a progress bar at 83% and a completed list of translations for GB, FR, HR, and CN. The 'Languages' tab shows a list of languages with green checkmarks indicating completion. The 'Statistics' tab shows a summary of the translation progress: Progress 83%, Available languages 4, Available text resources 3, Total strings 12, and No. translations 10. The 'Components Palette' on the left is identical to Fig. 130.

Fig. 131: 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. 132, and for adding a new multilingual *Text resource*, a dialog opens as shown in Fig. 133.



Fig. 132: Add new Language



Fig. 133: 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. 131 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.

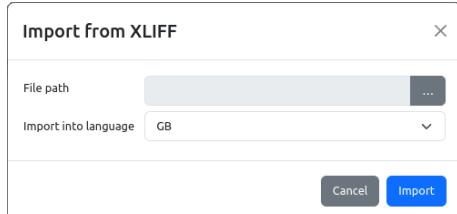


Fig. 134: Text import from XLIFF file

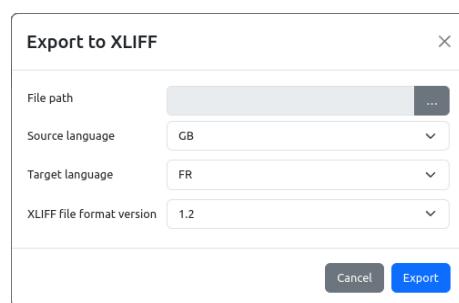


Fig. 135: Text export to XLIFF file

During import, a dialog opens as shown in Fig. 134 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. 135), 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. 136 shows the exported XLIFF file in the *Poedit* application (*Source language* is GB, *Target language* is FR).

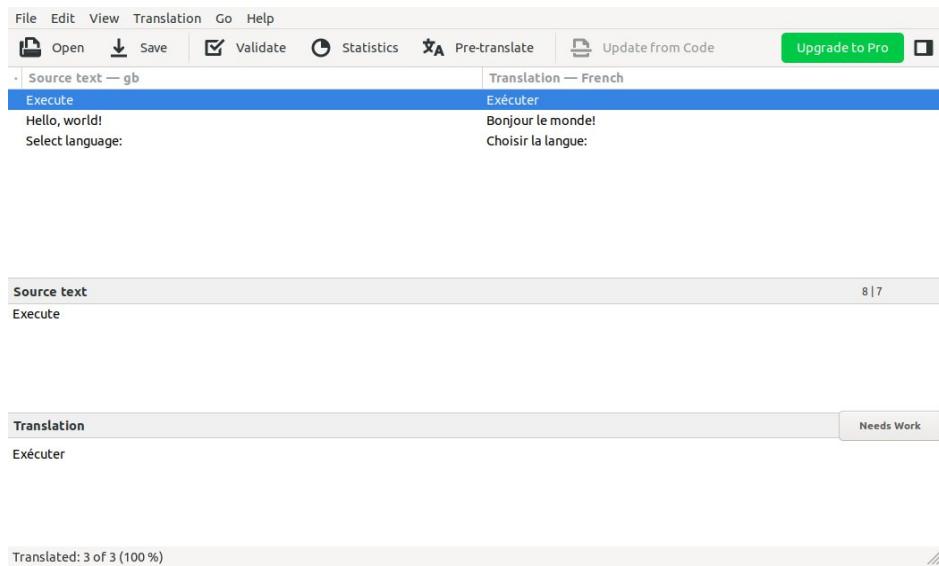


Fig. 136: Exported XLIFF file opened in Poedit

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. 137.

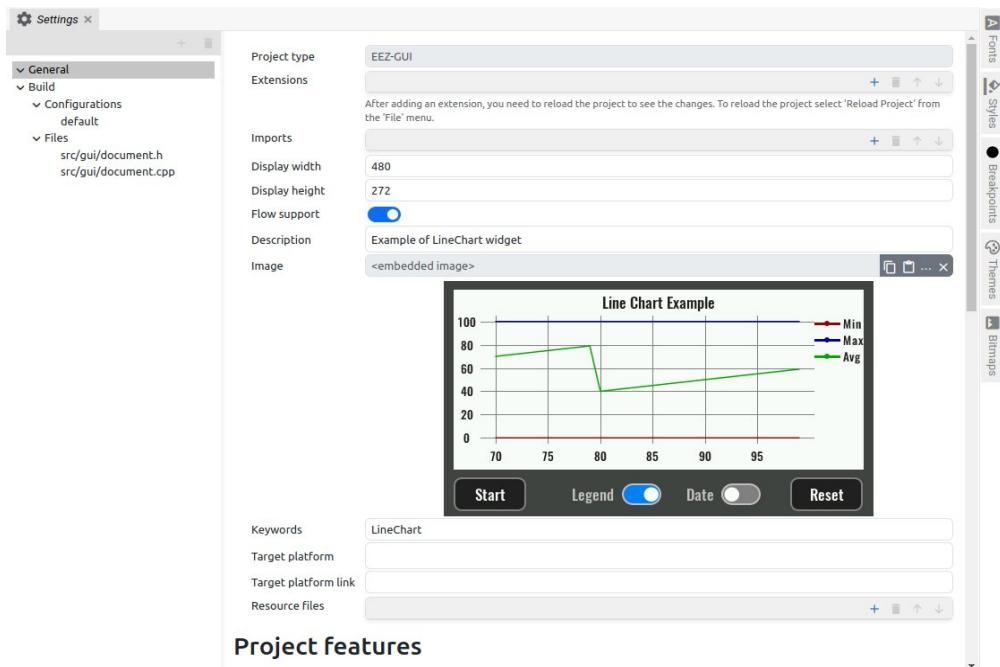


Fig. 137: General Project settings (*EEZ-GUI*)

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>New Project Examples</i> section.
<i>Image</i>	Project screenshot shown in the <i>New Project Examples</i> section.
<i>Keywords</i>	Project keywords shown in the <i>New Project Examples</i> section.

<i>Target platform</i>	Project target platform shown in the <i>New Project Examples</i> section.
<i>Target platform link</i>	Link to the website of the target platform shown in the New Project Examples section.
<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. 138 are shown for the EEZ-GUI project.

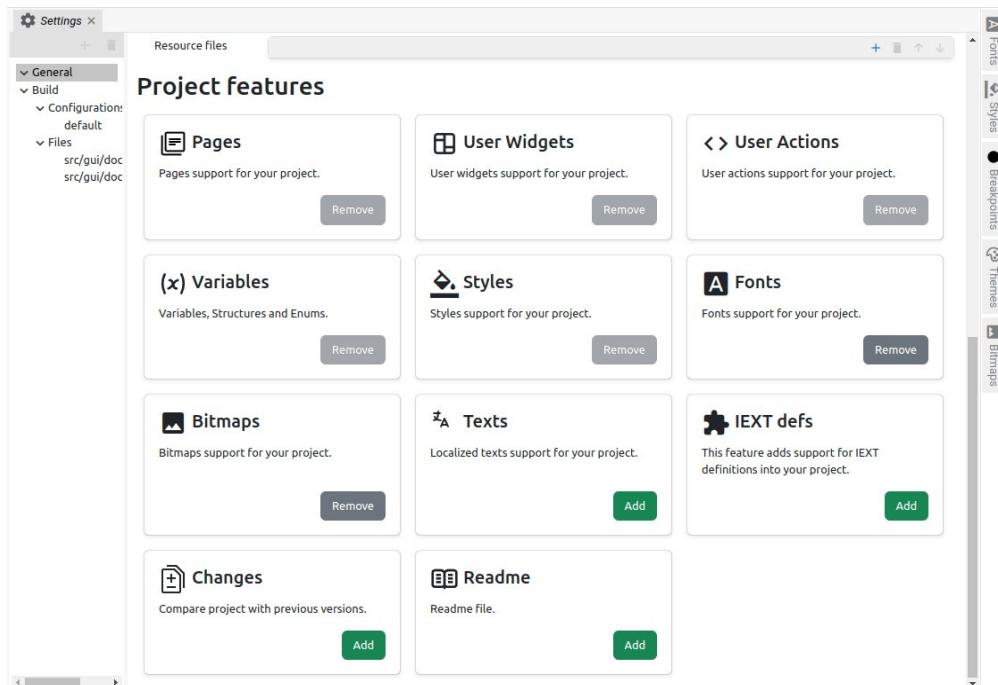


Fig. 138: Project features (EEZ-GUI)

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.

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. 139) is used to define in which configuration the item will be used.

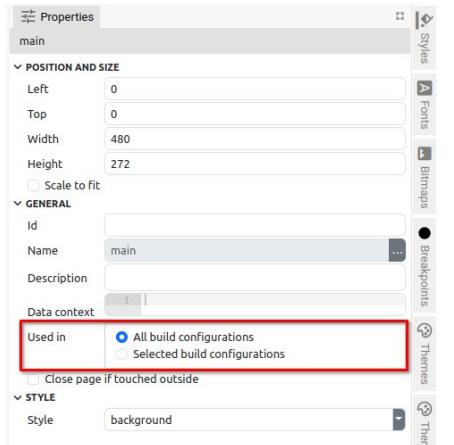


Fig. 139: EEZ-GUI project Used in parameter

Build Configuration parameters are shown in Fig. 140.



Fig. 140: EEZ-GUI build configuration settings

Item	Description
Name	The name of the build configuration.
Description	Description of the build configuration.
Properties	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

A1. AddToInstrumentHistory.....	A.5
A2. Animate.....	A.11
A3. CatchError.....	A.13
A4. CollectStream.....	A.15
A5. Comment.....	A.17
A6. Compare.....	A.19
A7. ConnectInstrument.....	A.21
A8. Constant.....	A.23
A9. Counter.....	A.25
A10. CSVParse.....	A.27
A11. CSVStringify.....	A.31
A12. DateNow.....	A.33
A13. Delay.....	A.35
A14. DisconnectInstrument.....	A.37
A15. DynamicCallAction.....	A.39
A16. End.....	A.41
A17. Error.....	A.43
A18. Eval JS.....	A.45
A19. Evaluate.....	A.47
A20. ExecuteCommand.....	A.49
A21. FileAppend.....	A.51
A22. FileOpenDialog.....	A.53
A23. FileRead.....	A.55
A24. FileSaveDialog.....	A.57
A25. FileWrite.....	A.59
A26. GetInstrument.....	A.61
A27. GetInstrumentProperties.....	A.65
A28. HTTP.....	A.69
A29. Input.....	A.71
A30. IsTrue.....	A.73
A31. JSONParse.....	A.75
A32. JSONStringify.....	A.79
A33. Log.....	A.81
A34. Loop.....	A.83
A35. LVGL.....	A.87
A36. MQTTConnect.....	A.91
A37. MQTTDisconnect.....	A.93
A38. MQTTEvent.....	A.95
A39. MQTTInit.....	A.97
A40. MQTTPublish.....	A.99
A41. MQTTSubscribe.....	A.101
A42. MQTTUnsubscribe.....	A.103
A43. NoOp.....	A.105

A44. OnEvent.....	A.107
A45. Output.....	A.109
A46. OverrideStyle.....	A.111
A47. PythonEnd.....	A.113
A48. PythonRun.....	A.115
A49. PythonSendMessage.....	A.119
A50. ReadSetting.....	A.121
A51. Regexp.....	A.123
A52. SCPI.....	A.127
A53. SelectInstrument.....	A.131
A54. SelectLanguage.....	A.133
A55. SerialConnect.....	A.135
A56. SerialDisconnect.....	A.137
A57. SerialInit.....	A.139
A58. SerialListPorts.....	A.141
A59. SerialRead.....	A.143
A60. SerialWrite.....	A.145
A61. SetPageDirection.....	A.147
A62. SetVariable.....	A.149
A63. ShowFileInFolder.....	A.151
A64. ShowKeyboard.....	A.153
A65. ShowKeypad.....	A.157
A66. ShowMessageBox.....	A.161
A67. ShowPage.....	A.165
A68. SortArray.....	A.167
A69. Start.....	A.169
A70. SwitchCase.....	A.171
A71. TestAndSet.....	A.173
A72. Watch.....	A.175
A73. WriteSetting.....	A.177

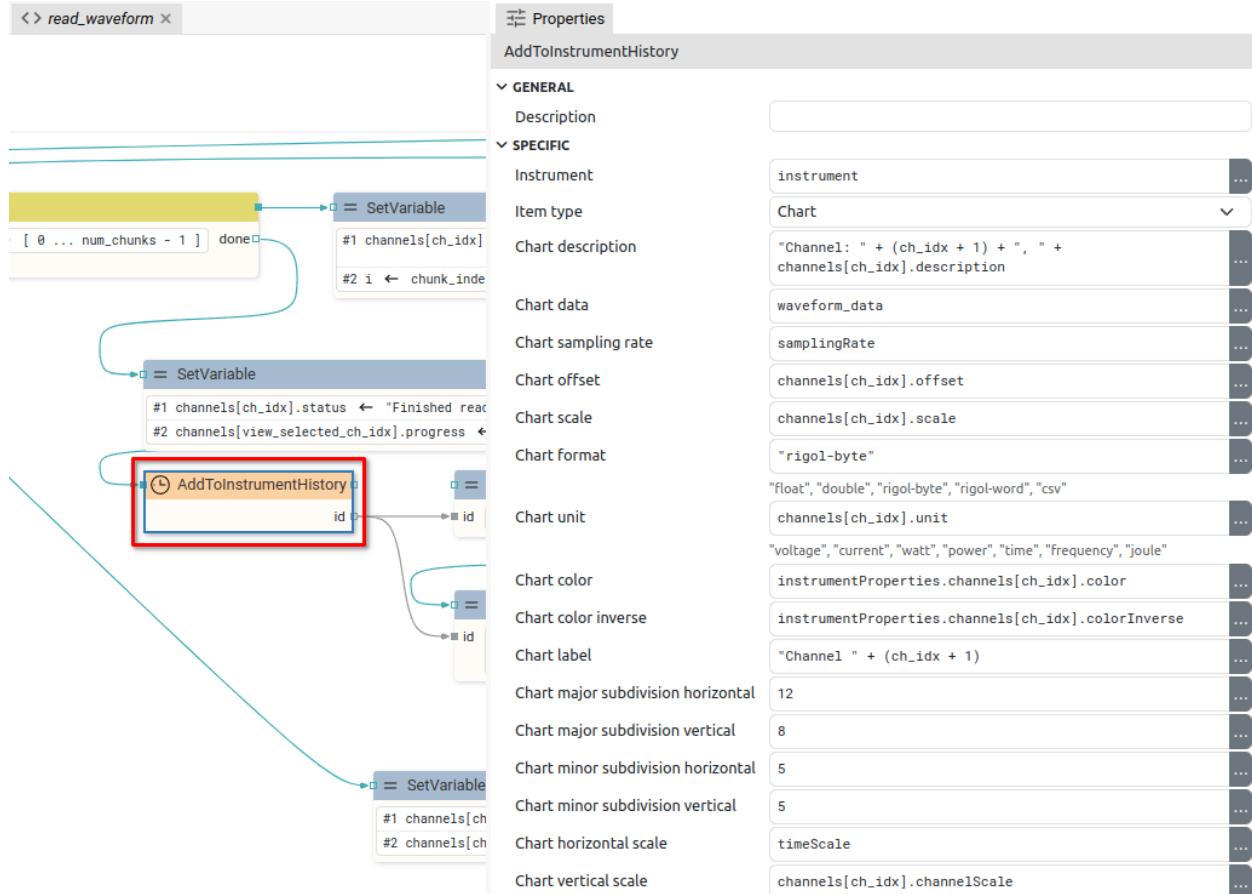
A1. AddToInstrumentHistory



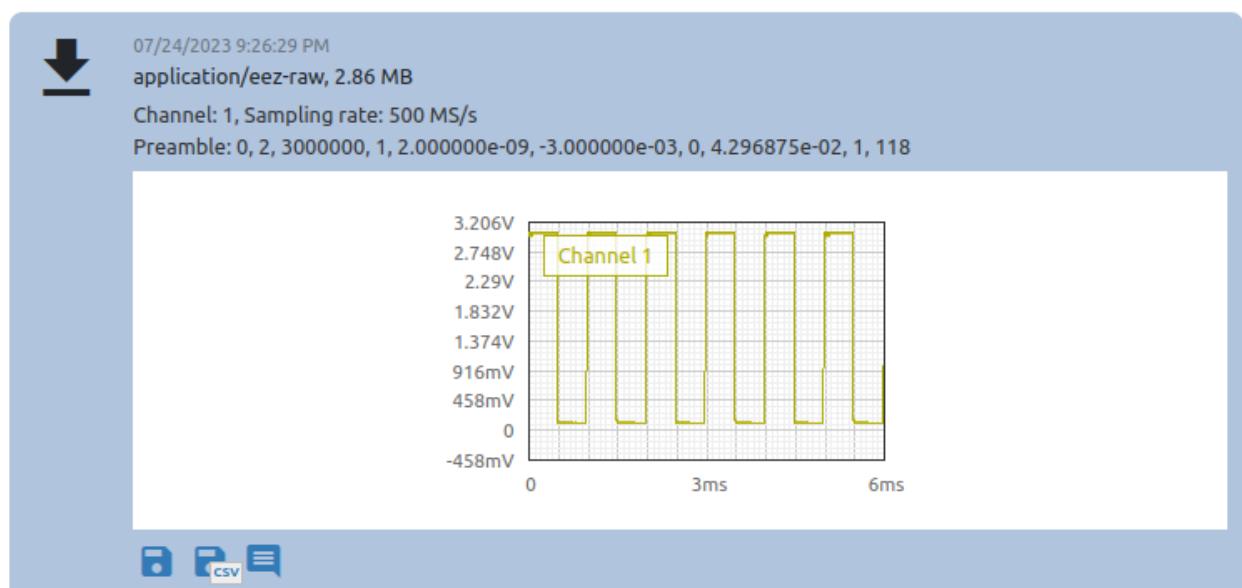
A1.1. Description

It is used to add a new item to the *History* view of the instrument. Currently, only adding chart items is 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)*

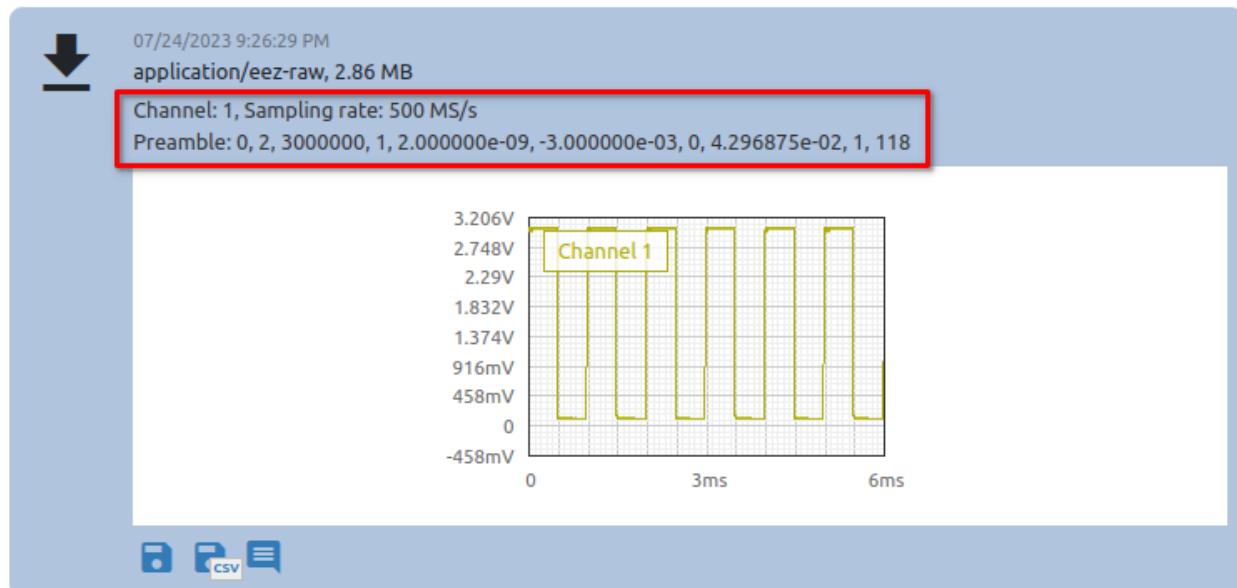
An instrument in whose *History* an item will be added.

A1.2.2. Item type *Enum*

Item type to be added, currently it can only be "Chart".

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

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



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

A string or blob containing the samples that will be displayed in the chart.

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

Sampling rate or number of samples per second (SPS).

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.

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

When displaying samples, the formula `offset + sample_value * scale` is used.

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.

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

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

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

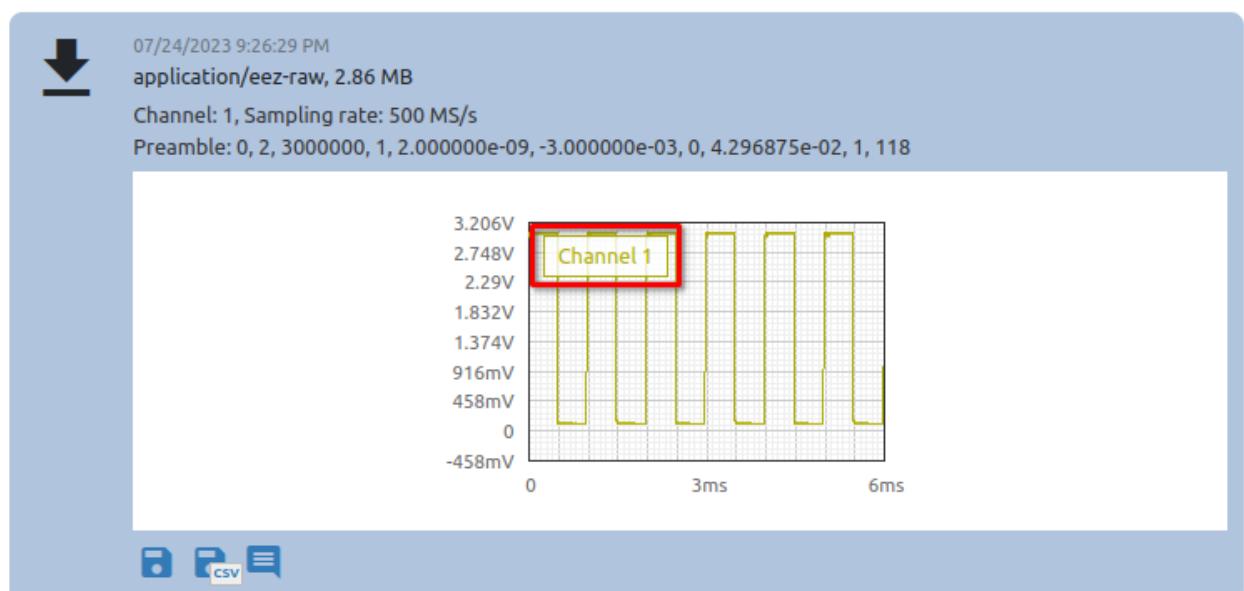
The color of the line in the chart if a dark background is selected.

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

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

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

Chart label:

**A1.2.13. Chart major subdivision horizontal** *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

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

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

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

Axes lines subdivision:

Dynamic

Fixed

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

Snap to grid

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

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

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

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

General**A1.2.19. 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**A1.2.20. 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.

A1.2.21. Outputs *Array*

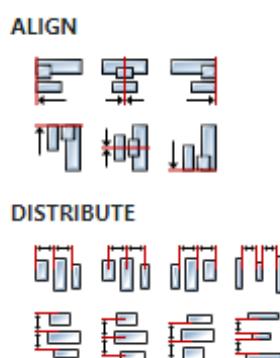
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component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

A1.2.22. 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.

Position and size**A1.2.23. 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.

**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.

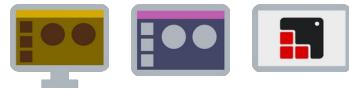
A1.4.2. id *DATA(string) / OPTIONAL*

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*

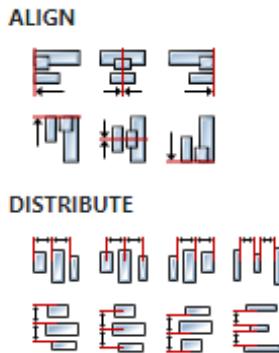
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the `Loop` component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the `Loop` component 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.

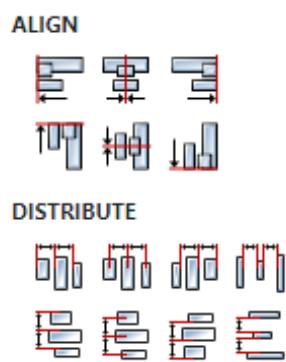
A3.2.4. Catch error *Boolean*

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. CollectStream



A4.1. 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.

A4.2. Properties

Specific

A4.2.1. Stream *EXPRESSION (any)*

A stream whose content will be concatenated into a string.

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 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.

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.

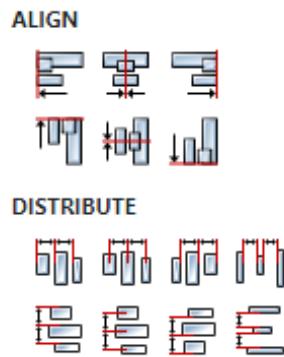
A4.2.5. Catch error *Boolean*

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

A4.4.2. 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).

A4.5. Examples

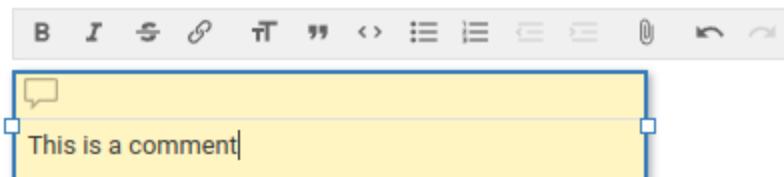
- *RegEx Stream*

A5. Comment



A5.1. Description

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



A6. Compare



A6.1. Description

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

A6.2. Properties

Specific

A6.2.1. A *EXPRESSION (any)*

Expression on the left side of the comparison.

A6.2.2. B *EXPRESSION (any)*

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

A6.2.3. 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`.

A6.2.4. 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

A6.2.5. 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

A6.2.6. 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.

A6.2.7. Outputs *Array*

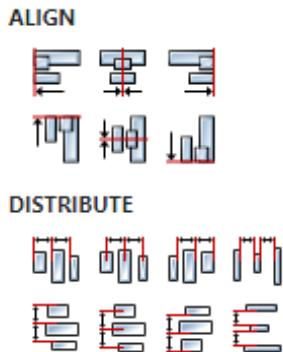
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `Variable` property instead of e.g. variable name. In that case, the *Loop* component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

A6.2.8. 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**A6.2.9. 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.

**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.

A6.4.2. True *SEQ / OPTIONAL*

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

A6.4.3. False *SEQ / OPTIONAL*

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

A7. ConnectInstrument



A7.1. 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.

A7.2. Properties

Specific

A7.2.1. Instrument *EXPRESSION (object:Instrument)*

Instrument object to connect to.

General

A7.2.2. 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

A7.2.3. 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.

A7.2.4. 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.

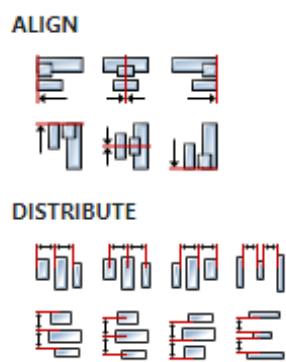
A7.2.5. Catch error *Boolean*

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

A7.2.6. 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.



A7.3. Inputs

A7.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A7.4. Outputs

A7.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A8. Constant



A8.1. Description

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

A8.2. Properties

Specific

A8.2.1. 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

A8.2.2. 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

A8.2.3. 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.

A8.2.4. 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.

A8.2.5. 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.

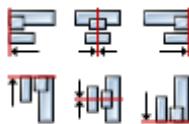
Position and size

A8.2.6. Align and distribute *Any*

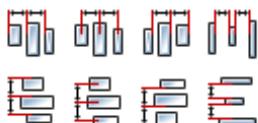
Alignment icons and component distribution. Alignment icons appear when two or more compo-

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

ALIGN



DISTRIBUTE



A8.3. Inputs

A8.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A8.4. Outputs

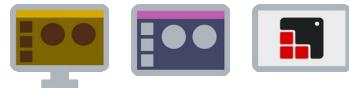
A8.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

A8.4.2. value *DATA(any) / MANDATORY*

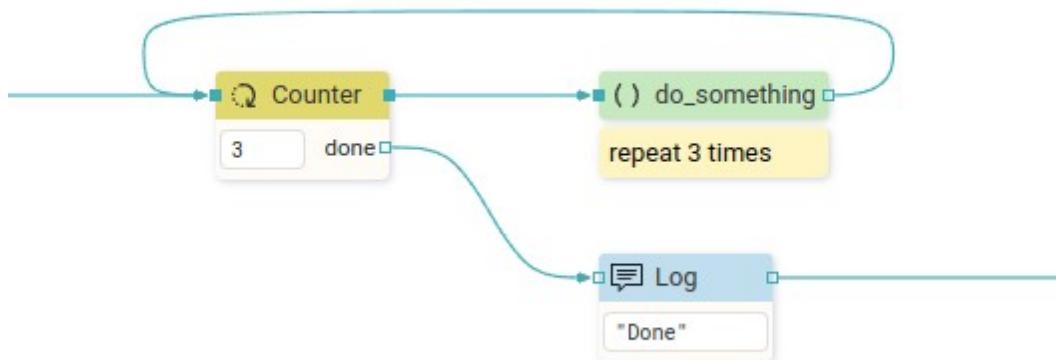
Data output through which the set constant is passed.

A9. Counter



A9.1. Description

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



A9.2. Properties

Specific

A9.2.1. Count value *EXPRESSION (integer)*

Expression that defines the number of repetitions in the loop.

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*

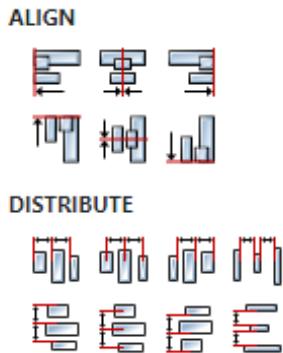
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *variable* property instead of e.g. variable name. In that case, the *Loop* component 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 / MANDATORY**

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

A9.4.2. done SEQ / OPTIONAL

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

A10. CSVParse



A10.1. Description

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

A10.2. Properties

Specific

A10.2.1. Input *EXPRESSION (string)*

CSV string to be parsed.

A10.2.2. Delimiter *EXPRESSION (string)*

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

A10.2.3. 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).

A10.2.4. 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

A10.2.5. 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

A10.2.6. 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.

A10.2.7. 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.

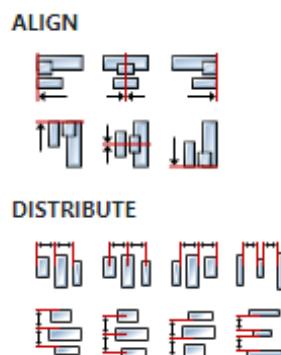
A10.2.8. Catch error *Boolean*

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

A10.2.9. 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.



A10.3. Inputs

A10.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A10.3.2. 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.

A10.4. Outputs

A10.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A10.4.2. 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:

FLOW

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:

The screenshot shows the Project editor interface with two main panels: 'Variables' on the left and 'Properties' on the right.

Variables Panel:

- Header: (x) Variables
- Buttons: Global (1), Local, Structs (1), Enums
- Search bar: 🔍
- Table: Shows one entry, 'CountryCity', which is currently selected.

Properties Panel:

- Header: Properties
- Section: Structure: CountryCity
- Fields Table:

Name	Type
country	string
city	string

A10.5. Examples

- CSV

A11. CSVStringify



A11.1. Description

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

A11.2. Properties

Specific

A11.2.1. Input *EXPRESSION (any)*

Flow value that will be converted into a CSV string.

A11.2.2. Delimiter *EXPRESSION (string)*

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

A11.2.3. Header *EXPRESSION (boolean)*

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

A11.2.4. 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

A11.2.5. 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

A11.2.6. 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.

A11.2.7. 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.

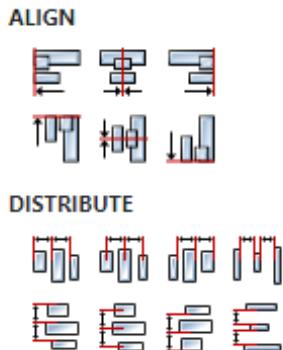
A11.2.8. Catch error *Boolean*

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

A11.2.9. 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.



A11.3. Inputs

A11.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A11.3.2. 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.

A11.4. Outputs

A11.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A11.4.2. result DATA(string) / MANDATORY

The constructed CSV string is sent through this output.

A11.5. Examples

- CSV

A12. DateNow



A12.1. Description

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

A12.2. Properties

General

A12.2.1. 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

A12.2.2. 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.

A12.2.3. 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.

A12.2.4. Catch error `Boolean`

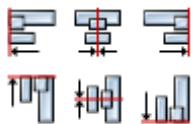
If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

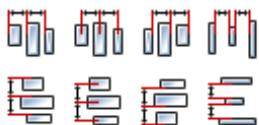
A12.2.5. Align and distribute Any

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



A12.3. Inputs

A12.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A12.4. Outputs

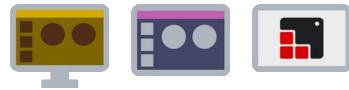
A12.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A12.4.2. value DATA(date) / MANDATORY

Data output through which the current time is passed.

A13. Delay



A13.1. Description

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

A13.2. Properties

Specific

A13.2.1.Milliseconds *EXPRESSION (integer)*

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

General

A13.2.2. 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

A13.2.3. 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.

A13.2.4. 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.

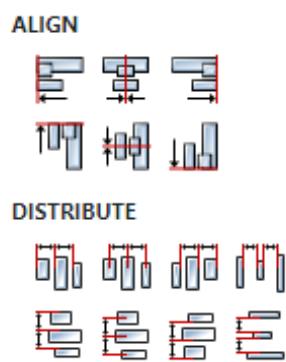
A13.2.5. Catch error *Boolean*

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

A13.2.6. 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.



A 13.3. Inputs

A 13.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A 13.4. Outputs

A 13.4.1. seqout SEQ / MANDATORY

A standard sequence output.

A14. DisconnectInstrument



A14.1. 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.

A14.2. Properties

Specific

A14.2.1. Instrument *EXPRESSION (object:Instrument)*

Instrument object to disconnect from.

General

A14.2.2. 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

A14.2.3. 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.

A14.2.4. 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.

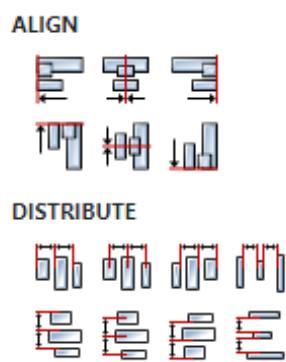
A14.2.5. Catch error *Boolean*

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

A14.2.6. 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.



A14.3. Inputs

A14.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A14.4. Outputs

A14.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A15. DynamicCallAction



A15.1. 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.

A15.2. Properties

Specific

A15.2.1. Action *EXPRESSION (string)*

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

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 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.

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.

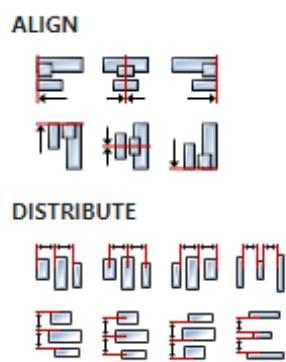
A15.2.5. Catch error *Boolean*

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 / OPTIONAL

A standard sequence input.

A15.4. Outputs

A15.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A16. End



A16.1. 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.

A16.2. Properties

General

A16.2.1. 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

A16.2.2. 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.

A16.2.3. 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.

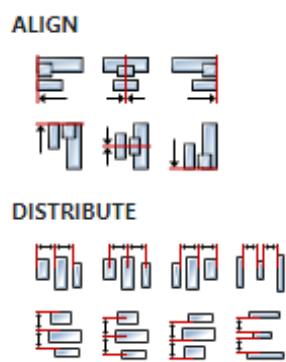
A16.2.4. Catch error Boolean

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

A16.2.5. 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.



A16.3. Inputs

A16.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A16.4. Outputs

A17. Error



A17.1. 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.

A17.2. Properties

Specific

A17.2.1. Message *EXPRESSION (string)*

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

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.

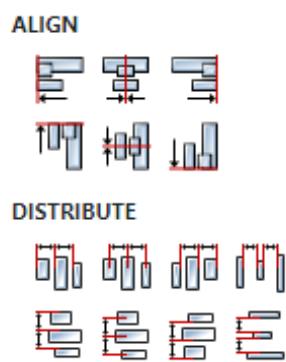
A17.2.5. Catch error *Boolean*

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.

A18. Eval JS



A18.1. Description

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

A18.2. Properties

Specific

A18.2.1. 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

A18.2.2. 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

A18.2.3. 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.

A18.2.4. 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.

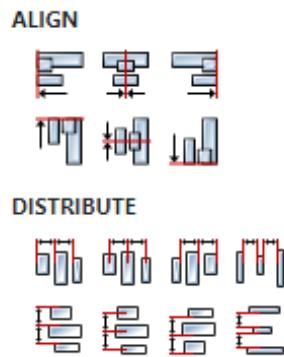
A18.2.5. Catch error *Boolean*

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

A18.2.6. 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.



A 18.3. Inputs

A 18.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A 18.4. Outputs

A 18.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A 18.4.2. 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.

A19. Evaluate



A19.1. Description

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

A19.2. Properties

Specific

A19.2.1. Expression *EXPRESSION (any)*

Expression to be evaluated.

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.

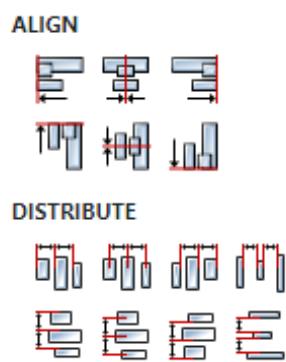
A19.2.5. Catch error *Boolean*

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

A19.4. Outputs

A19.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A19.4.2. result DATA(any) / MANDATORY

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

A20. ExecuteCommand



A20.1. 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.

A20.2. Properties

Specific

A20.2.1. Command *EXPRESSION (string)*

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

A20.2.2. Arguments *EXPRESSION (array:string)*

Array of string arguments that is passed to the command.

General

A20.2.3. 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

A20.2.4. 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.

A20.2.5. 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.

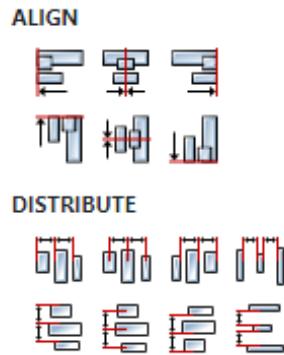
A20.2.6. Catch error *Boolean*

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

A20.2.7. 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.



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. 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.

A20.4.3. 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.

A20.4.4. 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.

A20.5. Examples

- *RegExp Stream*

A21. FileAppend



A21.1. 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.

A21.2. Properties

Specific

A21.2.1. File path *EXPRESSION (string)*

The full path of the file to be written.

A21.2.2. 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.

A21.2.3. 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

A21.2.4. 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

A21.2.5. 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.

A21.2.6. 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.

A21.2.7. Catch error *Boolean*

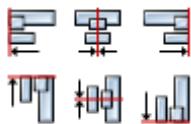
If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

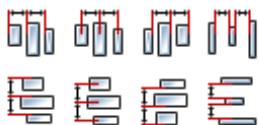
A21.2.8. Align and distribute Any

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



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.

A22. FileOpenDialog



A22.1. Description

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

A22.2. Properties

General

A22.2.1. 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

A22.2.2. 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.

A22.2.3. 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.

A22.2.4. Catch error *Boolean*

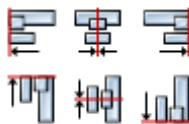
If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

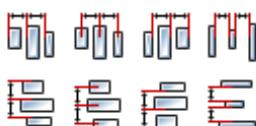
A22.2.5. Align and distribute Any

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



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. file_path DATA(string) / MANDATORY

Output to which the set file path is sent.

A23. FileRead



A23.1. Description

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

A23.2. Properties

Specific

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

The full path of the file to be read.

A23.2.2. 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

A23.2.3. 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

A23.2.4. 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.

A23.2.5. 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.

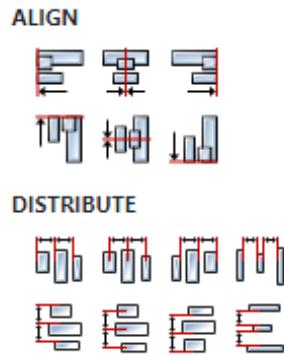
A23.2.6. Catch error *Boolean*

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

A23.2.7. 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.



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.

A23.4.2. content DATA(any) / MANDATORY

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

A23.5. Examples

- JSON
- CSV
- EEZ Chart

A24. FileSaveDialog



A24.1. Description

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

A24.2. Properties

Specific

A24.2.1. File name *EXPRESSION (string)*

The file name to be used by default.

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.

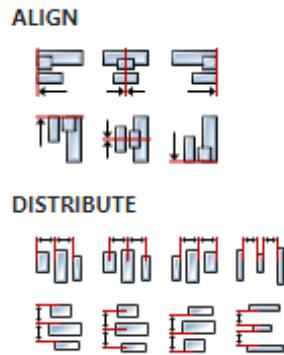
A24.2.5. Catch error *Boolean*

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.

A24.5. Examples

- *Screen Capture*

A25. FileWrite



A25.1. Description

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

A25.2. Properties

Specific

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

The full path of the file to be written.

A25.2.2. 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.

A25.2.3. 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

A25.2.4. 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

A25.2.5. 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.

A25.2.6. 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.

A25.2.7. Catch error *Boolean*

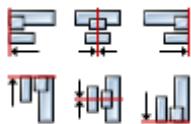
If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

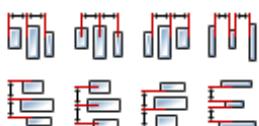
A25.2.8. Align and distribute Any

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



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.

A25.5. Examples

- CSV
- Screen Capture

A26. GetInstrument



A26.1. 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: Instruments (selected), Add Instrument, Deleted Instruments, History, Shortcuts and Groups, Notebooks, and a close button. Below the tabs, there is a card labeled "Selected instrument" containing a thumbnail of the instrument and the text "BB3 ATE Disconnected". To the right of this card is a sidebar with sections for Actions (Open in Tab, Open in New Window, Delete), Properties (Instrument: EEZ BB3 STM32, ID: 55, Label: BB3 ATE, IDN: Envox,EEZ BB3 (STM32),001C00393385107, Auto connect checked), and Connection (Interface: Ethernet). The "Instrument ID" field is highlighted with a red box.

The screenshot shows the instrument terminal window. The title bar says "Instrument ID" and the header bar says "BB3 ATE". The main area displays a history event: "LATEST HISTORY EVENT" at "04/10/2022 5:51:31 PM" with the message "DISCONNECTED after 58 seconds". On the left, there is a sidebar with links: Start Page, Dashboard, and Terminal. The "Instrument ID" header is also highlighted with a red box.

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.

A26.2. Properties

Specific

A26.2.1. Instrument ID *EXPRESSION (string)*

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

General

A26.2.2. 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**A26.2.3. 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.

A26.2.4. Outputs *Array*

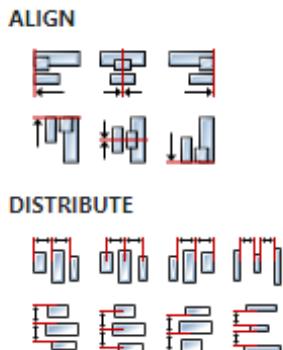
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the `variable` property instead of e.g. variable name. In that case, the *Loop* component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

A26.2.5. 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**A26.2.6. 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.

**A26.3. Inputs****A26.3.1. seqin** *SEQ / MANDATORY*

A standard sequence input.

A26.4. Outputs

A26.4.1. seqout SEQ / OPTIONAL

A standard sequence output

A26.4.2. instrument DATA(*object:Instrument*) / MANDATORY

The retrieved object is sent to this output.

A27. GetInstrumentProperties



A27.1. 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 LabVIEW Catalog interface. On the left, there is a list of instruments: Rigol DS1074B (Installed), Rigol DS1074Z (Installed), and Rigol DS1074Z-S (Installed). The Rigol DS1074Z is selected. On the right, detailed information about the Rigol DS1074Z is displayed, including its version (1.0.2), a description of its features (4 channels, 70-100 MHz), and its extendability. Below this is a code snippet showing the JSON structure of the instrument's properties. A red arrow points to the "Properties" tab in the code editor.

```
1 {  
2   "connection": {  
3     "ethernet": {  
4       "port": 5555  
5     },  
6     "usbTMC": {  
7       "idVendor": "0x1ab1",  
8       "idProduct": "0x04ce"  
9     }  
10 }
```

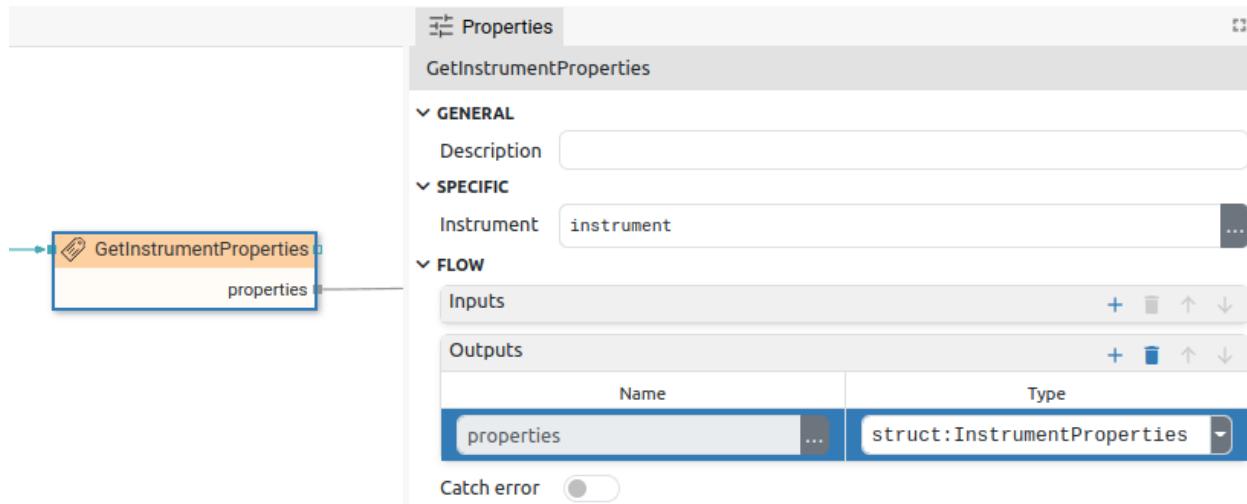
Now it is necessary to define the Flow variable type in which we want to store the properties we are interested in. In this case, we define the type `struct:InstrumentProperties` defined as follows:

The screenshot shows the LabVIEW Properties palette. Under the "Structs" tab, a new structure named "InstrumentProperties" is defined. It contains a single field named "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 LabVIEW Properties palette. Under the "Structs" tab, a new structure named "InstrumentPropertiesChannel" is defined. It contains 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`.

A27.2. Properties

Specific

A27.2.1. Instrument *EXPRESSION (object:Instrument)*

The instrument whose properties will be retrieved.

General

A27.2.2. 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

A27.2.3. 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.

A27.2.4. 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.

A27.2.5. 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-

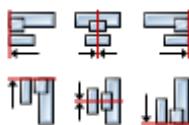
put. The data that will be passed through that output is the textual description of the error.

Position and size

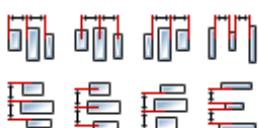
A27.2.6. Align and distribute Any

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



A27.3. Inputs

A27.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A27.4. Outputs

A27.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A27.4.2. properties DATA(any) / MANDATORY

Retrieved properties are sent to this output.

A27.5. Examples

- Rigol Waveform Data

A28. HTTP



A28.1. Description

Sends HTTP requests and returns the response.

A28.2. Properties

Specific

A28.2.1. Method *Enum*

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

A28.2.2. Url *EXPRESSION (string)*

The url of the request.

A28.2.3. Headers *Array*

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

A28.2.4. Body *EXPRESSION (string)*

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

General

A28.2.5. 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

A28.2.6. 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.

A28.2.7. 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.

A28.2.8. Catch error *Boolean*

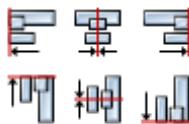
If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

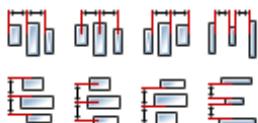
A28.2.9. Align and distribute Any

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



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.

A28.4.2. status DATA(integer) / OPTIONAL

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

A28.4.3. result DATA(string) / OPTIONAL

Message body of received response.

A28.5. Examples

- Simple HTTP

A29. Input



A29.1. Description

Adds data input to a user action or user widget.

A29.2. Properties

Specific

A29.2.1. Name *String*

Input name.

A29.2.2. Type *String*

Input data type.

General

A29.2.3. 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

A29.2.4. 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.

A29.2.5. 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.

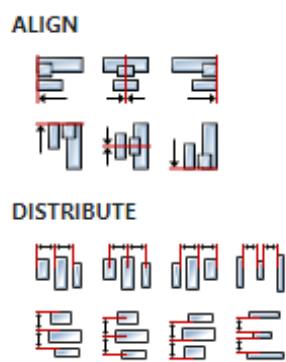
A29.2.6. Catch error *Boolean*

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

A29.2.7. 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.



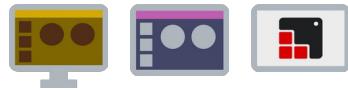
A29.3. Inputs

A29.4. Outputs

A29.4.1. seqout SEQ / MANDATORY

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

A30. IsTrue



A30.1. 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.

A30.2. Properties

Specific

A30.2.1. Value *EXPRESSION (boolean)*

Expression whose result is tested.

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 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.

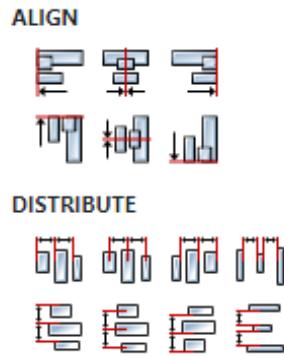
A30.2.5. Catch error *Boolean*

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 output. 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.



A30.3. Inputs

A30.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A30.3.2. 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.

A30.4. Outputs

A30.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A30.4.2. Yes SEQ / OPTIONAL

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

A30.4.3. No SEQ / OPTIONAL

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

A31. JSONParse



A31.1. Description

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

A31.2. Properties

Specific

A31.2.1. Value *EXPRESSION (string)*

JSON string to be parsed.

General

A31.2.2. 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

A31.2.3. 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.

A31.2.4. 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.

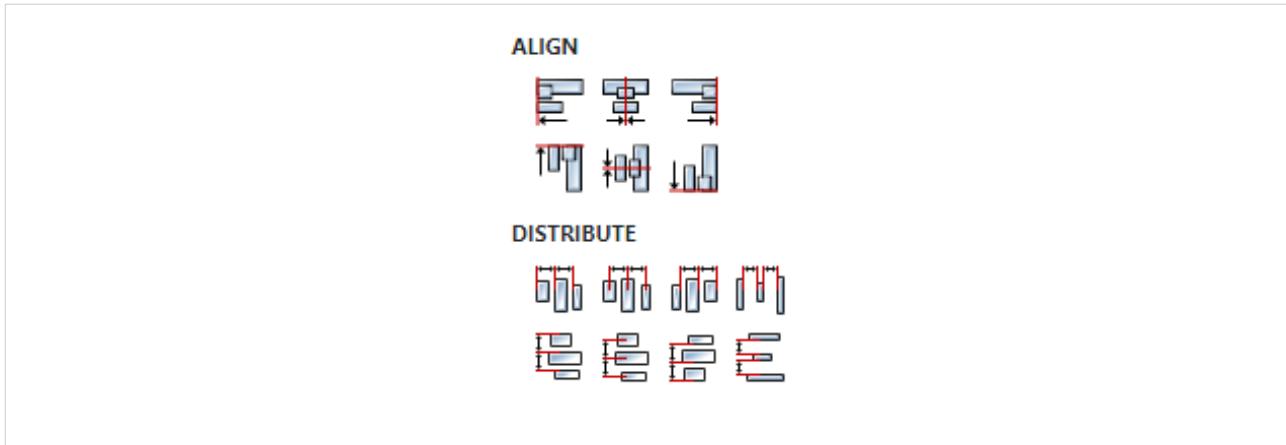
A31.2.5. Catch error *Boolean*

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

A31.2.6. 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.



A31.3. Inputs

A31.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A31.3.2. 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.

A31.4. Outputs

A31.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A31.4.2. 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:

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:

The screenshot shows the Project editor interface with the 'Structs' tab selected. A structure named 'CountryCity' is defined with two fields: 'country' (Type: string) and 'city' (Type: string).

Name	Type
country	string
city	string

A31.5. Examples

- `JSON`

A32. JSONStringify



A32.1. Description

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

A32.2. Properties

Specific

A32.2.1. Value *EXPRESSION (any)*

Flow value that will be converted into a JSON string.

A32.2.2. Indentation *EXPRESSION (integer)*

The indentation to be used in the generated JSON string.

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 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.

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.

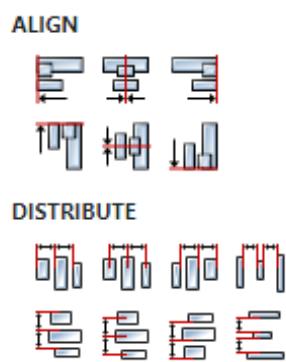
A32.2.6. Catch error *Boolean*

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

A32.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A32.4. Outputs

A32.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

A32.4.2. result *DATA(string) / MANDATORY*

The constructed JSON string is sent through this output.

A32.5. Examples

- *JSON*

A33. Log



A33.1. Description

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

A33.2. Properties

Specific

A33.2.1. Value *EXPRESSION (string)*

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

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.

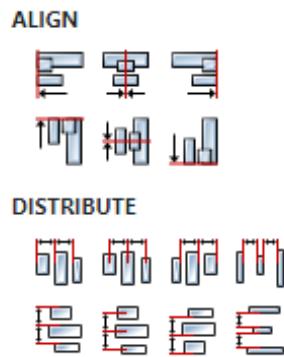
A33.2.5. Catch error *Boolean*

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 / OPTIONAL

A standard sequence input.

A33.3.2. 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.

A33.4. Outputs

A33.4.1. seqout SEQ / OPTIONAL

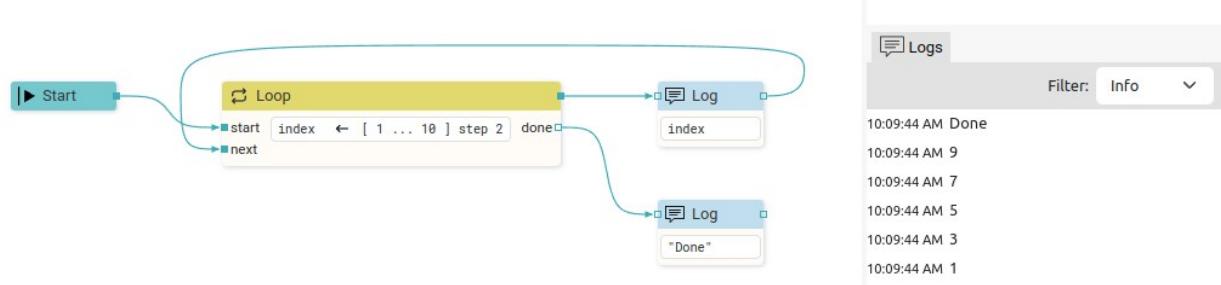
A standard sequence output.

A34. Loop



A34.1. 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.



A34.2. Properties

Specific

A34.2.1. 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.

A34.2.2. From *EXPRESSION (integer)*

The initial value of the variable.

A34.2.3. To *EXPRESSION (integer)*

The final value of the variable.

A34.2.4. Step *EXPRESSION (integer)*

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

General

A34.2.5. 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

A34.2.6. 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.

A34.2.7. 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.

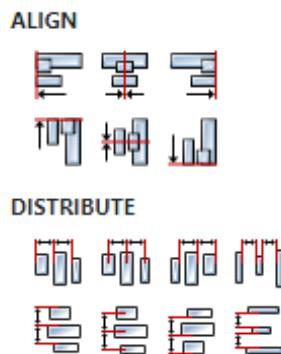
A34.2.8. Catch error Boolean

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

A34.2.9. 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.



A34.3. Inputs

A34.3.1. start SEQ / MANDATORY

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

A34.3.2. 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.

A34.4. Outputs

A34.4.1. seqout SEQ / MANDATORY

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

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

A34.5. Examples

- *Loop*



A35.1. Description

Performs one or more LVGL specific actions.

A35.2. Properties

Specific

A35.2.1. Actions *Array*

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

- **Change Screen** Changes the active page. The following options are available:
 - **Previous screen** - if it is checked, it will go to the previous screen, otherwise you should select the page you want to display.
 - **Screen** - name of the page to be displayed.
- **Fade mode** - selection of animation when moving from the previous page to a new page. The following options are available:
 - **None** - switch immediately after delay ms
 - **Over left / Over right / Over top / Over bottom** - move the new page over the others towards the given direction.
 - **Move Left/Right/Top/Bottom** - move both the old and new pages towards the given direction.
 - **Fade in / Fade out** - fade the new page over the old page, or vice versa.
 - **Out left / Out right / Out top / Out bottom** - move out the old page over the current one towards the given direction.
- **Speed** - animation duration in milliseconds.
- **Delay** - delay in milliseconds before the animation starts.
- **Play Animation** Animates the selected Widget property. The following options are available:
 - **Target** - Widget whose property is animated
 - **Property** - Widget property that is animated.
 - **Start** - initial property value.
 - **End** - the final value of the property.
 - **Delay** - delay in milliseconds before the animation starts.
 - **Time** - the total duration of the animation 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** - determines the animation curve. The following options are available:
 - **Linear** - calculate the current value of an animation applying linear characteristic
 - **Ease in** - calculate the current value of an animation slowing down the start phase
 - **Ease out** - calculate the current value of an animation slowing down the end phase
 - **Ease in out** - calculate the current value of an animation applying an "S" characteristic (cosine)
 - **Overshoot** - calculate the current value of an animation with overshoot at the end

- `Bounce` - calculate the current value of an animation with 3 bounces
- `Set Property` Changes the value of the selected property for the selected Widget. The following options are available:
 - `Target type` - The type of Widget that changes.
 - `Target` - Widget whose property is changed.
 - `Property` - the property that is being changed.
 - `Value` - new property value.
 - `Animated` - if there is a possibility to animate the property, then you can choose to make the change animated. For example for *Slider*, changing position *slider* (Value property) can be animated.

General

A35.2.2. 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

A35.2.3. 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.

A35.2.4. 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.

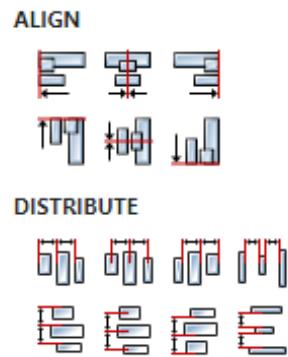
A35.2.5. Catch error `Boolean`

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

A35.2.6. 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.



A35.3. Inputs

A35.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A35.4. Outputs

A35.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A35.5. Examples

- *Change Screen*

A36. MQTTConnect



A36.1. 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.

A36.2. Properties

Specific

A36.2.1. Connection *EXPRESSION (object:MQTTConnection)*

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

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 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.

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.

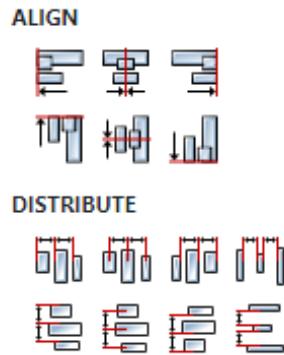
A36.2.5. Catch error *Boolean*

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.

A36.4. Outputs

A36.4.1. 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.

A36.5. Examples

- *MQTT*

A37. MQTTDisconnect



A37.1. Description

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

A37.2. Properties

Specific

A37.2.1. Connection *EXPRESSION (object:MQTTConnection)*

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

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.

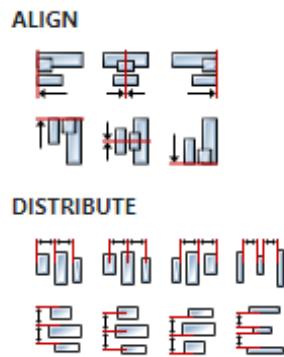
A37.2.5. Catch error *Boolean*

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.

A37.4. Outputs

A37.4.1. 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.

A37.5. Examples

- *MQTT*

A38. MQTTEvent



A38.1. 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.

A38.2. Properties

Specific

A38.2.1. Connection *EXPRESSION (object:MQTTConnection)*

MQTT connection to the server whose events are to be handled.

A38.2.2. 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

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.

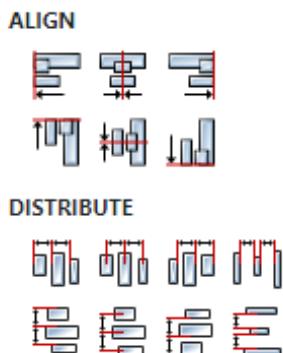
A38.2.6. Catch error Boolean

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.

A38.5. Examples

- *MQTT*

A39. MQTTInit



A39.1. 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.

A39.2. Properties

Specific

A39.2.1. Connection *ASSIGNABLE EXPRESSION (object:MQTTConnection)*

Connection object of type `object:MQTTConnection` which will be created and initialized.

A39.2.2. Protocol *EXPRESSION (string)*

The protocol used for the connection. Possible values are `"mqtt"` or for secure connection `"mqtts"`

A39.2.3. Host *EXPRESSION (string)*

The name of the MQTT server to connect to.

A39.2.4. Port *EXPRESSION (integer)*

The port number that will be used for the connection. The default is `1883`.

A39.2.5. User name *EXPRESSION (string)*

Username to be used for connection authorization. Can be left blank if not used.

A39.2.6. Password *EXPRESSION (string)*

User password to be used for connection authorization. Can be left blank if not used.

General

A39.2.7. 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

A39.2.8. 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.

A39.2.9. 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.

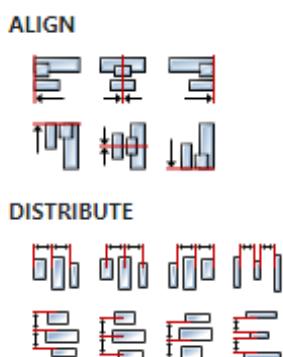
A39.2.10. Catch error Boolean

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

A39.2.11. 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.



A39.3. Inputs

A39.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A39.4. Outputs

A39.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A39.5. Examples

- *MQTT*

A40. MQTT Publish



A40.1. Description

Publishing a message in the selected topic.

A40.2. Properties

Specific

A40.2.1. Connection *EXPRESSION (object:MQTTConnection)*

The name of the connection to the MQTT server.

A40.2.2. Topic *EXPRESSION (string)*

The topic under which the message will be published.

A40.2.3. Payload *EXPRESSION (string)*

Message to be published.

General

A40.2.4. 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

A40.2.5. 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.

A40.2.6. 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.

A40.2.7. 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.

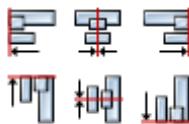
Position and size

A40.2.8. Align and distribute *Any*

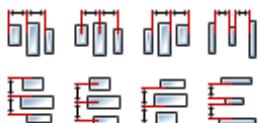
Alignment icons and component distribution. Alignment icons appear when two or more compo-

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

ALIGN



DISTRIBUTE



A40.3. Inputs

A40.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A40.4. Outputs

A40.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output

A40.5. Examples

- *MQTT*

A41. MQTTSubscribe



A41.1. 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.

A41.2. Properties

Specific

A41.2.1. Connection *EXPRESSION (object:MQTTConnection)*

The name of the connection to the MQTT server.

A41.2.2. 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

A41.2.3. 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

A41.2.4. 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.

A41.2.5. 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.

A41.2.6. Catch error *Boolean*

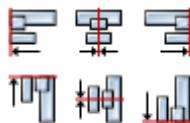
If this checkbox is enabled then an `@Error` output will be added to the component and if an error occurs in this component during the execution of the Flow, the Flow will continue through that output. The data that will be passed through that output is the textual description of the error.

Position and size

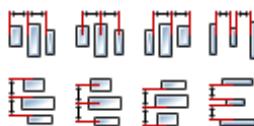
A41.2.7. Align and distribute *Any*

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



A41.3. Inputs

A41.3.1. seqin *SEQ / OPTIONAL*

A standard sequence input.

A41.4. Outputs

A41.4.1. seqout *SEQ / OPTIONAL*

A standard sequence output.

A41.5. Examples

- *MQTT*

A42. MQTTUnsubscribe



A42.1. Description

Unsubscribe from a topic.

A42.2. Properties

Specific

A42.2.1. Connection *EXPRESSION (object:MQTTConnection)*

The name of the connection to the MQTT server.

A42.2.2. Topic *EXPRESSION (string)*

Topic to unsubscribe from.

General

A42.2.3. 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

A42.2.4. 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.

A42.2.5. 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.

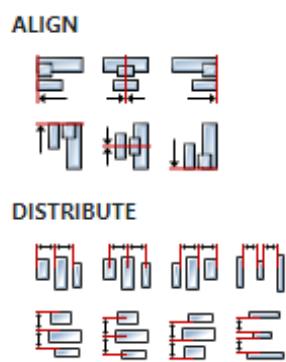
A42.2.6. Catch error *Boolean*

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

A42.2.7. 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.



A42.3. Inputs

A42.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A42.4. Outputs

A42.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A42.5. Examples

- MQTT

A43. NoOp



A43.1. Description

This action does nothing, i.e. Flow execution continues through `seqout`.

A43.2. Properties

Specific

A43.2.1. Name *String*

The name displayed in the component view within the Flow.

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 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.

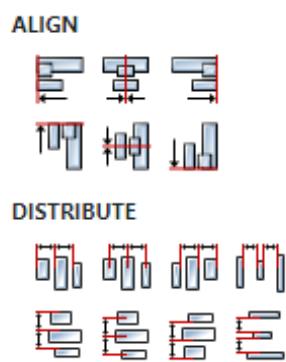
A43.2.5. Catch error *Boolean*

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

A44. OnEvent



A44.1. Description

It is used to process events that can be broadcast within the page where the Action is located.

A44.2. Properties

Specific

A44.2.1. 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

A44.2.2. 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

A44.2.3. 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.

A44.2.4. 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.

A44.2.5. 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.

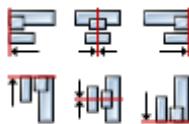
Position and size

A44.2.6. Align and distribute *Any*

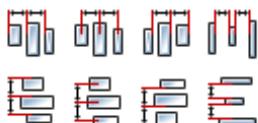
Alignment icons and component distribution. Alignment icons appear when two or more compo-

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

ALIGN



DISTRIBUTE



A44.3. Outputs

A44.3.1. seqout SEQ / MANDATORY

A standard sequence output. Flow execution continues through this output when the selected event is emitted.

A44.3.2. 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](#)).

A44.4. Examples

- *Tetris*

A45. Output



A45.1. Description

Adds data output to a user action or user widget.

A45.2. Properties

Specific

A45.2.1. Name *String*

Output name.

A45.2.2. Type *String*

Output data type.

General

A45.2.3. 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

A45.2.4. 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.

A45.2.5. 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.

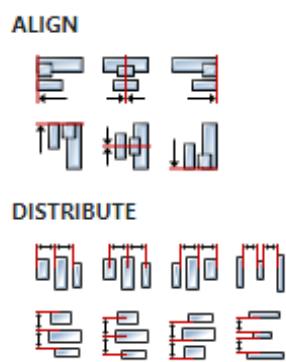
A45.2.6. Catch error *Boolean*

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

A45.2.7. 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.



A45.3. Inputs

A45.3.1. seqin SEQ / MANDATORY

Data is received through this input, which is then forwarded to the caller of the user action.

A46. OverrideStyle



A46.1. 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.

A46.2. Properties

Specific

A46.2.1. From *ObjectReference*

The style to be replaced.

A46.2.2. To *ObjectReference*

A new style that will replace the existing one.

General

A46.2.3. 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

A46.2.4. 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.

A46.2.5. 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.

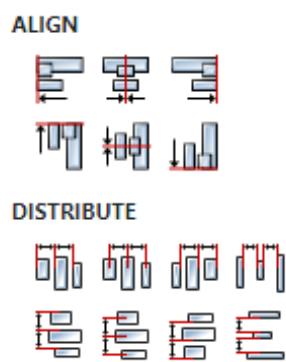
A46.2.6. Catch error *Boolean*

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

A46.2.7. 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.



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.

A47. PythonEnd



A47.1. Description

Stops a running python script.

A47.2. Properties

Specific

A47.2.1. 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

A47.2.2. 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

A47.2.3. 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.

A47.2.4. 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.

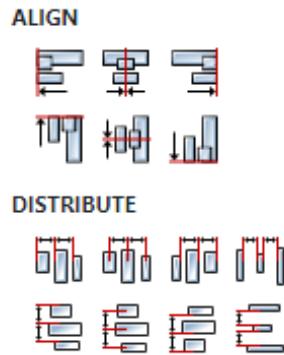
A47.2.5. Catch error *Boolean*

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

A47.2.6. 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.



A47.3. Inputs

A47.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A47.3.2. 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.

A47.4. Outputs

A47.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A48. PythonRun



A48.1. 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.

A48.2. Properties

Specific

A48.2.1. Script source option *Enum*

The source of the python script can be specified in three ways:

- Inline script
- Inline script as expression
- Script file

A48.2.2. Inline script *Python*

If `Inline script` was selected for `Script source option`, then the source code of the script should be entered here.

A48.2.3. 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.

A48.2.4. 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.

A48.2.5. 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

A48.2.6. 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

A48.2.7. 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.

A48.2.8. 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.

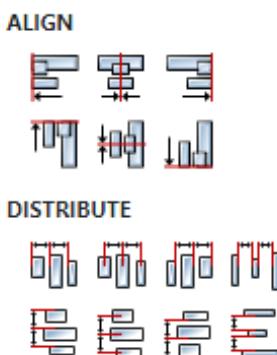
A48.2.9. Catch error Boolean

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

A48.2.10. 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.



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.

A48.4.2. handle DATA(integer) / OPTIONAL

Returns the handle of the running script used in *PythonEnd* and *PythonSendMessage* Actions.

A48.4.3. 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.

A48.5. Examples

- *Charts*

A49. PythonSendMessage



A49.1. Description

Sends a message from Flow to a running Python script.

A49.2. Properties

Specific

A49.2.1. 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.

A49.2.2. Message *EXPRESSION (string)*

Message to be sent.

General

A49.2.3. 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

A49.2.4. 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.

A49.2.5. 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.

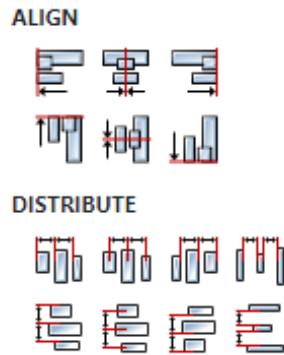
A49.2.6. Catch error *Boolean*

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

A49.2.7. 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.



A49.3. Inputs

A49.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A49.3.2. 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.

A49.4. Outputs

A49.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A49.5. Examples

- *Charts*

A50. ReadSetting



A50.1. 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

A50.2. Properties

Specific

A50.2.1. Key *EXPRESSION (string)*

A string containing the name of the key whose value is to be retrieved.

General

A50.2.2. 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

A50.2.3. 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.

A50.2.4. 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.

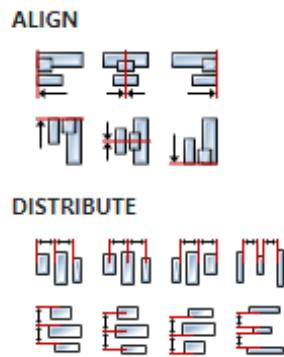
A50.2.5. Catch error *Boolean*

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

A50.2.6. 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.



A50.3. Inputs

A50.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A50.4. Outputs

A50.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A50.4.2. value DATA(any) / MANDATORY

The obtained `Value` of the defined `Key` is sent through this output.

A51. Regexp



A51.1. Description

Searches a set string or stream, using a pattern written according to the rules of the regular expression syntax.

A51.2. Properties

Specific

A51.2.1. Pattern *EXPRESSION (string)*

Regular expression used for searching.

A51.2.2. Text *EXPRESSION (string)*

The text to be searched can be a string or a stream.

A51.2.3. Global *EXPRESSION (boolean)*

This option determines whether only the first occurrence of the pattern or every occurrence of the pattern is searched.

A51.2.4. Case insensitive *EXPRESSION (boolean)*

This option determines whether the search will be case sensitive or not.

General

A51.2.5. 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

A51.2.6. 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.

A51.2.7. 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.

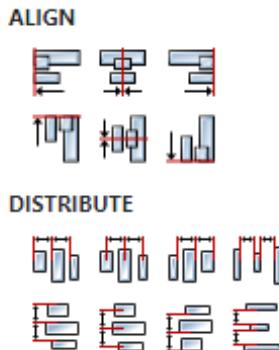
A51.2.8. Catch error *Boolean*

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

A51.2.9. 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.



A51.3. Inputs

A51.3.1. seqin SEQ / OPTIONAL

A standard sequence input. This input needs to be used once at the beginning.

A51.3.2. next SEQ / OPTIONAL

Use this input to get the next match.

A51.3.3. 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.

A51.4. Outputs

A51.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A51.4.2. 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.

A51.5. Examples

- *RegExp String*
- *RegExp Stream*



A52.1. 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.

A52.2. Properties

Specific

A52.2.1. 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.

A52.2.2. 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:

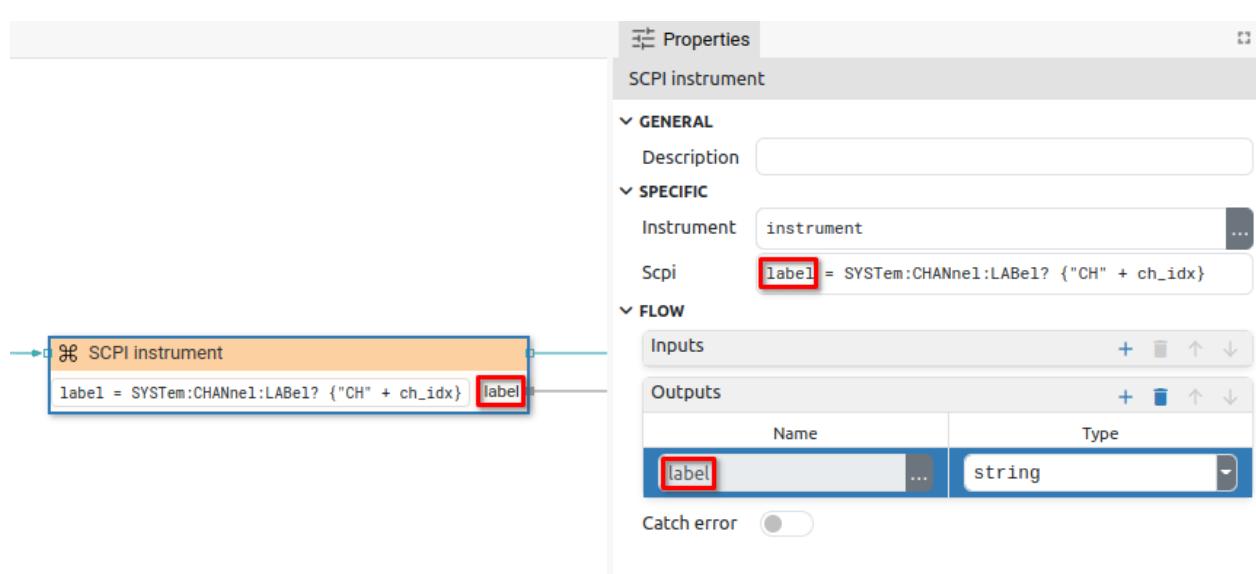


A flow expression is inserted inside an SCPI command/query by entering it inside curly brackets.

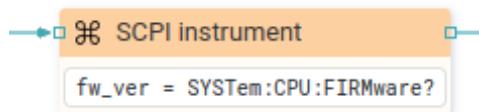
Also in the example above, a Flow `Catch Error` has been added to catch an error during the execution of the SCPI component.

For an SCPI query, it must be specified where the result is sent, and there we have two options:

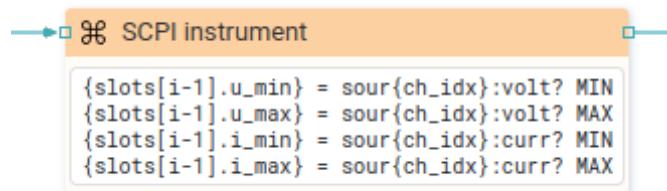
- Sending results to Flow output. It is necessary to add a new output using the "Flow - Outputs" section in the properties of this component, where it is necessary to write: `output_name=query?`. Here's an example, taken from the *BB3 Dashboard* example:



- 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.



A52.2.3. 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.

A52.2.4. 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

A52.2.5. 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

A52.2.6. 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.

A52.2.7. 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.

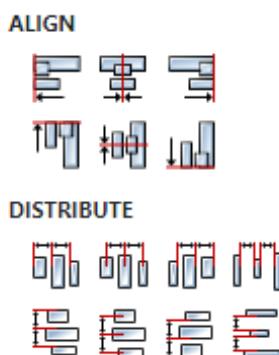
A52.2.8. Catch error *Boolean*

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

A52.2.9. 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.



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.

A52.5. Examples

- *BB3 Dashboard*
- *Plotly*
- *Rigol Waveform Data*
- *Screen Capture*

A53. SelectInstrument



A53.1. 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, the **Variables** panel lists several variables: `channels array:struct:Channel`, `CHUNK_MAX_POINTS integer`, `instrument object:Instrument` (which is highlighted), `instrumentProperties struct:InstrumentProperties`, `numDisplayedChannels integer`, `samplingRate integer`, `timeScale float`, and `view_selected_ch_idx integer`. On the right, the **Properties** panel shows the properties for the `instrument` variable:

Global variable: instrument	
GENERAL	
Id	
Name	instrument
Description	
Type	object:Instrument
Default value	null
<input type="checkbox"/> Persistent	

A red arrow points to the `Persistent` checkbox in the Properties panel.

A53.2. Properties

General

A53.2.1. 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

A53.2.2. 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.

A53.2.3. 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.

A53.2.4. Catch error *Boolean*

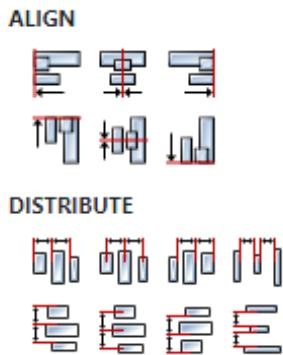
If this checkbox is enabled then an `@Error` output will be added to the component and if an error

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

A53.2.5. 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.



A53.3. Inputs

A53.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A53.4. Outputs

A53.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A53.4.2. instrument DATA(object:Instrument) / MANDATORY

The selected instrument is sent to this output.

A54. SelectLanguage



A54.1. 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.

A54.2. Properties

Specific

A54.2.1. Language *EXPRESSION (any)*

ID of the language that will become the new active language.

General

A54.2.2. 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

A54.2.3. 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.

A54.2.4. 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.

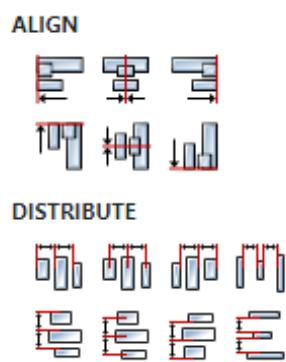
A54.2.5. Catch error *Boolean*

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

A54.2.6. 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.



A54.3. Inputs

A54.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A54.4. Outputs

A54.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A54.5. Examples

- *Multi-Language*
- *Multi-Language Dashboard*

A55. SerialConnect



A55.1. 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.

A55.2. Properties

Specific

A55.2.1. Connection *EXPRESSION (object:SerialConnection)*

The name of the connection to be used for serial communication.

General

A55.2.2. 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

A55.2.3. 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.

A55.2.4. 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.

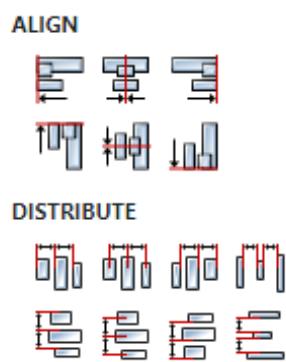
A55.2.5. Catch error *Boolean*

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

A55.2.6. 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.



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.

A55.5. Examples

- *SerialPort*

A56. SerialDisconnect



A56.1. Description

Performs disconnection from serial port, after which Flow execution continues through `seqout` output.

A56.2. Properties

Specific

A56.2.1. Connection *EXPRESSION (object:SerialConnection)*

The name of the connection that will be terminated.

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 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.

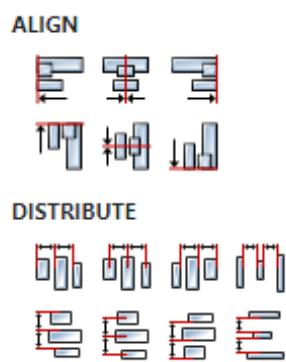
A56.2.5. Catch error *Boolean*

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.

A56.5. Examples

- *SerialPort*

A57. SerialInit



A57.1. 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.

A57.2. Properties

Specific

A57.2.1. Connection *ASSIGNABLE EXPRESSION (object:SerialConnection)*

Connection object of type `object:SerialConnection` to be created and initialized.

A57.2.2. Port *EXPRESSION (object:string)*

Serial port name.

A57.2.3. Baud rate *EXPRESSION (object:number)*

Serial port speed.

A57.2.4. Data bits *EXPRESSION (object:number)*

Serial port data bits. Allowed values are `5`, `6`, `7` or `8`.

A57.2.5. Stop bits *EXPRESSION (object:number)*

Serial port stop bits. Allowed values are `1` or `2`.

A57.2.6. Parity *EXPRESSION (object:string)*

Serial port parity. Allowed values are `"none"`, `"even"`, `"mark"`, `"odd"` or `"space"`

General

A57.2.7. 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

A57.2.8. 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.

A57.2.9. 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.

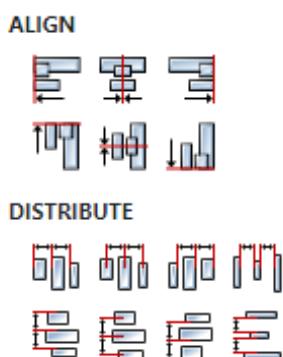
A57.2.10. Catch error Boolean

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

A57.2.11. 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.



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.

A58. SerialListPorts



A58.1. Description

Retrieves the list of serial ports detected on the system and sends it through `ports` output.

A58.2. Properties

General

A58.2.1. 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

A58.2.2. 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.

A58.2.3. 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.

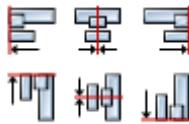
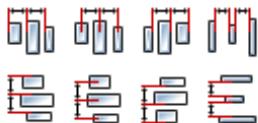
A58.2.4. Catch error Boolean

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

A58.2.5. 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****A58.3. Inputs****A58.3.1. seqin SEQ / OPTIONAL**

A standard sequence input.

A58.4. Outputs**A58.4.1. seqout SEQ / OPTIONAL**

A standard sequence output.

A58.4.2. 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.

A59. SerialRead



A59.1. Description

Sends the read stream received via the selected serial connection to the `data` output.

A59.2. Properties

Specific

A59.2.1. Connection *EXPRESSION (object:SerialConnection)*

The name of the serial connection.

General

A59.2.2. 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

A59.2.3. 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.

A59.2.4. 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.

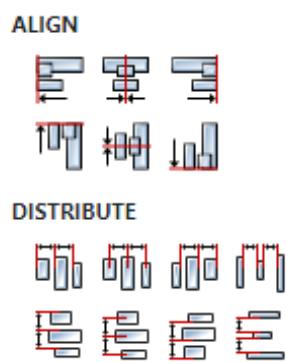
A59.2.5. Catch error *Boolean*

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

A59.2.6. 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.



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. data *DATA(string) / MANDATORY*

Output to which the read stream is sent.

A59.5. Examples

- *SerialPort*

A60. SerialWrite



A60.1. Description

Sends a string to the serial port.

A60.2. Properties

Specific

A60.2.1. Connection *EXPRESSION (object:SerialConnection)*

The name of the serial connection.

A60.2.2. Data *EXPRESSION (string)*

The string that is sent to the serial port.

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.

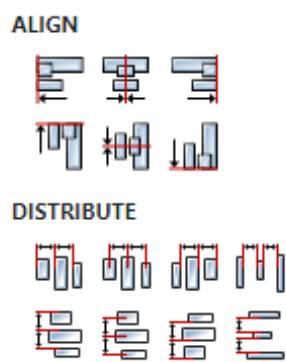
A60.2.6. Catch error *Boolean*

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.

A60.4. Outputs

A60.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A60.5. Examples

- *SerialPort*

A61. SetPageDirection



A61.1. Description

It is used to change the page layout from LTR (left to right) to RTL (right to left) and vice versa.

A61.2. Properties

Specific

A61.2.1. Direction *Enum*

Selected page layout: `LTR` or `RTL`.

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.

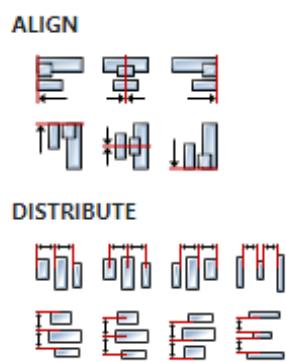
A61.2.5. Catch error *Boolean*

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 / OPTIONAL

A standard sequence input.

A61.4. Outputs

A61.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A62. SetVariable



A62.1. Description

It is used to set a new value to one or more variables.

A62.2. Properties

Specific

A62.2.1. 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

A62.2.2. 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

A62.2.3. 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.

A62.2.4. 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.

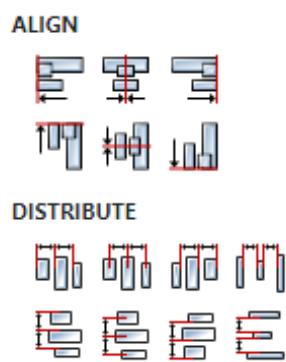
A62.2.5. Catch error *Boolean*

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

A62.2.6. 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.



A62.3. Inputs

A62.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A62.4. Outputs

A62.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A63. ShowFileInFolder



A63.1. Description

Displays the set file in the system file manager. When possible, the set file will also be selected.

A63.2. Properties

Specific

A63.2.1. File path *EXPRESSION (string)*

Path to the file that will be displayed in the system file manager.

General

A63.2.2. 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

A63.2.3. 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.

A63.2.4. 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.

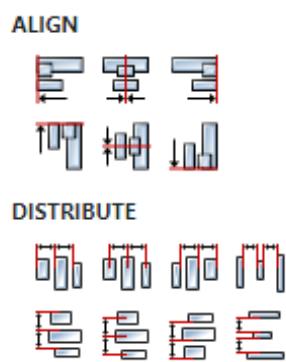
A63.2.5. Catch error *Boolean*

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

A63.2.6. 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.



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

- *Screen Capture*

A64. ShowKeyboard



A64.1. 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:



A64.2. Properties

Specific

A64.2.1. Label *EXPRESSION (string)*

The label that will be displayed on the keyboard page (e.g. the name of the parameter whose value is entered).

A64.2.2. Initial text *EXPRESSION (string)*

Initial (default) text that will be displayed in the input field.

A64.2.3. Min chars *EXPRESSION (integer)*

Defines the minimum length of the entered text.

A64.2.4. Max chars *EXPRESSION (integer)*

Defines the maximum length of the entered text.

A64.2.5. 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**A64.2.6. 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**A64.2.7. 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.

A64.2.8. Outputs *Array*

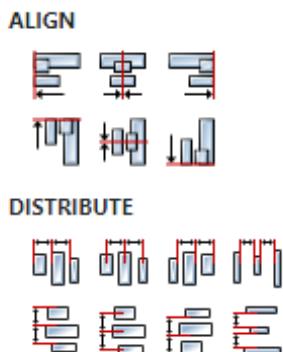
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component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

A64.2.9. 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**A64.2.10. 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.

**A64.3. Inputs****A64.3.1. seqin** *SEQ / MANDATORY*

A standard sequence input.

A64.4. Outputs

A64.4.1. result DATA(string) / MANDATORY

Output to which the entered text is sent.

A64.4.2. canceled DATA(null) / OPTIONAL

Flow execution continues through this output if the cancel button is pressed.

A64.5. Examples

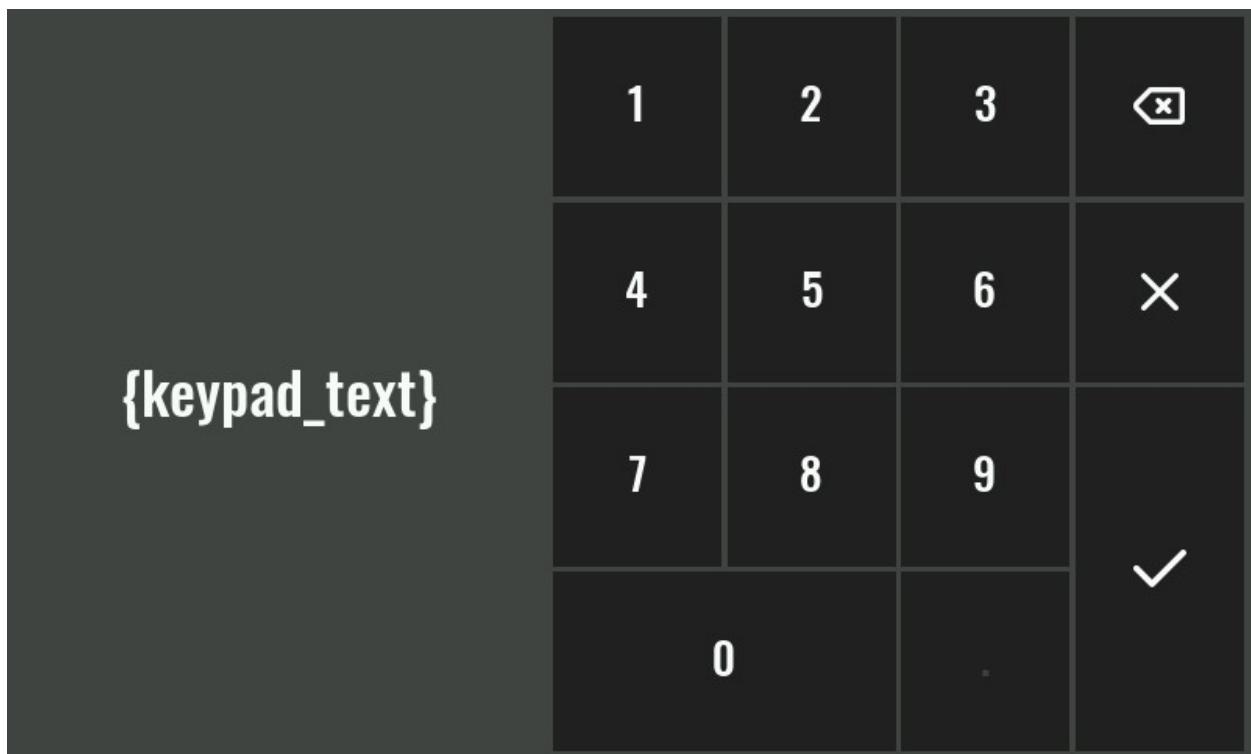
- *Keyboard, Keypad and Message Box*
- *stm32f469i-disco-eez-flow-demo*

A65. ShowKeypad



A65.1. 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:



A65.2. Properties

Specific

A65.2.1. Label *EXPRESSION (string)*

The label that will be displayed on the keyboard page (e.g. the name of the parameter whose value is entered).

A65.2.2. Initial value *EXPRESSION (float)*

Initial (default) number that will be displayed in the input field.

A65.2.3. Min *EXPRESSION (integer)*

The entered number must be greater than or equal to this number.

A65.2.4. Max *EXPRESSION (integer)*

The entered number must be less than or equal to this number.

A65.2.5. 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.

A65.2.6. Unit *EXPRESSION (string)*

Units that will be displayed when entering a number.

General**A65.2.7. 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**A65.2.8. 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.

A65.2.9. Outputs *Array*

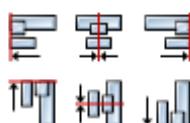
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component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

A65.2.10. 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**A65.2.11. 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**

A65.3. Inputs

A65.3.1. seqin SEQ / MANDATORY

A standard sequence input.

A65.4. Outputs

A65.4.1. result DATA(float) / MANDATORY

Output to which the entered numeric value is sent.

A65.4.2. canceled DATA(null) / OPTIONAL

Flow execution continues through this output if the cancel button is pressed.

A65.5. Examples

- *stm32f469i-disco-eez-flow-demo*
- *eyboard, Keypad and Message Box*

A66. ShowMessageBox



A66.1. Description

This Action is used to display *Info*, *Error* or *Question* message boxes.

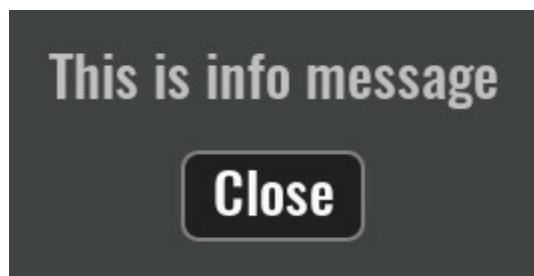
A66.2. Properties

Specific

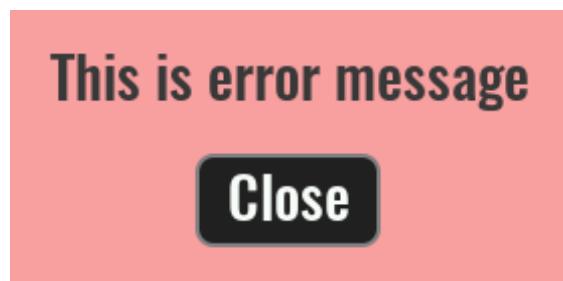
A66.2.1. Message type *Enum*

Defines the message box that will be displayed:

- `Info`



- `Error`



- `Question`



A66.2.2. Message *EXPRESSION (string)*

The content of the message to be displayed.

A66.2.3. 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**A66.2.4. 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**A66.2.5. 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.

A66.2.6. Outputs *Array*

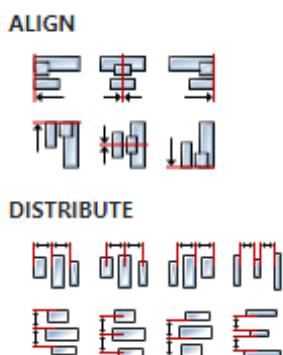
Additional component outputs that the user can add to send data through. Each output is assigned a name and type. An example of using this output is e.g. in the *Loop* component, where we can put the output name for the *Variable* property instead of e.g. variable name. In that case, the *Loop* component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

A66.2.7. 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**A66.2.8. 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.

**A66.3. Inputs****A66.3.1. seqin** *SEQ / MANDATORY*

A standard sequence input.

A66.4. Outputs

A66.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A66.5. Examples

- *Keyboard, Keypad and Message Box*

A67. ShowPage



A67.1. Description

This Action sets a new active page: the previous page will be hidden and the new page will be displayed.

A67.2. Properties

Specific

A67.2.1. Page *ObjectReference*

The name of the new page to be displayed.

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.

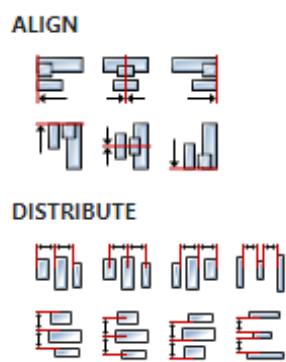
A67.2.5. Catch error *Boolean*

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 / MANDATORY

A standard sequence input.

A67.4. Outputs

A67.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A68. SortArray



A68.1. 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.

A68.2. Properties

Specific

A68.2.1. Array *EXPRESSION (array:any)*

Array variable to be sorted.

A68.2.2. Structure name *ObjectReference*

Select the name of the structure here when the array is a variable of type `array:struct`.

A68.2.3. Structure field name *Enum*

Select the name of the field to be sorted by if the array is a variable of type `array:struct`.

A68.2.4. Ascending *Boolean*

Sorting mode selection (ascending if enabled, otherwise descending).

A68.2.5. Ignore case *Boolean*

Specifies whether letter case is ignored if strings are sorted or not.

General

A68.2.6. 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

A68.2.7. 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.

A68.2.8. 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.

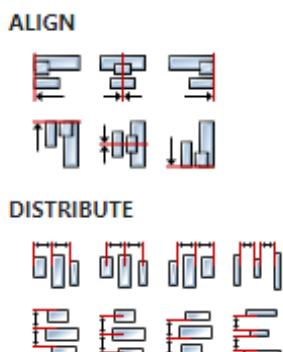
A68.2.9. Catch error Boolean

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

A68.2.10. 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.



A68.3. Inputs

A68.3.1. seqin SEQ / OPTIONAL

A standard sequence input.

A68.4. Outputs

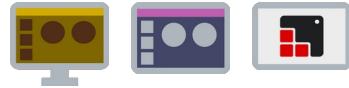
A68.4.1. seqout SEQ / OPTIONAL

A standard sequence output.

A68.4.2. result DATA(any) / MANDATORY

Output through which the sorted array is passed.

A69. Start



A69.1. 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.

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.

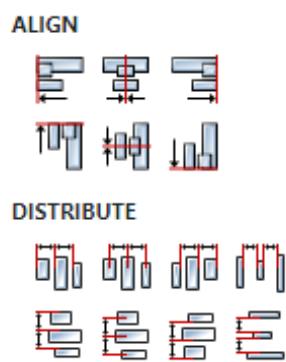
A69.2.4. Catch error *Boolean*

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.



A69.3. Inputs

A69.4. Outputs

A69.4.1. seqout SEQ / MANDATORY

Connect this output to the action you want to be executed first when the Flow starts.

A70. SwitchCase



A70.1. 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.

A70.2. Properties

Specific

A70.2.1. 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

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 out-

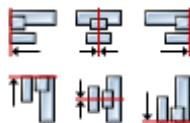
put. The data that will be passed through that output is the textual description of the error.

Position and size

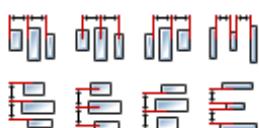
A70.2.6. Align and distribute Any

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.

ALIGN



DISTRIBUTE



A70.3. Inputs

A70.3.1. seqin SEQ / OPTIONAL

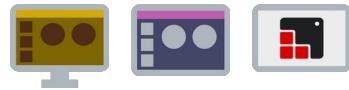
A standard sequence input.

A70.4. Outputs

A70.4.1. seqout SEQ / OPTIONAL

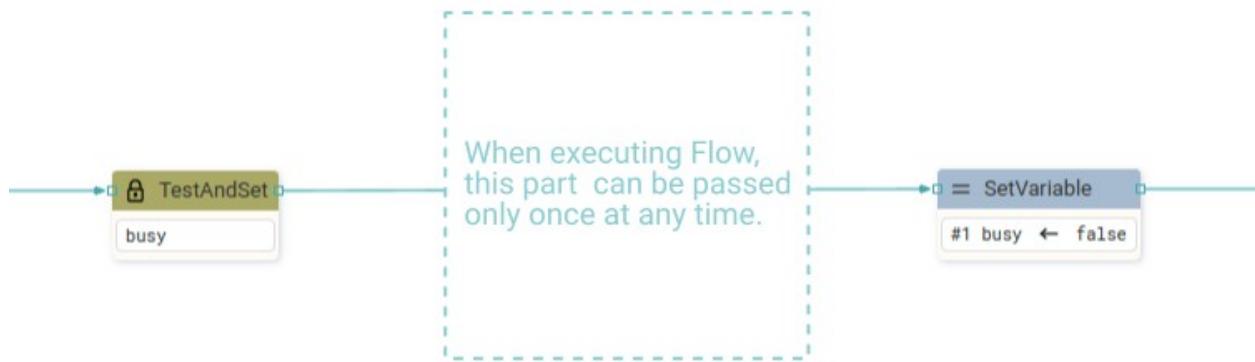
A standard sequence output.

A71. TestAndSet



A71.1. 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.



A71.2. Properties

Specific

A71.2.1. Variable `ASSIGNABLE EXPRESSION (boolean)`

The variable to be tested and set.

General

A71.2.2. 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

A71.2.3. 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.

A71.2.4. 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.

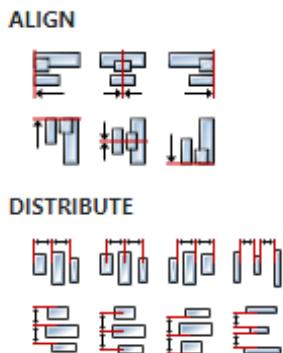
A71.2.5. Catch error Boolean

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

A71.2.6. 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.



A71.3. Inputs

A71.3.1. seqin SEQ / OPTIONAL

A standard sequential input.

A71.4. Outputs

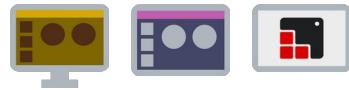
A71.4.1. seqout SEQ / OPTIONAL

Flow execution continues through this sequential output when the variable becomes `false`.

A71.5. 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.

A72. Watch



A72.1. 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.

A72.2. Properties

Specific

A72.2.1. Expression *EXPRESSION (any)*

Expression to be evaluated.

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 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.

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.

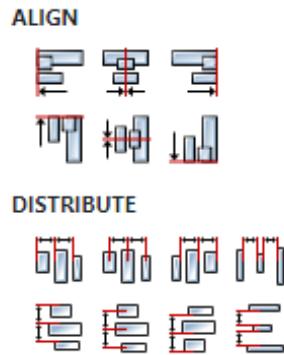
A72.2.5. Catch error *Boolean*

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.

A72.4.2. 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.

A73. WriteSetting



A73.1. 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.

A73.2. Properties

Specific

A73.2.1. Key *EXPRESSION (string)*

A string containing the name of the key to be added/updated.

A73.2.2. Value *EXPRESSION (any)*

The value of the key that will be created or updated.

General

A73.2.3. 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

A73.2.4. 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.

A73.2.5. 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.

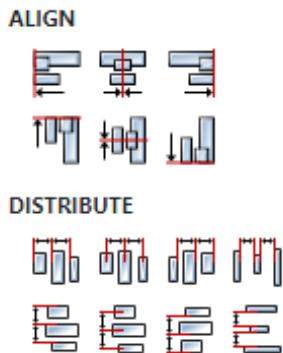
A73.2.6. Catch error *Boolean*

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

A73.2.7. 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.



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.

EEZ Studio

Widgets

Table of Contents

W1. Arc.....	W.5
W2. Bar.....	W.11
W3. BarGraph.....	W.17
W4. Bitmap (Dashboard).....	W.21
W5. Bitmap (EEZ-GUI).....	W.25
W6. Button (Dashboard).....	W.29
W7. Button (EEZ-GUI).....	W.33
W8. ButtonGroup.....	W.37
W9. Button (LVGL).....	W.41
W10. Calendar.....	W.47
W11. Chart.....	W.53
W12. Checkbox (Dashboard).....	W.59
W13. Checkbox (LVGL).....	W.63
W14. Colorwheel.....	W.69
W15. Container.....	W.75
W16. DisplayData.....	W.79
W17. Dropdown (Dashboard).....	W.83
W18. Dropdown (EEZ-GUI).....	W.87
W19. Dropdown (LVGL).....	W.91
W20. EEZChart.....	W.97
W21. Gauge (Dashboard).....	W.103
W22. Gauge (EEZ-GUI).....	W.107
W23. Grid.....	W.111
W24. Image.....	W.115
W25. Imgbutton.....	W.121
W26. Input (EEZ-GUI).....	W.127
W27. Keyboard.....	W.131
W28. Label.....	W.137
W29. LineChart (Dashboard).....	W.143
W30. LineChart (EEZ-GUI).....	W.147
W31. List.....	W.153
W32. Markdown.....	W.157
W33. Meter.....	W.161
W34. MultilineText.....	W.167
W35. Panel.....	W.171
W36. Progress (Dashboard).....	W.177
W37. Progress (EEZ-GUI).....	W.181
W38. QRCode (Dashboard).....	W.185
W39. QRCode (EEZ-GUI).....	W.189
W40. Rectangle (Dashboard).....	W.193
W41. Rectangle (EEZ-GUI).....	W.197
W42. Roller (EEZ-GUI).....	W.201
W43. Roller (LVGL).....	W.205

W44. ScrollBar.....	W.211
W45. Select.....	W.215
W46. Slider (Dashboard).....	W.219
W47. Slider (EEZ-GUI).....	W.223
W48. Slider (LVGL).....	W.227
W49. Spinner (Dashboard).....	W.233
W50. Spinner (LVGL).....	W.237
W51. Switch (Dashboard).....	W.243
W52. Switch (EEZ-GUI).....	W.247
W53. Switch (LVGL).....	W.251
W54. Terminal.....	W.257
W55. Textarea.....	W.261
W56. Text (Dashboard).....	W.267
W57. Text (EEZ-GUI).....	W.271
W58. TextInput.....	W.275
W59. ToggleButton.....	W.279
W60. UpDown.....	W.283

W1. Arc



W1.1. Description

The Arc consists of a background and a foreground arc. The foreground (indicator) can be touch-adjusted.

More info ([link](#))

W1.2. Properties

Specific

W1.2.1. Range min *Number*

The minimum value that can be selected by the `Value` property.

W1.2.2. Range max *Number*

The maximum value that can be selected by the `Value` property.

W1.2.3. 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.

W1.2.4. Value type *Enum*

Defines whether the `Value` will be given as a Literal or as an Expression.

W1.2.5. 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.

W1.2.6. 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.

W1.2.7. 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.

W1.2.8. Rotation *Number*

An offset to the 0 degree position.

General

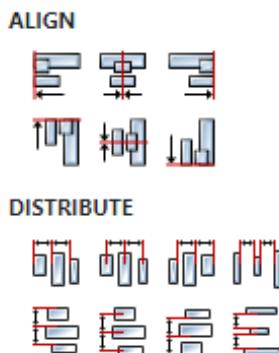
W1.2.9. Name *String*

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

W1.2.10. 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.



W1.2.11. Center widget Any

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



W1.2.12. 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 +, -, *, / operators in expressions. Brackets can also be used.

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

W1.2.13. 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.

W1.2.14. Top Number

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

W1.2.15. 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.

W1.2.16. Width Number

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

W1.2.17. 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.

W1.2.18. Height *Number*

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

W1.2.19. 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.

Style

W1.2.20. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W1.2.21. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W1.2.22. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W1.2.23. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W1.2.24. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W1.2.25. Click focusable *Boolean*

Add focused state to the object when clicked.

W1.2.26. Checkable *Boolean*

Toggle checked state when the object is clicked.

W1.2.27. Scrollable *Boolean*

Make the object scrollable.

W1.2.28. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W1.2.29. Scroll momentum Boolean

Make the object scroll further when "thrown".

W1.2.30. Scroll one Boolean

Allow scrolling only one snappable children.

W1.2.31. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W1.2.32. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W1.2.33. Scroll chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W1.2.34. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W1.2.35. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W1.2.36. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W1.2.37. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W1.2.38. Event bubble Boolean

Propagate the events to the parent too.

W1.2.39. Gesture bubble Boolean

Propagate the gestures to the parent.

W1.2.40. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W1.2.41. Ignore layout Boolean

Make the object positionable by the layouts.

W1.2.42. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W1.2.43. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States

W1.2.44. Checked EXPRESSION (boolean)

Toggled or checked state.

W1.2.45. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W1.2.46. Disabled *EXPRESSION (boolean)*

Disabled state

W1.2.47. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W1.2.48. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W1.2.49. Pressed *Boolean*

Being pressed.

Events

W1.2.50. 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

W1.2.51. 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.

W1.2.52. 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.

W1.2.53. 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 out-

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

W1.3. Examples

- *LVGL Widgets Demo*
- *Smart Home*

W2. Bar



W2.1. 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](#))

W2.2. Properties

Specific

W2.2.1. Min Number

The minimum value that `Value` and `Value start` can contain.

W2.2.2. Max Number

The maximum value that `Value` and `Value start` can contain.

W2.2.3. 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).

W2.2.4. Value EXPRESSION (integer)

The end value on the bar.

W2.2.5. Value type Enum

Select between `Literal` and `Expression`. If `Expression` is selected then `Value` can be evaluated from the expression.

W2.2.6. Value start EXPRESSION (integer)

The start value on the bar if `RANGE` mode is selected.

W2.2.7. Value start type Enum

Select between `Literal` and `Expression`. If `Expression` is selected then `Value start` can be evaluated from the expression.

General

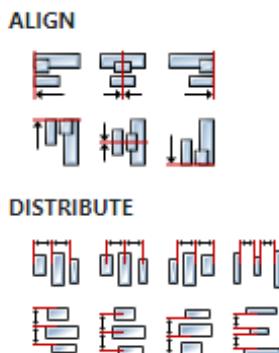
W2.2.8. Name String

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

W2.2.9. 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.



W2.2.10. Center widget Any

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



W2.2.11. 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 +, -, *, / operators in expressions. Brackets can also be used.

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

W2.2.12. 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.

W2.2.13. Top Number

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

W2.2.14. 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.

W2.2.15. Width Number

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

W2.2.16. 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.

W2.2.17. Height *Number*

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

W2.2.18. 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.

Style

W2.2.19. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W2.2.20. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W2.2.21. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W2.2.22. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W2.2.23. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W2.2.24. Click focusable *Boolean*

Add focused state to the object when clicked.

W2.2.25. Checkable *Boolean*

Toggle checked state when the object is clicked.

W2.2.26. Scrollable *Boolean*

Make the object scrollable.

W2.2.27. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W2.2.28. Scroll momentum Boolean

Make the object scroll further when "thrown".

W2.2.29. Scroll one Boolean

Allow scrolling only one snappable children.

W2.2.30. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W2.2.31. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W2.2.32. Scroll chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W2.2.33. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W2.2.34. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W2.2.35. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W2.2.36. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W2.2.37. Event bubble Boolean

Propagate the events to the parent too.

W2.2.38. Gesture bubble Boolean

Propagate the gestures to the parent.

W2.2.39. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W2.2.40. Ignore layout Boolean

Make the object positionable by the layouts.

W2.2.41. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W2.2.42. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States

W2.2.43. Checked EXPRESSION (boolean)

Toggled or checked state.

W2.2.44. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W2.2.45. Disabled *EXPRESSION (boolean)*

Disabled state

W2.2.46. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W2.2.47. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W2.2.48. Pressed *Boolean*

Being pressed.

Events

W2.2.49. 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

W2.2.50. 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.

W2.2.51. 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.

W2.2.52. 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 out-

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

W2.3. Examples

- *Dashboard Widgets Demo*

W3. BarGraph



W3.1. 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.

W3.2. Properties

Specific

W3.2.1. Data *EXPRESSION (any)*

This is the value within the range `[Min, Max]` for which the bar and text will be rendered.

W3.2.2. 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

W3.2.3. Display value *Boolean*

When checked, `Data` value will also be displayed as text.

W3.2.4. 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.

W3.2.5. 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.

W3.2.6. Min *EXPRESSION (any)*

The minimum value that `Data` can contain.

W3.2.7. Max *EXPRESSION (any)*

The maximum value that `Data` can contain.

W3.2.8. Refresh rate *EXPRESSION (any)*

Similar to the case of the `DisplayData` Widget, it defines the speed at which the text will be refreshed.

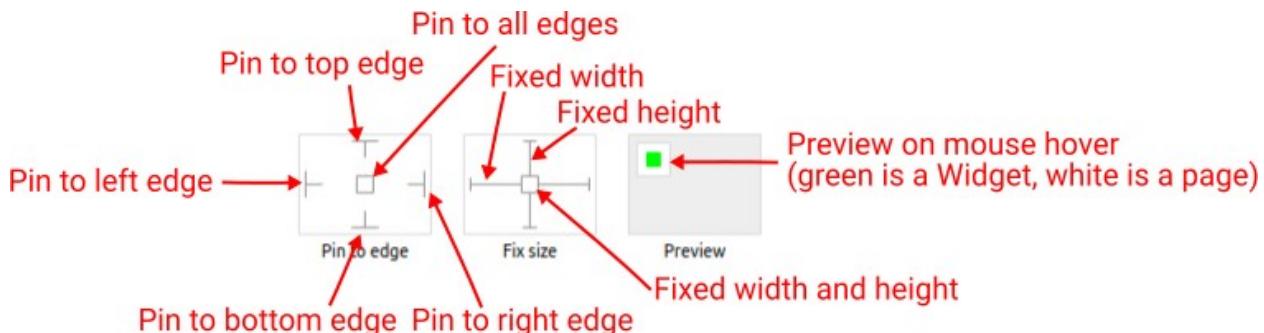
W3.2.9. Visible *EXPRESSION (boolean)*

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

W3.2.10. 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.

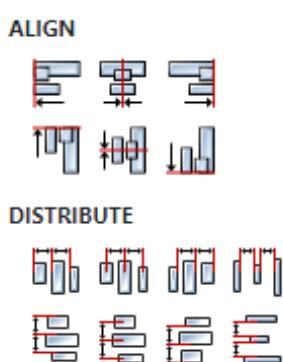
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W3.2.11. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W3.2.12. 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.

**W3.2.13. Center widget Any**

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



W3.2.14. 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`.

W3.2.15. Top Number

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

W3.2.16. Width Number

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

W3.2.17. Height Number

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

W3.2.18. Absolute position String

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

Style**W3.2.19. Default Object**

Style used when rendering of the Widget.

W3.2.20. Text Object

Style used to render the text inside the Widget.

W3.2.21. Threshold1 Object

Style used to render the `Threshold1` value.

W3.2.22. Threshold2 Object

Style used to render the `Threshold2` value.

Events**W3.2.23. 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**W3.2.24. 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.

W3.2.25. 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.

W3.2.26. 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

- `eez-gui-widgets-demo`

W4. Bitmap (Dashboard)



W4.1. 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.

W4.2. Properties

Specific

W4.2.1. 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).

W4.2.2. Bitmap *ObjectReference*

The name of the bitmap to be displayed.

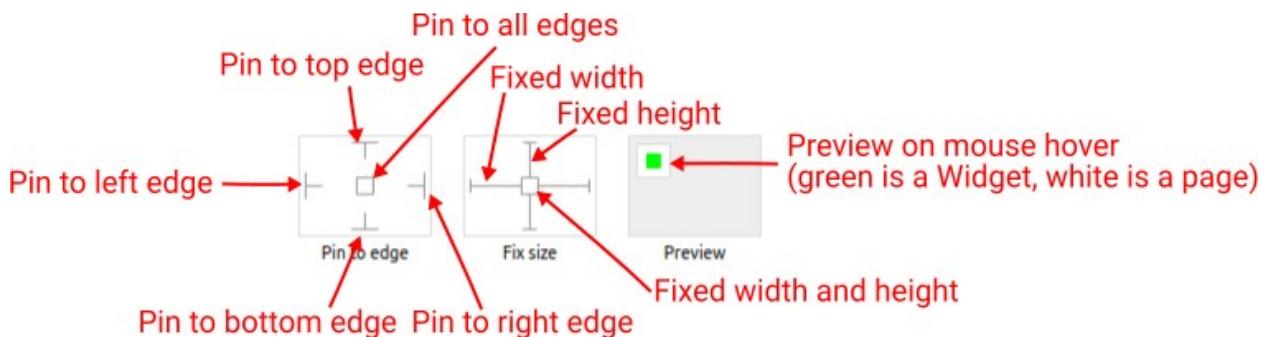
W4.2.3. Visible *EXPRESSION (boolean)*

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

W4.2.4. 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

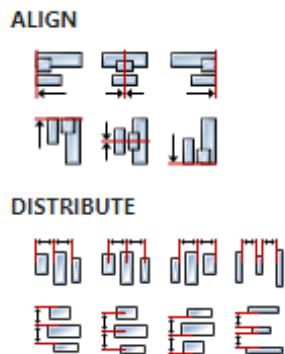
of the Widget will always be the same, in other words the Top position does not change the value. If *Pin to top edge* is not selected, then the Top position will scale proportionally as the page height scales.

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*.

W4.2.5. 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.



W4.2.6. Center widget Any

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



W4.2.7. 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`.

W4.2.8. Top Number

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

W4.2.9. Width Number

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

W4.2.10. Height Number

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

W4.2.11. Absolute position String

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

Style**W4.2.12. Default Object**

Style used when rendering the background of the Widget.

Events**W4.2.13. 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**W4.2.14. 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.

W4.2.15. 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.

W4.2.16. 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.

W4.3. Examples

- Dashboard Widgets Demo
- Screen Capture

W5. Bitmap (EEZ-GUI)



W5.1. Description

This Widget displays a bitmap.

W5.2. Properties

Specific

W5.2.1. 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.

W5.2.2. Bitmap *ObjectReference*

The name of the bitmap to be displayed.

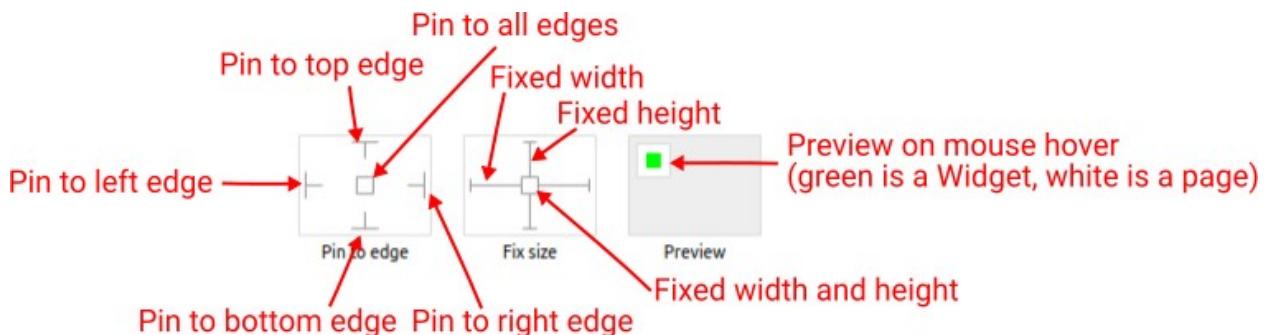
W5.2.3. Visible *EXPRESSION (boolean)*

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.

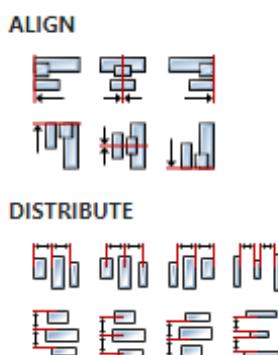
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W5.2.5. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W5.2.6. 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.

**W5.2.7. Center widget** Any

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

**W5.2.8. 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 +, -, *, / operators in expressions. Brackets can also be used.

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

W5.2.9. Top Number

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

W5.2.10. Width Number

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

W5.2.11. Height Number

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

W5.2.12. Absolute position String

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

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. 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.

W5.2.16. 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.

W5.2.17. 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.

W5.3. Examples

- `eez-gui-widgets-demo`

W6. Button (Dashboard)



W6.1. 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.

W6.2. Properties

Specific

W6.2.1. Label *EXPRESSION (any)*

The text that will be displayed inside the button.

W6.2.2. Enabled *EXPRESSION (any)*

If it is true, then the button is enabled, otherwise it will be eisabled.

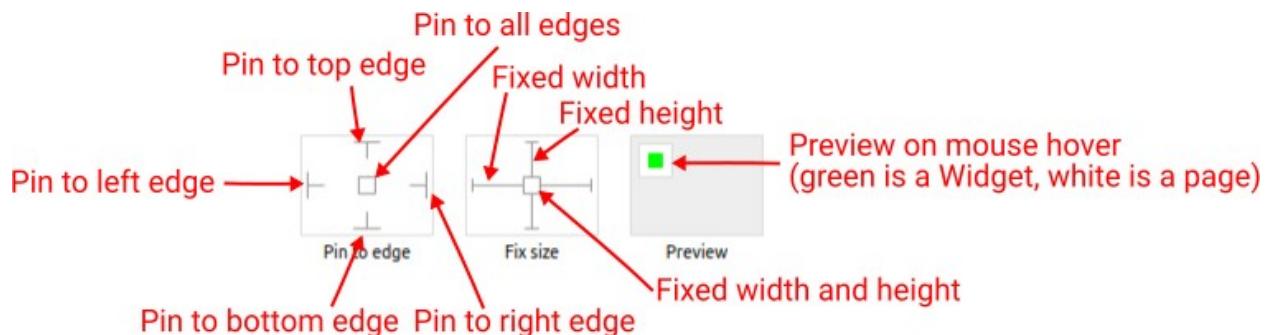
W6.2.3. Visible *EXPRESSION (boolean)*

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:



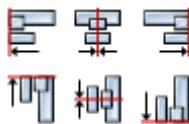
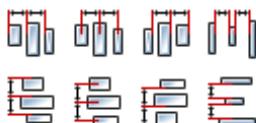
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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.

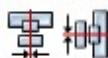
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W6.2.5. 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****W6.2.6. Center widget** *Any*

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

**W6.2.7. 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`.

W6.2.8. Top *Number*

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

W6.2.9. Width *Number*

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

W6.2.10. Height *Number*

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

W6.2.11. Absolute position *String*

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

Style**W6.2.12. Default** *Object*

Style to be used for rendering if the Widget is enabled.

W6.2.13. Disabled *Object*

Style to be used for rendering if the Widget is disabled.

Events

W6.2.14. 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

W6.2.15. 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.

W6.2.16. 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.

W6.2.17. 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.

W6.3. Examples

- [eez-gui-widgets-demo](#)

W7. Button (EEZ-GUI)



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 eisabled.

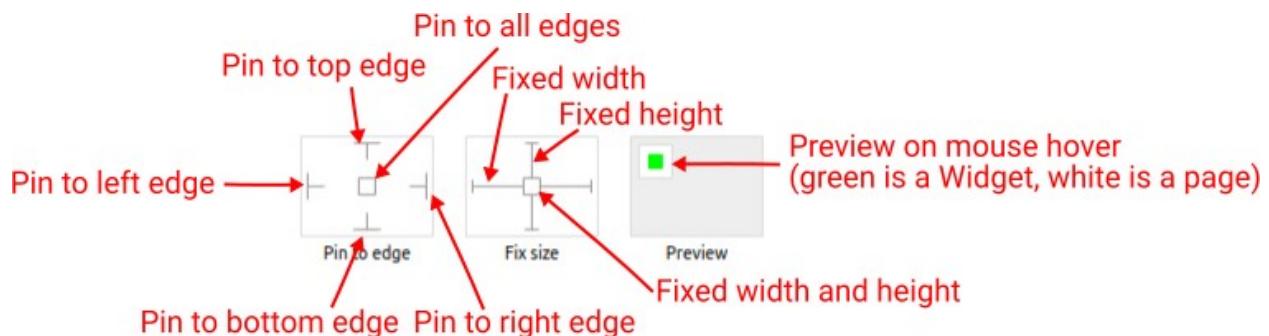
W7.2.3. Visible *EXPRESSION (boolean)*

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*.

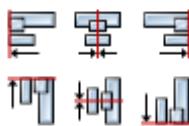
W7.2.5. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

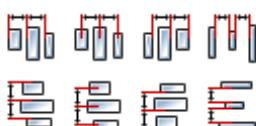
W7.2.6. Align and distribute Any

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

**W7.2.7. Center widget** Any

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

**W7.2.8. 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 +, -, *, / operators in expressions. Brackets can also be used.

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

W7.2.9. Top Number

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

W7.2.10. Width Number

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

W7.2.11. Height Number

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

W7.2.12. Absolute position String

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

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. 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.

W7.2.17. 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.

W7.2.18. 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.

W7.3. Examples

- [eez-gui-widgets-demo](#)

W8. ButtonGroup



W8.1. 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.

W8.2. Properties

Specific

W8.2.1. 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.

W8.2.2. 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.

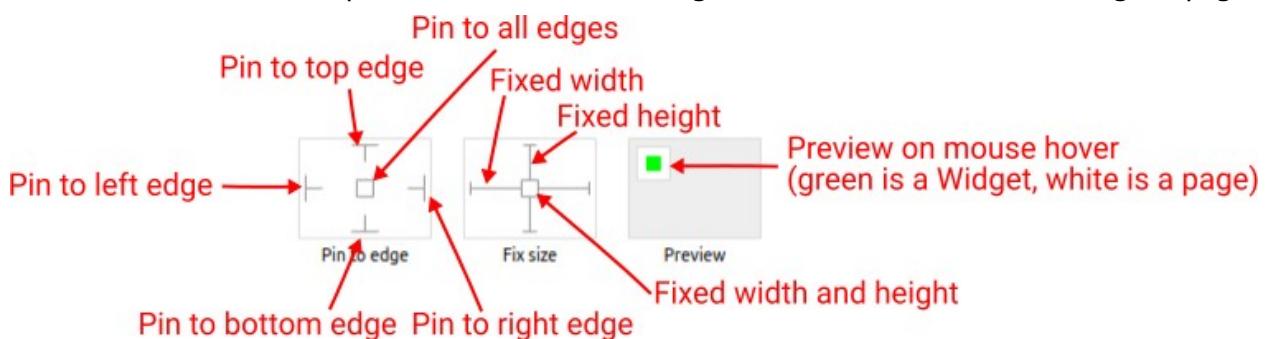
W8.2.3. Visible *EXPRESSION (boolean)*

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*.

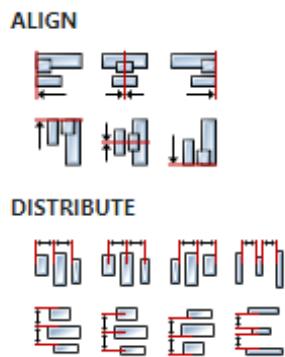
and *Fix width*.

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.

Style

W8.2.13. Default Object

Style is used to render a button that is not selected.

W8.2.14. Selected Object

Style used to render the selected button.

Events

W8.2.15. 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

W8.2.16. 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.

W8.2.17. 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.

W8.2.18. 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.

W8.3. Examples

- [eez-gui-widgets-demo](#)

W9. Button (LVGL)



W9.1. 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](#))

W9.2. Properties

General

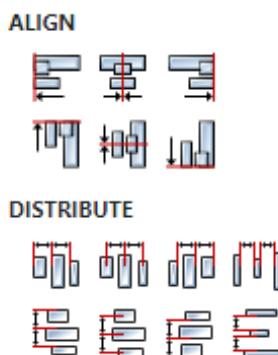
W9.2.1. Name *String*

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

W9.2.2. 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.



W9.2.3. Center widget *Any*

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



W9.2.4. 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`.

W9.2.5. 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.

W9.2.6. Top *Number*

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

W9.2.7. 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.

W9.2.8. Width *Number*

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

W9.2.9. 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.

W9.2.10. Height *Number*

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

W9.2.11. 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.

Style**W9.2.12. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W9.2.13. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W9.2.14. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W9.2.15. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W9.2.16. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W9.2.17. Click focusable Boolean

Add focused state to the object when clicked.

W9.2.18. Checkable Boolean

Toggle checked state when the object is clicked.

W9.2.19. Scrollable Boolean

Make the object scrollable.

W9.2.20. Scroll elastic Boolean

Allow scrolling inside but with slower speed.

W9.2.21. Scroll momentum Boolean

Make the object scroll further when "thrown".

W9.2.22. Scroll one Boolean

Allow scrolling only one snappable children.

W9.2.23. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W9.2.24. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W9.2.25. Scroll chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W9.2.26. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W9.2.27. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W9.2.28. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W9.2.29. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W9.2.30. Event bubble Boolean

Propagate the events to the parent too.

W9.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W9.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W9.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W9.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W9.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States

W9.2.36. Checked EXPRESSION (boolean)

Toggled or checked state.

W9.2.37. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W9.2.38. Disabled EXPRESSION (boolean)

Disabled state

W9.2.39. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W9.2.40. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W9.2.41. Pressed Boolean

Being pressed.

Events

W9.2.42. 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

W9.2.43. 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.

W9.2.44. 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.

W9.2.45. 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

- *LVGL Widgets Demo*

W10. Calendar



W10.1. Description

This Widget displays a calendar.
More info ([link](#))

W10.2. Properties

Specific

W10.2.1. Year *Number*

Initially selected year.

W10.2.2. Month *Number*

Initially selected month.

W10.2.3. Day *Number*

Initially selected day.

General

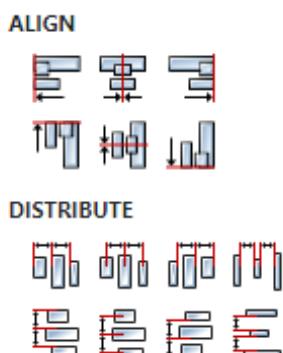
W10.2.4. Name *String*

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

W10.2.5. 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.



W10.2.6. Center widget *Any*

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



W10.2.7. 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`.

W10.2.8. 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.

W10.2.9. Top *Number*

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

W10.2.10. 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.

W10.2.11. Width *Number*

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

W10.2.12. 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.

W10.2.13. Height *Number*

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

W10.2.14. 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.

Style**W10.2.15. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W10.2.16. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W10.2.17. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W10.2.18. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W10.2.19. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W10.2.20. Click focusable *Boolean*

Add focused state to the object when clicked.

W10.2.21. Checkable *Boolean*

Toggle checked state when the object is clicked.

W10.2.22. Scrollable *Boolean*

Make the object scrollable.

W10.2.23. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W10.2.24. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W10.2.25. Scroll one *Boolean*

Allow scrolling only one snappable children.

W10.2.26. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W10.2.27. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W10.2.28. Scroll chain *Boolean*

Allow propagating both the horizontal and the vertical scroll to a parent.

W10.2.29. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W10.2.30. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W10.2.31. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W10.2.32. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W10.2.33. Event bubble Boolean

Propagate the events to the parent too.

W10.2.34. Gesture bubble Boolean

Propagate the gestures to the parent.

W10.2.35. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W10.2.36. Ignore layout Boolean

Make the object positionable by the layouts.

W10.2.37. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W10.2.38. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W10.2.39. Checked EXPRESSION (boolean)**

Toggled or checked state.

W10.2.40. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W10.2.41. Disabled EXPRESSION (boolean)

Disabled state

W10.2.42. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W10.2.43. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W10.2.44. Pressed Boolean

Being pressed.

Events**W10.2.45. 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

W10.2.46. 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.

W10.2.47. 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.

W10.2.48. 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.

W10.3. Examples

- *LVGL Widgets Demo*

W11. Chart



W11.1. Description

Charts are a basic object to visualize data points.
More info ([link](#))

W11.2. Properties

General

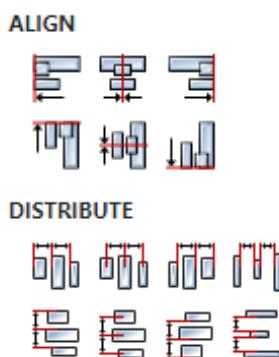
W11.2.1. Name *String*

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

W11.2.2. 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.



W11.2.3. Center widget *Any*

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



W11.2.4. 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`.

W11.2.5. 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.

W11.2.6. Top Number

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

W11.2.7. 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.

W11.2.8. Width Number

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

W11.2.9. 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.

W11.2.10. Height Number

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

W11.2.11. 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.

Style**W11.2.12. Use style ObjectReference**

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W11.2.13. Hidden EXPRESSION (boolean)**

Make the object hidden.

W11.2.14. Hidden flag type Enum

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W11.2.15. Clickable EXPRESSION (boolean)

Make the object clickable by input devices.

W11.2.16. Clickable flag type Enum

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or

not.

W11.2.17. Click focusable Boolean

Add focused state to the object when clicked.

W11.2.18. Checkable Boolean

Toggle checked state when the object is clicked.

W11.2.19. Scrollable Boolean

Make the object scrollable.

W11.2.20. Scroll elastic Boolean

Allow scrolling inside but with slower speed.

W11.2.21. Scroll momentum Boolean

Make the object scroll further when "thrown".

W11.2.22. Scroll one Boolean

Allow scrolling only one snappable children.

W11.2.23. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W11.2.24. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W11.2.25. Scroll chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W11.2.26. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W11.2.27. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W11.2.28. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W11.2.29. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W11.2.30. Event bubble Boolean

Propagate the events to the parent too.

W11.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W11.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W11.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W11.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W11.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W11.2.36. Checked EXPRESSION (boolean)**

Toggled or checked state.

W11.2.37. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W11.2.38. Disabled EXPRESSION (boolean)

Disabled state

W11.2.39. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W11.2.40. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W11.2.41. Pressed Boolean

Being pressed.

Events**W11.2.42. 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**W11.2.43. 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.

W11.2.44. 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.

W11.2.45. 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.

W11.3. Examples

- *Dashboard Widgets Demo*

W12. Checkbox (Dashboard)



W12.1. Description

Checkbox Widget is used when we want a turn ON or turn OFF option.

W12.2. Properties

Specific

W12.2.1. Value *EXPRESSION (any)*

Boolean variable in which `true` is stored when the switch is ON and `false` when the switch is OFF.

W12.2.2. Label *EXPRESSION (string)*

Label displayed next to the checkbox.

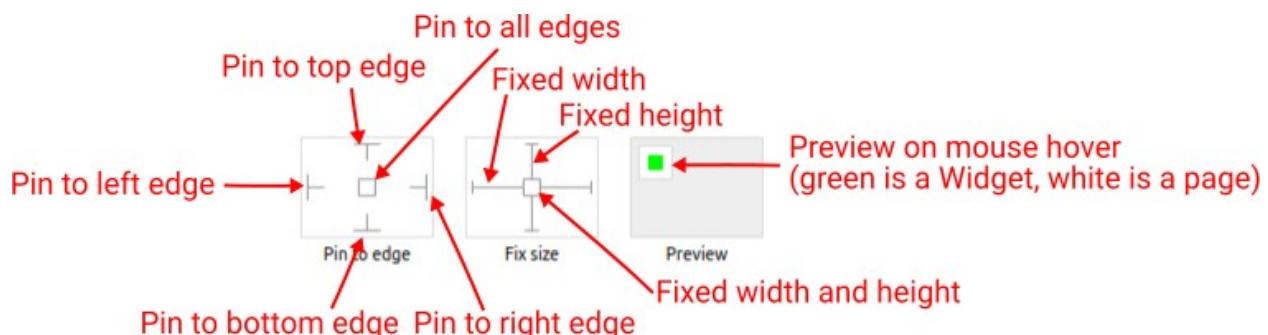
W12.2.3. Visible *EXPRESSION (boolean)*

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

W12.2.4. 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:



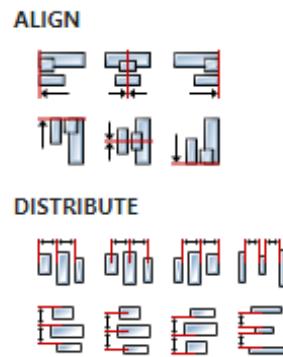
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

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. Top Number

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

W12.2.9. Width Number

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

W12.2.10. Height Number

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

W12.2.11. Absolute position String

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

Style**W12.2.12. Default Object**

Style used when rendering of the Widget.

Events**W12.2.13. 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

W12.2.14. 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.

W12.2.15. 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.

W12.2.16. 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.

W12.3. Examples

- *Dashboard Widgets Demo*

W13. Checkbox (LVGL)



W13.1. Description

Checkbox Widget is used when we want a turn ON or turn OFF option.
More info ([link](#))

W13.2. Properties

Specific

W13.2.1. Text *String*

Label displayed next to the checkbox.

General

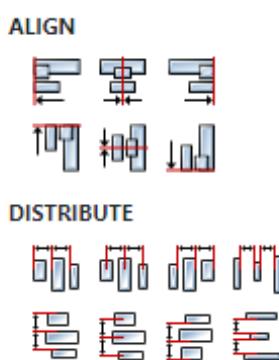
W13.2.2. Name *String*

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

W13.2.3. 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.



W13.2.4. Center widget *Any*

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



W13.2.5. 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.`

W13.2.6. 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.

W13.2.7. Top *Number*

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

W13.2.8. 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.

W13.2.9. Width *Number*

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

W13.2.10. 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.

W13.2.11. Height *Number*

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

W13.2.12. 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.

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 chain *Boolean*

Allow propagating both the horizontal and the vertical scroll to a parent.

W13.2.27. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W13.2.28. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W13.2.29. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W13.2.30. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W13.2.31. Event bubble *Boolean*

Propagate the events to the parent too.

W13.2.32. Gesture bubble Boolean

Propagate the gestures to the parent.

W13.2.33. Adv hit test Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W13.2.34. Ignore layout Boolean

Make the object positionable by the layouts.

W13.2.35. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W13.2.36. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W13.2.37. Checked EXPRESSION (boolean)**

Toggled or checked state.

W13.2.38. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W13.2.39. Disabled EXPRESSION (boolean)

Disabled state

W13.2.40. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W13.2.41. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W13.2.42. Pressed Boolean

Being pressed.

Events**W13.2.43. 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

W13.2.44. 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.

W13.2.45. 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.

W13.2.46. 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.

W13.3. Examples

- Dashboard Widgets Demo

W14. Colorwheel



W14.1. Description

This widget allows the user to select a color.
More info ([link](#))

W14.2. Properties

Specific

W14.2.1. Mode *Enum*

Select which part of the color will be changed with the Widget:

- HUE
- SATURATION
- VALUE

W14.2.2. Fixed mode *Boolean*

The color mode can be fixed (so as to not change with long press) using this item.

General

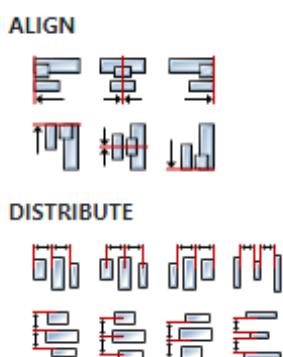
W14.2.3. Name *String*

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

W14.2.4. 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.



W14.2.5. Center widget *Any*

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



W14.2.6. 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`.

W14.2.7. 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.

W14.2.8. Top Number

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

W14.2.9. 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.

W14.2.10. Width Number

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

W14.2.11. 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.

W14.2.12. Height Number

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

W14.2.13. 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.

Style

W14.2.14. Use style ObjectReference

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W14.2.15. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W14.2.16. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W14.2.17. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W14.2.18. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W14.2.19. Click focusable *Boolean*

Add focused state to the object when clicked.

W14.2.20. Checkable *Boolean*

Toggle checked state when the object is clicked.

W14.2.21. Scrollable *Boolean*

Make the object scrollable.

W14.2.22. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W14.2.23. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W14.2.24. Scroll one *Boolean*

Allow scrolling only one snappable children.

W14.2.25. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W14.2.26. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W14.2.27. Scroll chain *Boolean*

Allow propagating both the horizontal and the vertical scroll to a parent.

W14.2.28. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W14.2.29. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W14.2.30. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W14.2.31. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W14.2.32. Event bubble Boolean

Propagate the events to the parent too.

W14.2.33. Gesture bubble Boolean

Propagate the gestures to the parent.

W14.2.34. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W14.2.35. Ignore layout Boolean

Make the object positionable by the layouts.

W14.2.36. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W14.2.37. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

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.

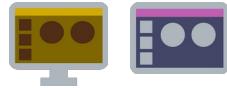
W14.2.47. Catch error Boolean

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. Container



W15.1. 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.

W15.2. Properties

Specific

W15.2.1. 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.

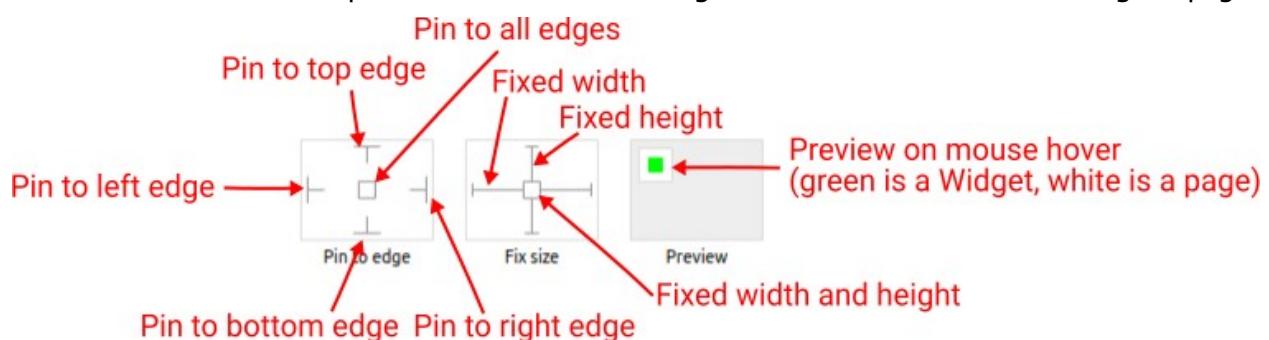
W15.2.2. Visible *EXPRESSION (boolean)*

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

W15.2.3. 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.

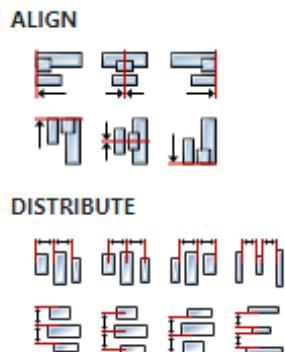
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W15.2.4. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W15.2.5. 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.



W15.2.6. Center widget Any

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



W15.2.7. 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.`

W15.2.8. Top Number

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

W15.2.9. Width Number

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

W15.2.10. Height Number

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

W15.2.11. Absolute position *String*

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

General**W15.2.12. Name** *String*

Optional name to display in the *Widgets Structure* panel in the editor. If not set then `Container` is displayed.

Style**W15.2.13. Default** *Object*

Style used when rendering the background of the Widget.

Events**W15.2.14. 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**W15.2.15. 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.

W15.2.16. 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.

W15.2.17. 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.

W15.3. Examples

- *eez-gui-widgets-demo*

W16. DisplayData



W16.1. Description

Similar to the `Text` Widget, but it has some more options that are set via the `Display option` and `Refresh rate` properties.

W16.2. Properties

Specific

W16.2.1. Data `EXPRESSION (any)`

An expression that, when calculated, is converted into a string and displayed inside the widget.

W16.2.2. 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

W16.2.3. 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.

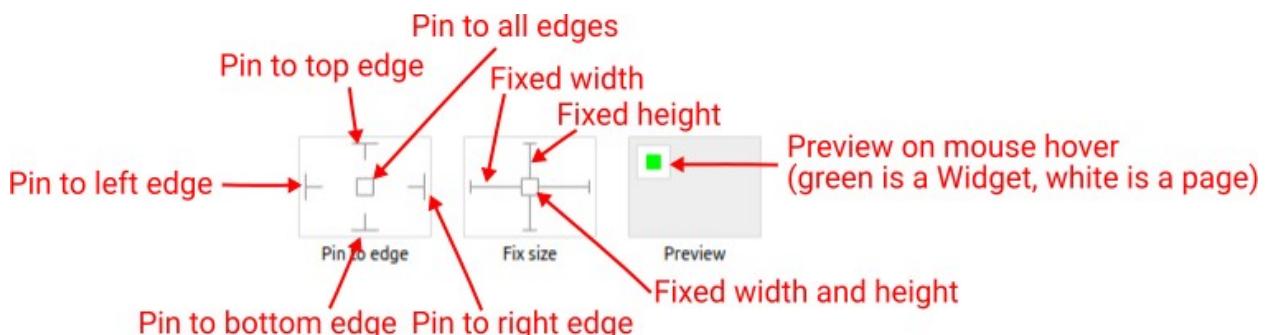
W16.2.4. Visible `EXPRESSION (boolean)`

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

W16.2.5. 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.

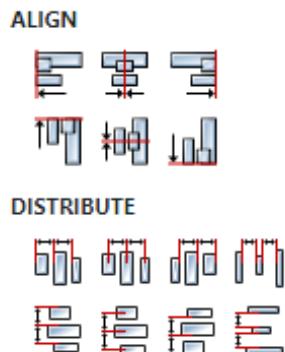
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W16.2.6. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W16.2.7. 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.



W16.2.8. Center widget Any

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



W16.2.9. 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.`

W16.2.10. Top Number

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

W16.2.11. Width Number

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

W16.2.12. Height Number

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

W16.2.13. Absolute position *String*

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

Style**W16.2.14. Default** *Object*

Style used when rendering of the Widget.

W16.2.15. Focused *Object*

Style to be used for rendering if the Widget is in focus.

Events**W16.2.16. 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**W16.2.17. 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.

W16.2.18. 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.

W16.2.19. 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.

W16.3. Examples

- `eez-gui-widgets-demo`

W17. Dropdown (Dashboard)



W17.1. Description

We can use this widget when we need to select one option from the list.

W17.2. Properties

Specific

W17.2.1. Data *EXPRESSION (integer)*

The variable in which the zero-based index of the selected option will be stored.

W17.2.2. Options *EXPRESSION (any)*

List of options.

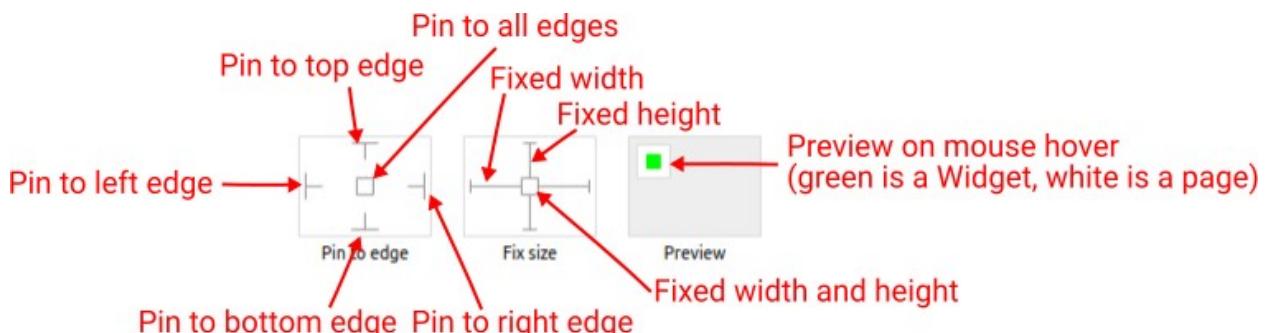
W17.2.3. Visible *EXPRESSION (boolean)*

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

W17.2.4. 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:



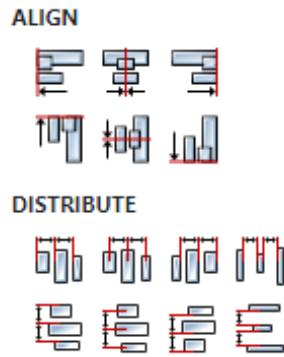
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

W17.2.5. 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.

**W17.2.6. Center widget Any**

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

**W17.2.7. 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.

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. Width Number

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

W17.2.10. Height Number

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

W17.2.11. Absolute position String

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

Style**W17.2.12. Default Object**

Style used when rendering of the Widget.

Events**W17.2.13. 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

W17.2.14. 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.

W17.2.15. 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.

W17.2.16. 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.

W17.3. Examples

- *Dashboard Widgets Demo*

W18. Dropdown (EEZ-GUI)



W18.1. Description

We can use this widget when we need to select one option from the list.

W18.2. Properties

Specific

W18.2.1. Data *EXPRESSION (integer)*

The variable in which the zero-based index of the selected option will be stored.

W18.2.2. Options *EXPRESSION (any)*

List of options.

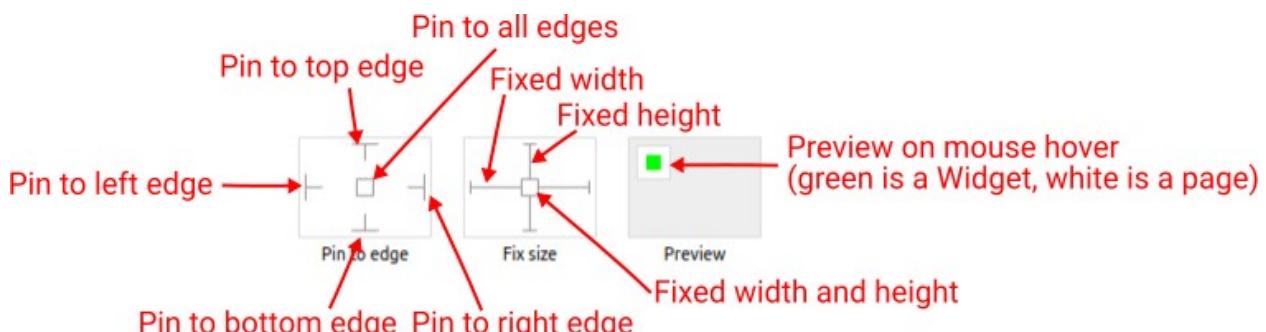
W18.2.3. Visible *EXPRESSION (boolean)*

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

W18.2.4. 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.

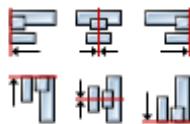
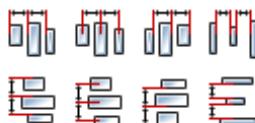
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W18.2.5. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

W18.2.6. 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****W18.2.7. Center widget Any**

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

**W18.2.8. 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.

W18.2.9. Top Number

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

W18.2.10. Width Number

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

W18.2.11. Height Number

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

W18.2.12. Absolute position String

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

Style**W18.2.13. Default Object**

Style used when rendering of the Widget.

Events

W18.2.14. 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

W18.2.15. 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.

W18.2.16. 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.

W18.2.17. 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.

W18.3. Examples

- `eez-gui-widgets-demo`

W19. Dropdown (LVGL)



W19.1. Description

The drop-down list allows the user to select one value from a list.
More info ([link](#))

W19.2. Properties

Specific

W19.2.1. Options *EXPRESSION (array:string)*

List of options.

W19.2.2. 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.

W19.2.3. Selected *EXPRESSION (integer)*

The zero-based index of the selected option.

W19.2.4. 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

W19.2.5. Name *String*

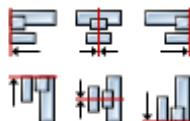
Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

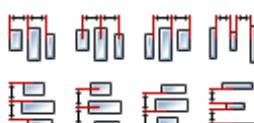
W19.2.6. Align and distribute *Any*

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



W19.2.7. Center widget *Any*

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

**W19.2.8. 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`.

W19.2.9. 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.

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. 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.

W19.2.12. Width *Number*

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

W19.2.13. 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.

W19.2.14. Height *Number*

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

W19.2.15. 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.

Style

W19.2.16. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W19.2.17. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W19.2.18. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W19.2.19. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W19.2.20. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W19.2.21. Click focusable *Boolean*

Add focused state to the object when clicked.

W19.2.22. Checkable *Boolean*

Toggle checked state when the object is clicked.

W19.2.23. Scrollable *Boolean*

Make the object scrollable.

W19.2.24. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W19.2.25. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W19.2.26. Scroll one *Boolean*

Allow scrolling only one snappable children.

W19.2.27. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W19.2.28. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W19.2.29. Scroll chain *Boolean*

Allow propagating both the horizontal and the vertical scroll to a parent.

W19.2.30. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W19.2.31. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W19.2.32. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W19.2.33. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W19.2.34. Event bubble Boolean

Propagate the events to the parent too.

W19.2.35. Gesture bubble Boolean

Propagate the gestures to the parent.

W19.2.36. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W19.2.37. Ignore layout Boolean

Make the object positionable by the layouts.

W19.2.38. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W19.2.39. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W19.2.40. Checked EXPRESSION (boolean)**

Toggled or checked state.

W19.2.41. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W19.2.42. Disabled EXPRESSION (boolean)

Disabled state

W19.2.43. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W19.2.44. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W19.2.45. Pressed Boolean

Being pressed.

Events

W19.2.46. 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

W19.2.47. 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.

W19.2.48. 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.

W19.2.49. 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.

W19.3. Examples

- *Dashboard Widgets Demo*

W20. EEZChart



W20.1. Description

Displays a line chart using the same Widget as in the Instrument *History* panel.

W20.2. Properties

Specific

W20.2.1. 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.

W20.2.2. 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`.

W20.2.3. 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`.

W20.2.4. 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`.

W20.2.5. 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`.

W20.2.6. Color *EXPRESSION (string)*

The color of the line in the chart.

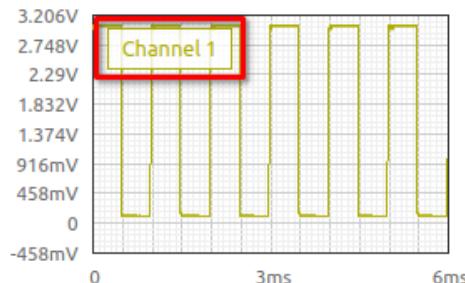
This property is only used when the `Chart mode` is `Single chart`.

W20.2.7. 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`.

W20.2.8. 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`.

W20.2.9. 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`.

W20.2.10. 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.

W20.2.11. 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`.

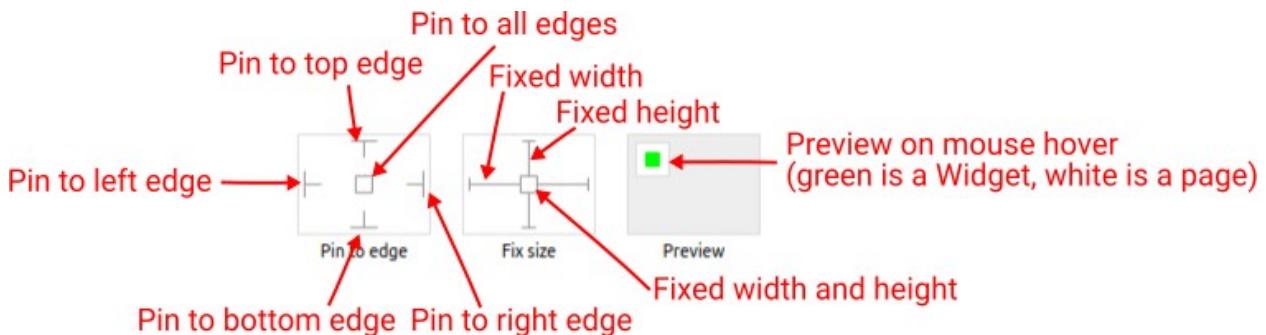
W20.2.12. Visible *EXPRESSION (boolean)*

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

W20.2.13. 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:



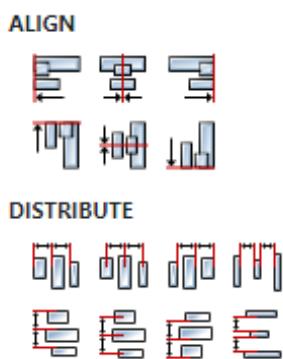
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

W20.2.14. 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.



W20.2.15. Center widget Any

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



W20.2.16. 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.`

W20.2.17. Top Number

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

W20.2.18. Width Number

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

W20.2.19. Height Number

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

W20.2.20. Absolute position String

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

Style

W20.2.21. Default Object

Style used when rendering of the Widget.

Events

W20.2.22. 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

W20.2.23. 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.

W20.2.24. 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.

W20.2.25. 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.

W20.3. Examples

- *Line Chart*
- *EEZ Chart*
- *Rigol Waveform Data*

W21. Gauge (Dashboard)



W21.1. Description

Displays the value selected through the `Data` property inside the gauge chart.

W21.2. Properties

Specific

W21.2.1. Data *EXPRESSION (any)*

The value within the `[Min, Max]` range that is displayed in the Widget.

W21.2.2. Title *String*

The name displayed above the gauge chart.

W21.2.3. Min range *EXPRESSION (float)*

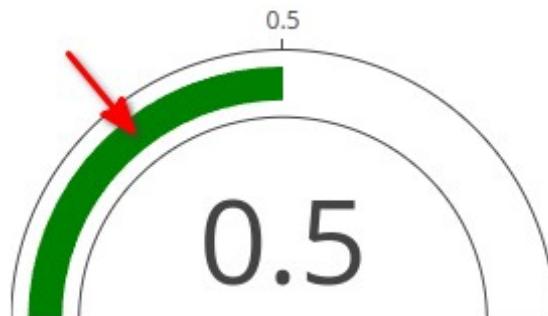
The minimum value that `Data` can contain.

W21.2.4. Max range *EXPRESSION (float)*

The maximum value that `Data` can contain.

W21.2.5. Color *Color*

The color of the arc bar inside the chart.



W21.2.6. Margin *Object*

Manually selected margin values between the Widget borders and the chart itself within the Widget.

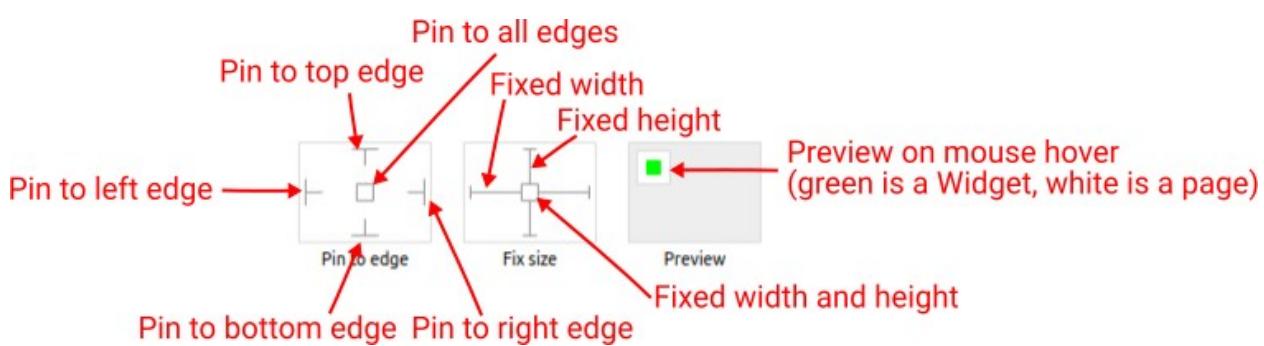
W21.2.7. Visible *EXPRESSION (boolean)*

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

W21.2.8. 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:



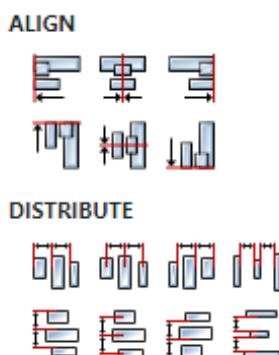
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

W21.2.9. 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.



W21.2.10. Center widget Any

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



W21.2.11. 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.`

W21.2.12. Top *Number*

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

W21.2.13. Width *Number*

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

W21.2.14. Height *Number*

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

W21.2.15. Absolute position *String*

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

Style**W21.2.16. Default** *Object*

Style used when rendering the background of the Widget.

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

- *Dashboard Widgets Demo*

W22. Gauge (EEZ-GUI)



W22.1. 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.

W22.2. Properties

Specific

W22.2.1. Data *EXPRESSION (any)*

The value within the `[Min, Max]` range that is displayed in the Widget.

W22.2.2. Min *EXPRESSION (any)*

The minimum value that `Data` can contain.

W22.2.3. Max *EXPRESSION (any)*

The maximum value that `Data` can contain.

W22.2.4. 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).

W22.2.5. Unit *EXPRESSION (any)*

The unit that will be displayed to the right of the `Data` value.

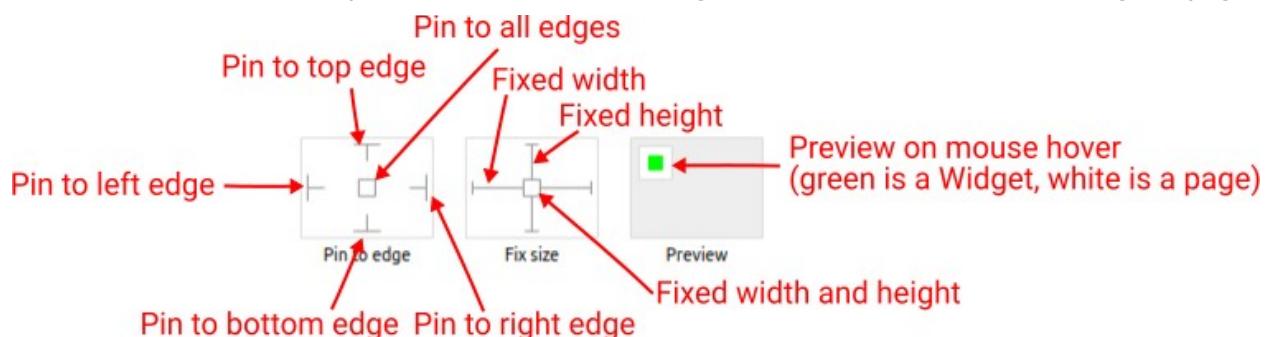
W22.2.6. Visible *EXPRESSION (boolean)*

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

W22.2.7. 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.

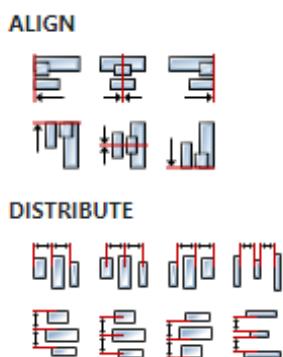
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W22.2.8. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W22.2.9. 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.



W22.2.10. Center widget Any

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



W22.2.11. 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`.

W22.2.12. Top Number

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

W22.2.13. Width Number

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

W22.2.14. Height Number

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

W22.2.15. Absolute position *String*

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

Style**W22.2.16. Default** *Object*

Style used when rendering the background of the Widget.

W22.2.17. Bar *Object*

Style used to render the bar inside the Widget.

W22.2.18. Value *Object*

Style used to render the value specified through `Data`.

W22.2.19. Ticks *Object*

The style used to render the ticks on the scale.

W22.2.20. Threshold *Object*

Style used to render the threshold line.

Events**W22.2.21. 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**W22.2.22. 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.

W22.2.23. 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.

W22.2.24. 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

- *eez-gui-widgets-demo*

W23. Grid



W23.1. 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`).

W23.2. Properties

Specific

W23.2.1. 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.

W23.2.2. 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.

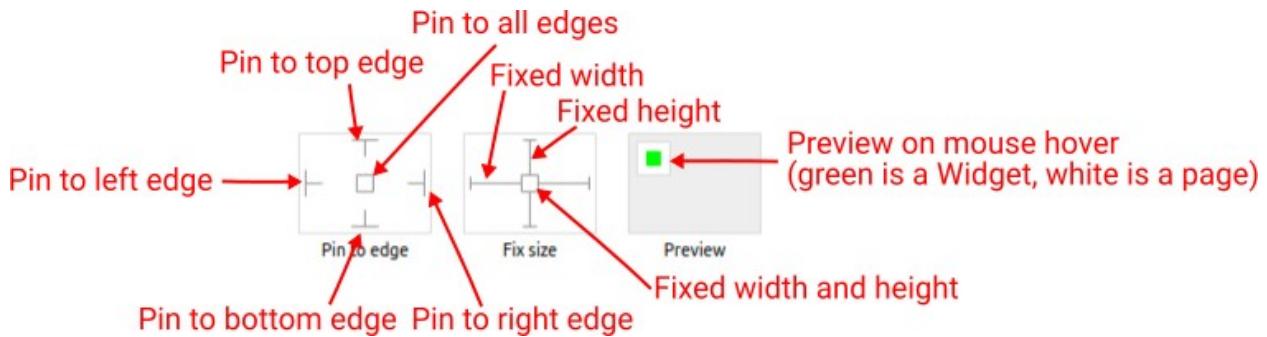
W23.2.3. Visible *EXPRESSION (boolean)*

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.

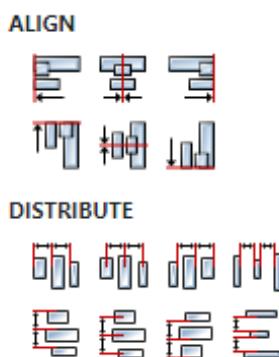
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

Style**W23.2.13. Default** *Object*

Style used when rendering the background of the Widget.

Events**W23.2.14. 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**W23.2.15. 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.

W23.2.16. 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.

W23.2.17. 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.

W23.3. Examples

- `eez-gui-widgets-demo`
- `Tetris`

W24. Image



W24.1. Description

This Widget displays an image.
More info ([link](#))

W24.2. Properties

Specific

W24.2.1. Image *ObjectReference*

The name of the bitmap to be displayed.

W24.2.2. Pivot X *Number*

X position of the center of rotation. If left blank, the center of rotation is in the middle of the Widget.

W24.2.3. Pivot Y *Number*

Y position of the center of rotation. If left blank, the center of rotation is in the middle of the Widget.

W24.2.4. 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.

W24.2.5. 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.

General

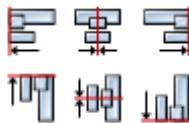
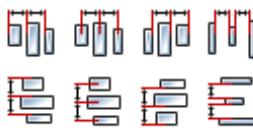
W24.2.6. Name *String*

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

W24.2.7. 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****W24.2.8. Center widget Any**

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

**W24.2.9. 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.

W24.2.10. 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.

W24.2.11. Top Number

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

W24.2.12. 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.

W24.2.13. Width Number

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

W24.2.14. 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.

W24.2.15. Height *Number*

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

W24.2.16. 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.

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 chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W24.2.31. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W24.2.32. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W24.2.33. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W24.2.34. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W24.2.35. Event bubble Boolean

Propagate the events to the parent too.

W24.2.36. Gesture bubble Boolean

Propagate the gestures to the parent.

W24.2.37. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W24.2.38. Ignore layout Boolean

Make the object positionable by the layouts.

W24.2.39. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W24.2.40. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W24.2.41. Checked EXPRESSION (boolean)**

Toggled or checked state.

W24.2.42. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W24.2.43. Disabled *EXPRESSION (boolean)*

Disabled state

W24.2.44. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W24.2.45. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W24.2.46. Pressed *Boolean*

Being pressed.

Events**W24.2.47. 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**W24.2.48. 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.

W24.2.49. 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.

W24.2.50. 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.

W24.3. Examples

- *LVGL Widgets Demo*

W25. Imgbutton



W25.1. 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](#))

W25.2. Properties

Specific

W25.2.1. Released image *ObjectReference*

The image for the `Released` state.

W25.2.2. Pressed image *ObjectReference*

The image for the `Pressed` state.

W25.2.3. Disabled image *ObjectReference*

The image for the `Disabled` state.

W25.2.4. Checked released image *ObjectReference*

The image when the widget is in the both `Checked` and `Released` state.

W25.2.5. Checked pressed image *ObjectReference*

The image when the widget is in the both `Checked` and `Pressed` state.

W25.2.6. Checked disabled image *ObjectReference*

The image when the Widget is in the both `Checked` and `Disabled` state.

General

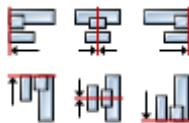
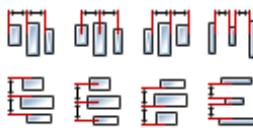
W25.2.7. Name *String*

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.

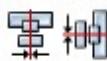
Position and size

W25.2.8. 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****W25.2.9. Center widget Any**

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

**W25.2.10. 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.

W25.2.11. 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.

W25.2.12. Top Number

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

W25.2.13. 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.

W25.2.14. Width Number

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

W25.2.15. 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.

W25.2.16. Height *Number*

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

W25.2.17. 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.

Style**W25.2.18. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W25.2.19. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W25.2.20. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W25.2.21. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W25.2.22. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W25.2.23. Click focusable *Boolean*

Add focused state to the object when clicked.

W25.2.24. Checkable *Boolean*

Toggle checked state when the object is clicked.

W25.2.25. Scrollable *Boolean*

Make the object scrollable.

W25.2.26. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W25.2.27. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W25.2.28. Scroll one *Boolean*

Allow scrolling only one snappable children.

W25.2.29. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W25.2.30. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W25.2.31. Scroll chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W25.2.32. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W25.2.33. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W25.2.34. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W25.2.35. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W25.2.36. Event bubble Boolean

Propagate the events to the parent too.

W25.2.37. Gesture bubble Boolean

Propagate the gestures to the parent.

W25.2.38. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W25.2.39. Ignore layout Boolean

Make the object positionable by the layouts.

W25.2.40. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W25.2.41. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States

W25.2.42. Checked EXPRESSION (boolean)

Toggled or checked state.

W25.2.43. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W25.2.44. Disabled *EXPRESSION (boolean)*

Disabled state

W25.2.45. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W25.2.46. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W25.2.47. Pressed *Boolean*

Being pressed.

Events**W25.2.48. 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**W25.2.49. 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.

W25.2.50. 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.

W25.2.51. 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.

W25.3. Examples

- *LVGL Widgets Demo*



W26.1. 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.

W26.2. Properties

Specific

W26.2.1. Data *EXPRESSION (any)*

The variable in which the entered number or text will be stored.

W26.2.2. Input type *Enum*

Choose whether `Number` or `Text` is entered.

W26.2.3. 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.

W26.2.4. 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.

W26.2.5. 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.

W26.2.6. 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.

W26.2.7. 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.

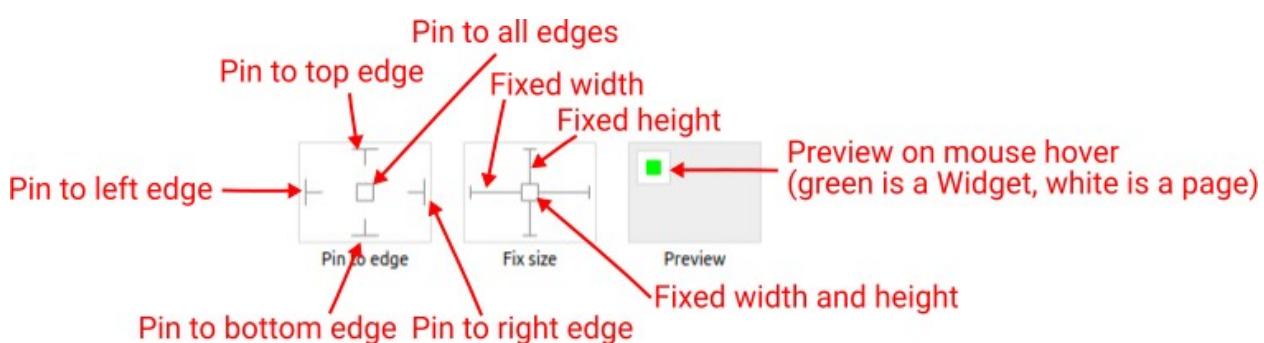
W26.2.8. Visible *EXPRESSION (boolean)*

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

W26.2.9. 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.

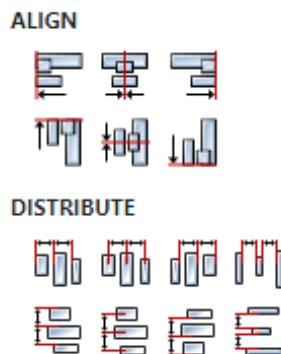
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W26.2.10. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W26.2.11. 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.



W26.2.12. Center widget Any

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



W26.2.13. 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.`

W26.2.14. Top Number

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

W26.2.15. Width Number

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

W26.2.16. Height Number

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

W26.2.17. Absolute position String

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

Style

W26.2.18. Default Object

Style used when rendering of the Widget.

Events

W26.2.19. 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

W26.2.20. 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.

W26.2.21. 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.

W26.2.22. 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.

W26.3. Examples

- *eez-gui-widgets-demo*
- *stm32f469i-disco-eez-flow-demo*

W27. Keyboard



W27.1. Description

The virtual (on screen) keyboard to write texts into a Text area.
More info ([link](#))

W27.2. Properties

Specific

W27.2.1. Textarea *Enum*

The name of `Textarea` Widget attached to this Widget.

W27.2.2. 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

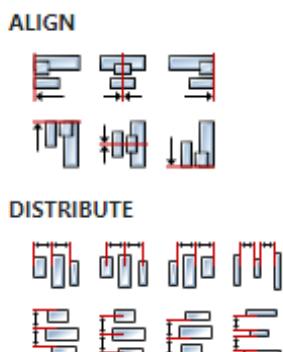
W27.2.3. Name *String*

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

W27.2.4. 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.



W27.2.5. Center widget *Any*

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



W27.2.6. 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`.

W27.2.7. 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.

W27.2.8. Top Number

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

W27.2.9. 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.

W27.2.10. Width Number

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

W27.2.11. 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.

W27.2.12. Height Number

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

W27.2.13. 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.

Style

W27.2.14. Use style ObjectReference

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W27.2.15. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W27.2.16. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W27.2.17. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W27.2.18. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W27.2.19. Click focusable *Boolean*

Add focused state to the object when clicked.

W27.2.20. Checkable *Boolean*

Toggle checked state when the object is clicked.

W27.2.21. Scrollable *Boolean*

Make the object scrollable.

W27.2.22. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W27.2.23. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W27.2.24. Scroll one *Boolean*

Allow scrolling only one snappable children.

W27.2.25. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W27.2.26. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W27.2.27. Scroll chain *Boolean*

Allow propagating both the horizontal and the vertical scroll to a parent.

W27.2.28. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W27.2.29. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W27.2.30. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W27.2.31. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W27.2.32. Event bubble Boolean

Propagate the events to the parent too.

W27.2.33. Gesture bubble Boolean

Propagate the gestures to the parent.

W27.2.34. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W27.2.35. Ignore layout Boolean

Make the object positionable by the layouts.

W27.2.36. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W27.2.37. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W27.2.38. Checked EXPRESSION (boolean)**

Toggled or checked state.

W27.2.39. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W27.2.40. Disabled EXPRESSION (boolean)

Disabled state

W27.2.41. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W27.2.42. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W27.2.43. Pressed Boolean

Being pressed.

Events**W27.2.44. 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

W27.2.45. 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.

W27.2.46. 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.

W27.2.47. 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.

W27.3. Examples

- *LVGL Widgets Demo*

W28. Label



W28.1. Description

A Widget used to display text.
More info ([link](#))

W28.2. Properties

Specific

W28.2.1. Text *EXPRESSION (string)*

Text to be displayed.

W28.2.2. Text type *Enum*

Here we can choose whether the `Text` property will be calculated from the Expression.

W28.2.3. 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.

W28.2.4. 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

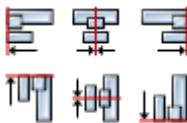
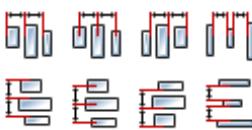
W28.2.5. Name *String*

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.

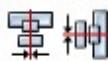
Position and size

W28.2.6. 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****W28.2.7. Center widget Any**

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

**W28.2.8. 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.

W28.2.9. 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.

W28.2.10. Top Number

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

W28.2.11. 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.

W28.2.12. Width Number

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

W28.2.13. 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.

W28.2.14. Height *Number*

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

W28.2.15. 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.

Style**W28.2.16. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W28.2.17. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W28.2.18. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W28.2.19. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W28.2.20. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W28.2.21. Click focusable *Boolean*

Add focused state to the object when clicked.

W28.2.22. Checkable *Boolean*

Toggle checked state when the object is clicked.

W28.2.23. Scrollable *Boolean*

Make the object scrollable.

W28.2.24. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W28.2.25. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W28.2.26. Scroll one *Boolean*

Allow scrolling only one snappable children.

W28.2.27. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W28.2.28. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W28.2.29. Scroll chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W28.2.30. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W28.2.31. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W28.2.32. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W28.2.33. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W28.2.34. Event bubble Boolean

Propagate the events to the parent too.

W28.2.35. Gesture bubble Boolean

Propagate the gestures to the parent.

W28.2.36. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W28.2.37. Ignore layout Boolean

Make the object positionable by the layouts.

W28.2.38. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W28.2.39. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W28.2.40. Checked EXPRESSION (boolean)**

Toggled or checked state.

W28.2.41. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W28.2.42. Disabled *EXPRESSION (boolean)*

Disabled state

W28.2.43. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W28.2.44. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W28.2.45. Pressed *Boolean*

Being pressed.

Events**W28.2.46. 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**W28.2.47. 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.

W28.2.48. 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.

W28.2.49. 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.

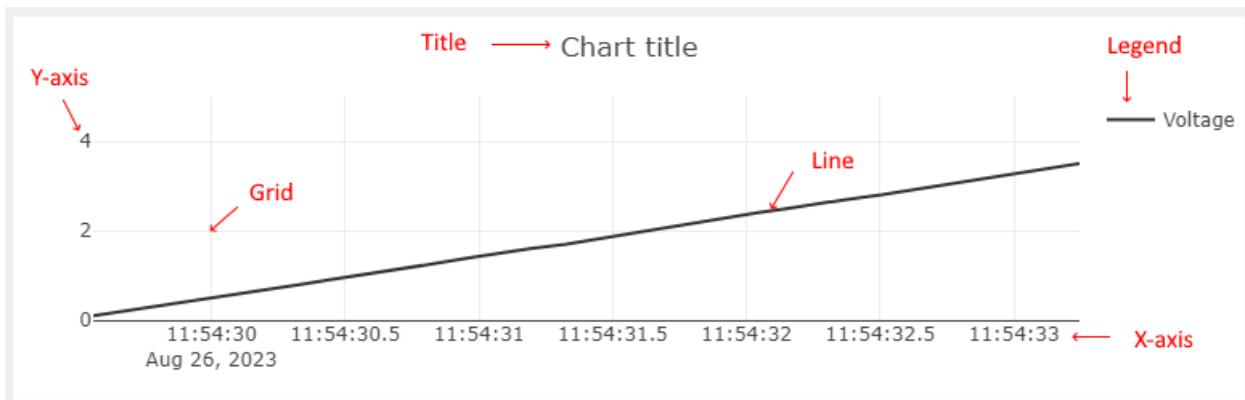
W29. LineChart (Dashboard)



W29.1. 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.

W29.2. Properties

Specific

W29.2.1. 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.

W29.2.2. 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.

W29.2.3. Title *String*

Name of the chart.

W29.2.4. Show legend *Boolean*

It should be set if we want to render the legend.

W29.2.5. 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.

W29.2.6. 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.

W29.2.7. 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.

W29.2.8. Max points Number

The maximum number of points that will be displayed.

W29.2.9. 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).

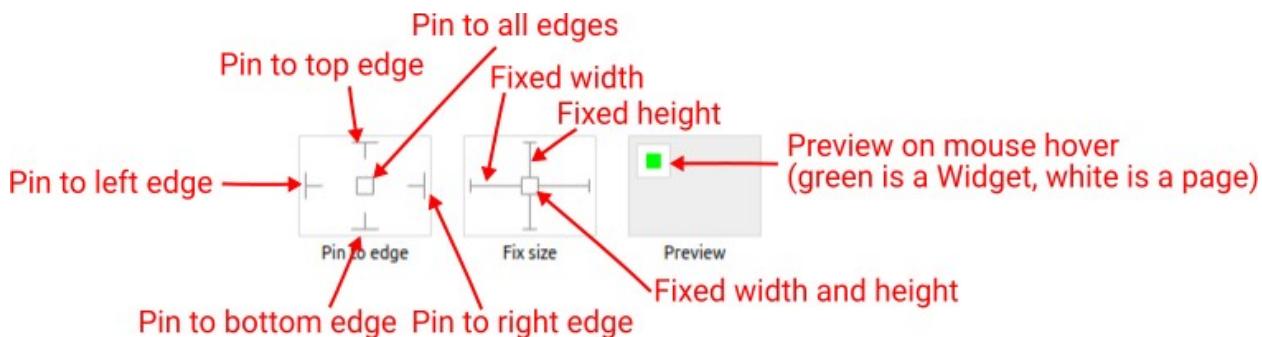
W29.2.10. Visible EXPRESSION (boolean)

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

W29.2.11. 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:



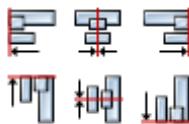
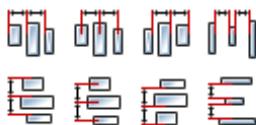
With the **Pin to edge** option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the **Scale to fit** option is selected. E.g. if we selected **Pin to top edge** then the distance between the top edge of the page and 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.

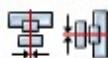
Note: If **Pin to left edge** and **Pin to right edge** are selected, then the **Fix width** option will be disabled, and conversely if **Fix width** is selected, then both **Pin to left edge** and **Pin to right edge** cannot be selected, because both cannot be satisfied. The same applies to **Pin to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

W29.2.12. 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.

ALIGN**DISTRIBUTE****W29.2.13. Center widget** *Any*

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

**W29.2.14. 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`.

W29.2.15. Top *Number*

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

W29.2.16. Width *Number*

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

W29.2.17. Height *Number*

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

W29.2.18. Absolute position *String*

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

Style**W29.2.19. Default** *Object*

Style used when rendering of the Widget.

Events

W29.2.20. 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**W29.2.21. 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.

W29.2.22. 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.

W29.2.23. 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.

W29.3. Inputs**W29.3.1. 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.

W29.3.2. 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.

W29.4. Examples

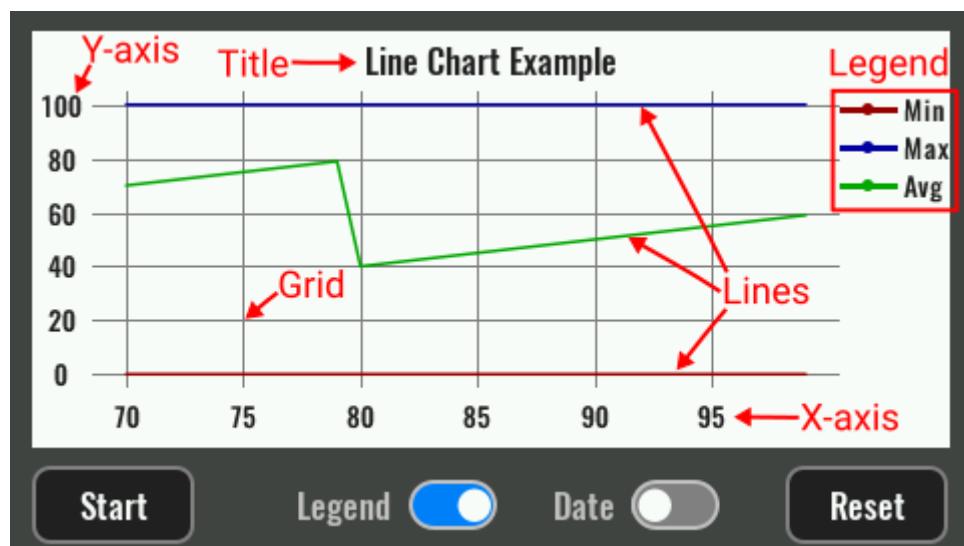
- *Dashboard Widgets Demo*



W30.1. 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.

W30.2. Properties

Specific

W30.2.1. 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.

W30.2.2. 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.

W30.2.3. Show title *EXPRESSION (boolean)*

It should be set if we want to render the title.

W30.2.4. Show legend *EXPRESSION (boolean)*

It should be set if we want to render the legend.

W30.2.5. Show X axis *EXPRESSION (boolean)*

It should be set if we want to render the X-axis.

W30.2.6. Show Y axis *EXPRESSION (boolean)*

It should be set if we want to render the Y-axis.

W30.2.7. Show grid *EXPRESSION (boolean)*

It should be set if we want to render the grid.

W30.2.8. Title *EXPRESSION (string)*

Name of the chart.

W30.2.9. 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.

W30.2.10. 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.

W30.2.11. 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.

W30.2.12. Max points *Number*

The maximum number of points that will be displayed.

W30.2.13. 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).

W30.2.14. Marker *EXPRESSION (float)*

At this position, a vertical line will be displayed inside the chart using `Marker` style.

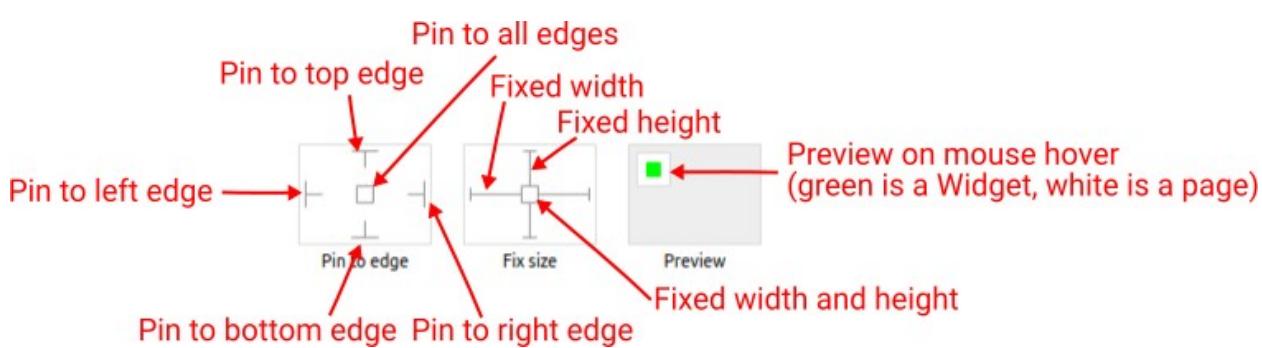
W30.2.15. Visible *EXPRESSION (boolean)*

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

W30.2.16. 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.

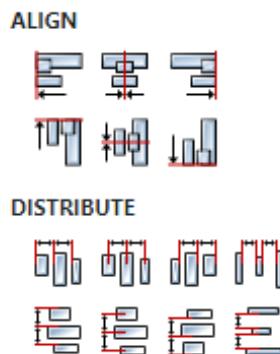
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W30.2.17. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W30.2.18. 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.



W30.2.19. Center widget Any

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



W30.2.20. 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.`

W30.2.21. Top Number

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

W30.2.22. Width Number

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

W30.2.23. Height Number

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

W30.2.24. Absolute position String

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

Style

W30.2.25. Default Object

Style used when rendering of the Widget.

W30.2.26. Title Object

Style used to render the title.

W30.2.27. Legend Object

Style used to render the legend.

W30.2.28. X axis Object

Style used to render the X-axis.

W30.2.29. Y axis Object

Style used to render the Y-axis.

W30.2.30. Marker Object

Style used to render the marker.

Events

W30.2.31. 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

W30.2.32. 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.

W30.2.33. 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.

W30.2.34. 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. Inputs

W30.3.1. 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.

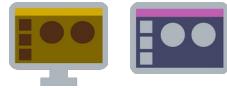
W30.3.2. value DATA(any) / MANDATORY

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.

W30.4. Examples

- *Line Chart*

W31. List



W31.1. 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`).

W31.2. Properties

Specific

W31.2.1. 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.

W31.2.2. List type *Enum*

Defines vertical or horizontal orientation.

W31.2.3. Gap *Number*

The distance in pixels between two grid elements.

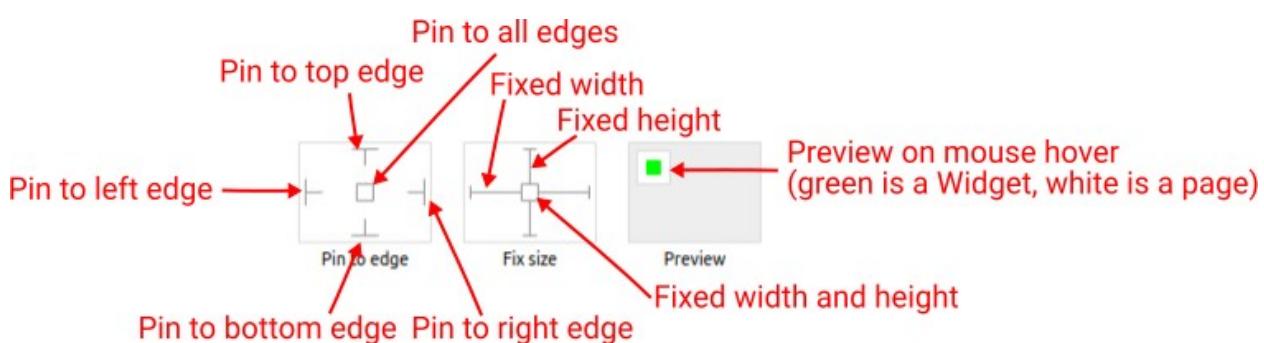
W31.2.4. Visible *EXPRESSION (boolean)*

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

W31.2.5. 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.

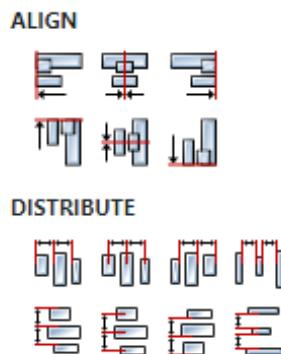
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W31.2.6. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W31.2.7. 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.



W31.2.8. Center widget Any

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



W31.2.9. 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.`

W31.2.10. Top Number

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

W31.2.11. Width Number

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

W31.2.12. Height Number

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

W31.2.13. Absolute position String

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

Style

W31.2.14. Default Object

Style used when rendering the background of the Widget.

Events

W31.2.15. 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

W31.2.16. 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.

W31.2.17. 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.

W31.2.18. 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.

W31.3. Examples

- *eez-gui-widgets-demo*
- *CSV*
- *JSON*
- *MQTT*
- *Simple HTTP*
- *Charts*
- *Regexp String*
- *Multi-Language*

W32. Markdown



W32.1. Description

Widget for displaying Markdown text.

W32.2. Properties

Specific

W32.2.1. Text *MultilineText*

Markdown text to be displayed.

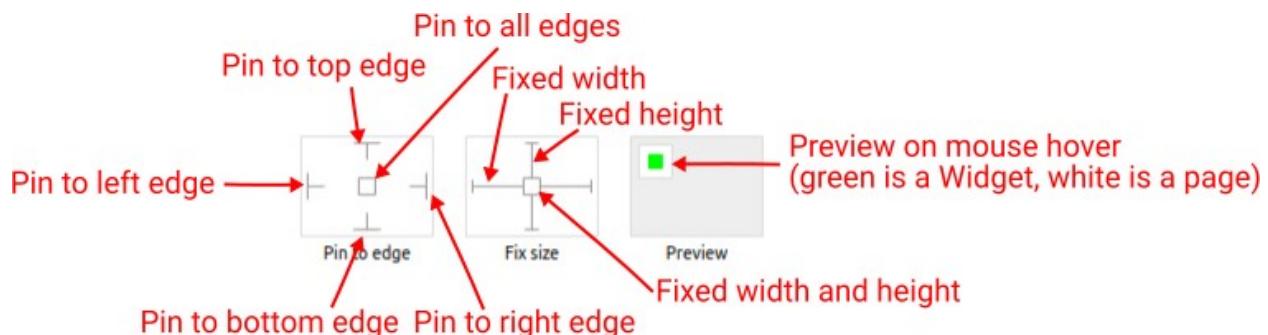
W32.2.2. Visible *EXPRESSION (boolean)*

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

W32.2.3. 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:



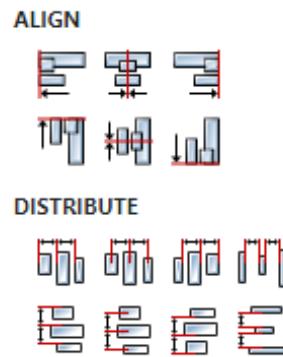
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

W32.2.4. 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.

**W32.2.5. Center widget Any**

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

**W32.2.6. 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.

W32.2.7. Top Number

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

W32.2.8. Width Number

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

W32.2.9. Height Number

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

W32.2.10. Absolute position String

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

Style**W32.2.11. Default Object**

Style used when rendering the background of the Widget.

Events**W32.2.12. 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

W32.2.13. 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.

W32.2.14. 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.

W32.2.15. 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.

W32.3. Examples

- *Dashboard Widgets Demo*

W33. Meter



W33.1. Description

The `Meter` Widget can visualize data in very flexible ways. It can show arcs, needles, ticks lines and labels.

More info ([link](#))

W33.2. Properties

Specific

W33.2.1. 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

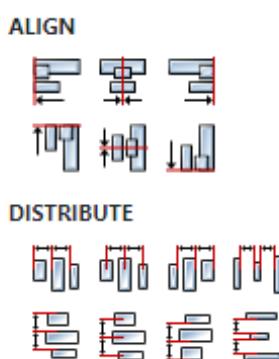
W33.2.2. Name *String*

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

W33.2.3. 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.



W33.2.4. Center widget *Any*

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



W33.2.5. 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`.

W33.2.6. 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.

W33.2.7. Top *Number*

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

W33.2.8. 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.

W33.2.9. Width *Number*

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

W33.2.10. 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.

W33.2.11. Height *Number*

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

W33.2.12. 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.

Style**W33.2.13. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W33.2.14. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W33.2.15. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W33.2.16. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W33.2.17. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W33.2.18. Click focusable *Boolean*

Add focused state to the object when clicked.

W33.2.19. Checkable *Boolean*

Toggle checked state when the object is clicked.

W33.2.20. Scrollable *Boolean*

Make the object scrollable.

W33.2.21. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W33.2.22. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W33.2.23. Scroll one *Boolean*

Allow scrolling only one snappable children.

W33.2.24. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W33.2.25. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W33.2.26. Scroll chain *Boolean*

Allow propagating both the horizontal and the vertical scroll to a parent.

W33.2.27. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W33.2.28. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W33.2.29. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W33.2.30. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W33.2.31. Event bubble Boolean

Propagate the events to the parent too.

W33.2.32. Gesture bubble Boolean

Propagate the gestures to the parent.

W33.2.33. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W33.2.34. Ignore layout Boolean

Make the object positionable by the layouts.

W33.2.35. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W33.2.36. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W33.2.37. Checked EXPRESSION (boolean)**

Toggled or checked state.

W33.2.38. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W33.2.39. Disabled EXPRESSION (boolean)

Disabled state

W33.2.40. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W33.2.41. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W33.2.42. Pressed Boolean

Being pressed.

Events**W33.2.43. 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

W33.2.44. 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.

W33.2.45. 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.

W33.2.46. 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

- *Meter*
- *Dashboard Widgets Demo*

W34. MultilineText



W34.1. Description

A widget used to display multiple lines text.

W34.2. Properties

Specific

W34.2.1. 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.

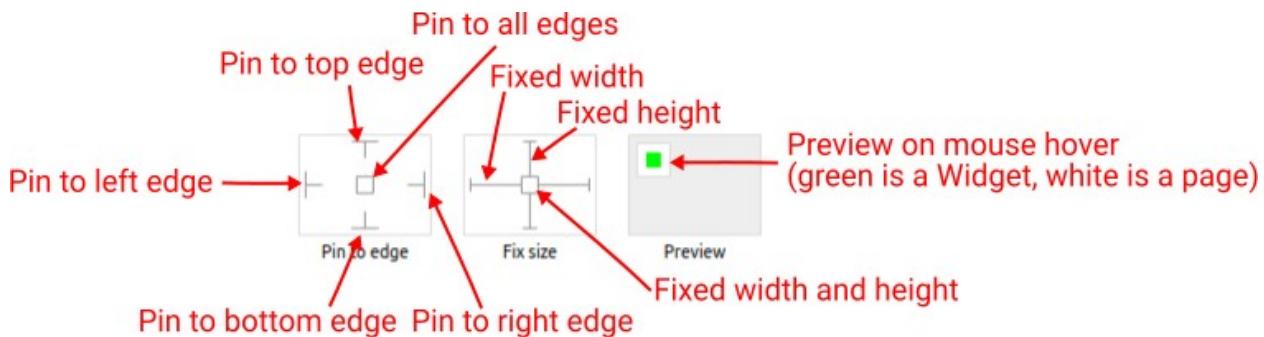
W34.2.2. Visible *EXPRESSION (boolean)*

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

W34.2.3. 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.

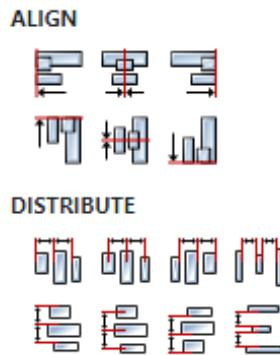
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W34.2.4. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

W34.2.5. 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.



W34.2.6. Center widget Any

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



W34.2.7. 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.`

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. Width Number

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

W34.2.10. Height Number

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

W34.2.11. Absolute position String

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

General

W34.2.12. 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.

Indentation

W34.2.13. First line Number

First line indentation.

W34.2.14. Hanging Number

Indentation of other lines.

Style

W34.2.15. Default Object

Style that will be used to render the Widget.

Events

W34.2.16. 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

W34.2.17. 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.

W34.2.18. 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.

W34.2.19. 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.

W34.3. Examples

- `eez-gui-widgets-demo`

W35. Panel



W35.1. 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.

W35.2. Properties

General

W35.2.1. Name *String*

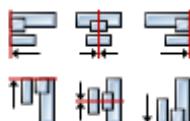
Widget name. We reference the Widget within the project by its name, for example in the LVGL action. For each Widget, we must choose a unique name within the entire project. This field is optional and does not need to be set if we do not need to reference the Widget.

Position and size

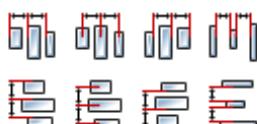
W35.2.2. Align and distribute *Any*

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

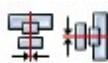


DISTRIBUTE



W35.2.3. Center widget *Any*

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



W35.2.4. 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`.

W35.2.5. 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.

W35.2.6. Top *Number*

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

W35.2.7. 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.

W35.2.8. Width *Number*

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

W35.2.9. 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.

W35.2.10. Height *Number*

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

W35.2.11. 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.

Style**W35.2.12. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W35.2.13. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W35.2.14. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W35.2.15. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W35.2.16. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W35.2.17. Click focusable *Boolean*

Add focused state to the object when clicked.

W35.2.18. Checkable *Boolean*

Toggle checked state when the object is clicked.

W35.2.19. Scrollable *Boolean*

Make the object scrollable.

W35.2.20. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W35.2.21. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W35.2.22. Scroll one *Boolean*

Allow scrolling only one snappable children.

W35.2.23. Scroll chain hor *Boolean*

Allow propagating the horizontal scroll to a parent.

W35.2.24. Scroll chain ver *Boolean*

Allow propagating the vertical scroll to a parent.

W35.2.25. Scroll chain *Boolean*

Allow propagating both the horizontal and the vertical scroll to a parent.

W35.2.26. Scroll on focus *Boolean*

Automatically scroll object to make it visible when focused.

W35.2.27. Scroll with arrow *Boolean*

Allow scrolling the focused object with arrow keys.

W35.2.28. Snappable *Boolean*

If scroll snap is enabled on the parent it can snap to this object.

W35.2.29. Press lock *Boolean*

Keep the object pressed even if the press slid from the object.

W35.2.30. Event bubble *Boolean*

Propagate the events to the parent too.

W35.2.31. Gesture bubble *Boolean*

Propagate the gestures to the parent.

W35.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W35.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W35.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W35.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States

W35.2.36. Checked EXPRESSION (boolean)

Toggled or checked state.

W35.2.37. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W35.2.38. Disabled EXPRESSION (boolean)

Disabled state

W35.2.39. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W35.2.40. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W35.2.41. Pressed Boolean

Being pressed.

Events

W35.2.42. 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**W35.2.43. 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.

W35.2.44. 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.

W35.2.45. 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. Progress (Dashboard)



W36.1. Description

We can use this Widget, for example, when we want to display the execution progress of an operation.

W36.2. Properties

Specific

W36.2.1. Data *EXPRESSION (integer)*

A value that goes from Min (progress is at 0%) to Max (progress is at 100%).

W36.2.2. Min *EXPRESSION (any)*

The minimum value that **Data** can contain.

W36.2.3. Max *EXPRESSION (any)*

The maximum value that **Data** can contain.

W36.2.4. Orientation *Enum*

There are two orientations that the **Progress** widget can have:

- Horizontal
- Vertical

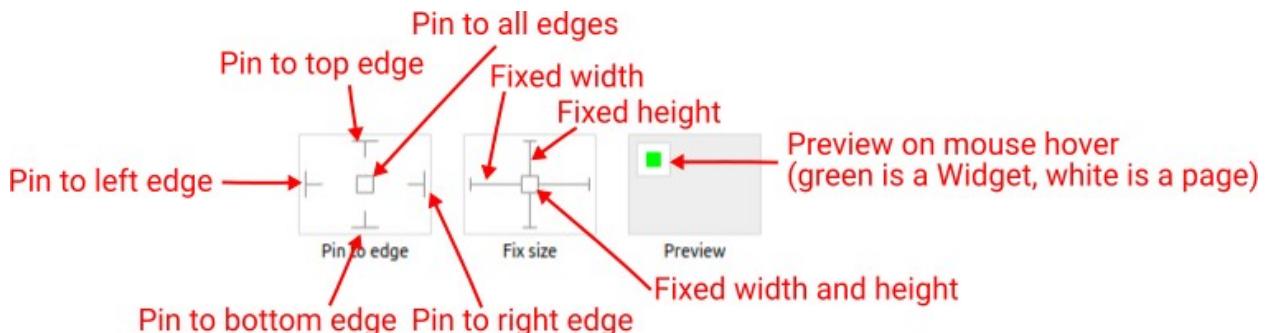
W36.2.5. Visible *EXPRESSION (boolean)*

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

W36.2.6. 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.

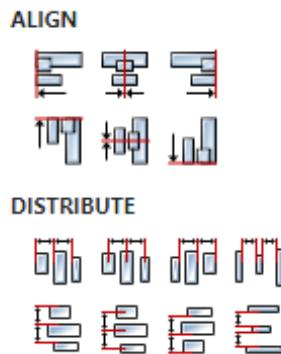
Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the

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*.

W36.2.7. 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.



W36.2.8. Center widget Any

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



W36.2.9. 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.`

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. Width Number

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

W36.2.12. Height Number

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

W36.2.13. Absolute position String

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

Style

W36.2.14. Default Object

Style used when rendering of the Widget.

Events

W36.2.15. 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

W36.2.16. 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.

W36.2.17. 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.

W36.2.18. 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.

W36.3. Examples

- *Dashboard Widgets Demo*



W37.1. Description

We can use this Widget, for example, when we want to display the execution progress of an operation.

W37.2. Properties

Specific

W37.2.1. Data *EXPRESSION (integer)*

A value that goes from Min (progress is at 0%) to Max (progress is at 100%).

W37.2.2. Min *EXPRESSION (any)*

The minimum value that `Data` can contain.

W37.2.3. Max *EXPRESSION (any)*

The maximum value that `Data` can contain.

W37.2.4. Orientation *Enum*

There are two orientations that the `Progress` widget can have:

- Horizontal



- Vertical



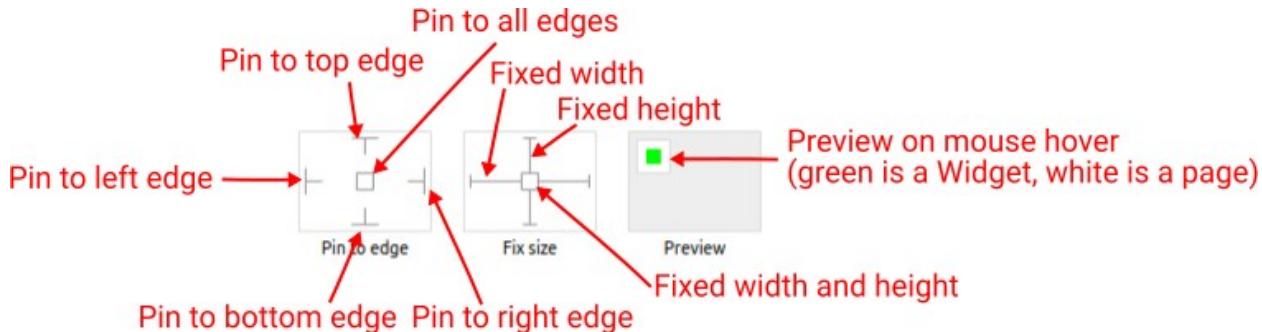
W37.2.5. Visible *EXPRESSION (boolean)*

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

W37.2.6. 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.

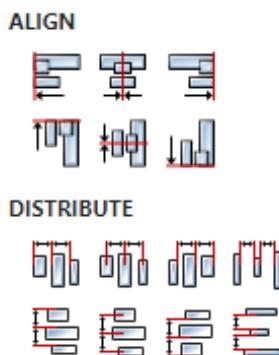
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W37.2.7. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W37.2.8. 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.



W37.2.9. Center widget Any

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



W37.2.10. 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`.

W37.2.11. Top *Number*

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

W37.2.12. Width *Number*

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

W37.2.13. Height *Number*

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

W37.2.14. Absolute position *String*

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

Style**W37.2.15. Default** *Object*

Style used when rendering of the Widget.

Events**W37.2.16. 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**W37.2.17. 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.

W37.2.18. 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.

W37.2.19. 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.

W37.3. Examples

- `eez-gui-widgets-demo`

W38. QRCode (Dashboard)



W38.1. Description

Renders a QR Code representing the given text string at the given error correction level.

W38.2. Properties

Specific

W38.2.1. Text *EXPRESSION (any)*

Text for which the QR Code will be generated.

W38.2.2. 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.

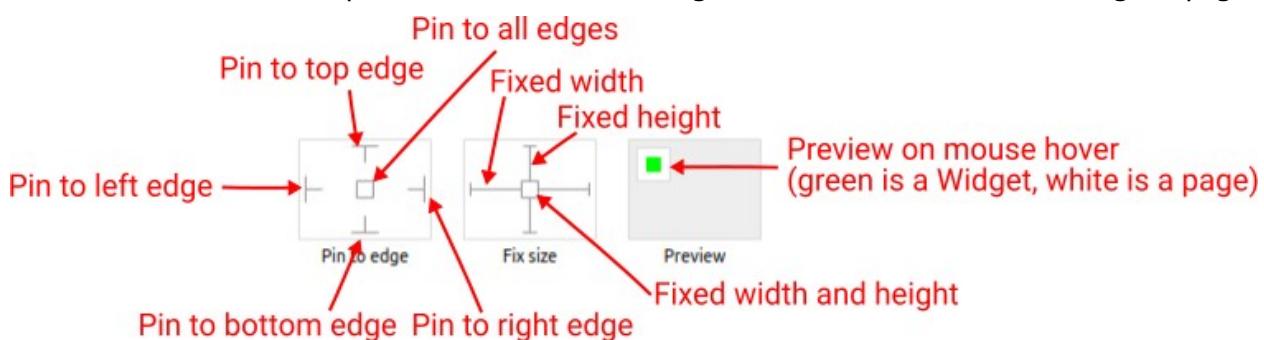
W38.2.3. Visible *EXPRESSION (boolean)*

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

W38.2.4. 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:



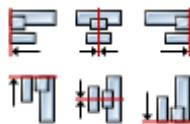
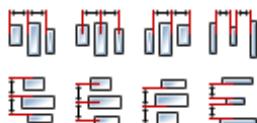
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

W38.2.5. 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****W38.2.6. Center widget** *Any*

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

**W38.2.7. 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`.

W38.2.8. Top *Number*

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

W38.2.9. Width *Number*

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

W38.2.10. Height *Number*

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

W38.2.11. Absolute position *String*

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

Style**W38.2.12. Default** *Object*

Style used when rendering of the Widget.

Events

W38.2.13. 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**W38.2.14. 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.

W38.2.15. 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.

W38.2.16. 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.

W38.3. Examples

- *Dashboard Widgets Demo*



W39.1. Description

Renders a QR Code representing the given text string at the given error correction level.

W39.2. Properties

Specific

W39.2.1. Text *EXPRESSION (any)*

Text for which the QR Code will be generated.

W39.2.2. 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.

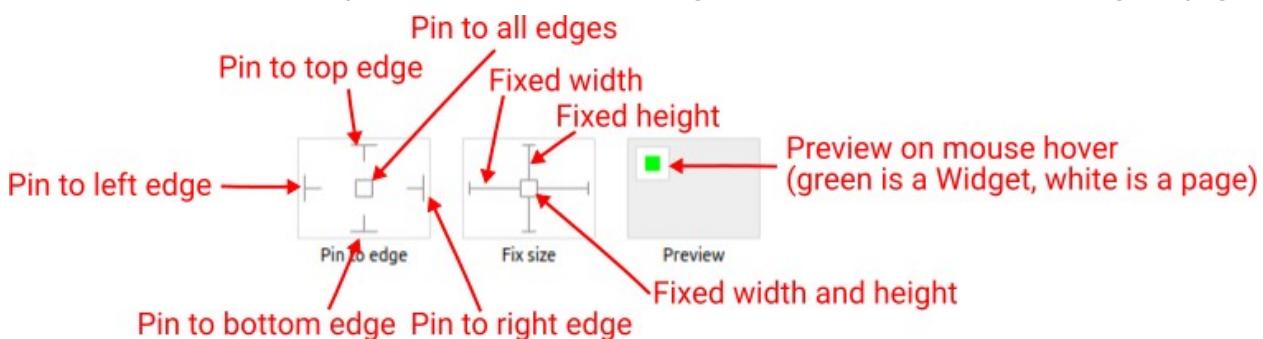
W39.2.3. Visible *EXPRESSION (boolean)*

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

W39.2.4. 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.

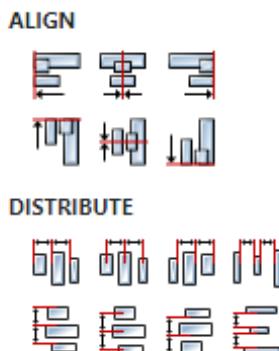
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W39.2.5. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

W39.2.6. 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.

**W39.2.7. Center widget** *Any*

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

**W39.2.8. 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`.

W39.2.9. Top *Number*

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

W39.2.10. Width *Number*

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

W39.2.11. Height *Number*

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

W39.2.12. Absolute position *String*

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

Style**W39.2.13. Default** *Object*

Style used when rendering of the Widget.

Events

W39.2.14. 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

W39.2.15. 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.

W39.2.16. 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.

W39.2.17. 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.

W39.3. Examples

- `eez-gui-widgets-demo`

W40. Rectangle (Dashboard)



W40.1. 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).

W40.2. Properties

Specific

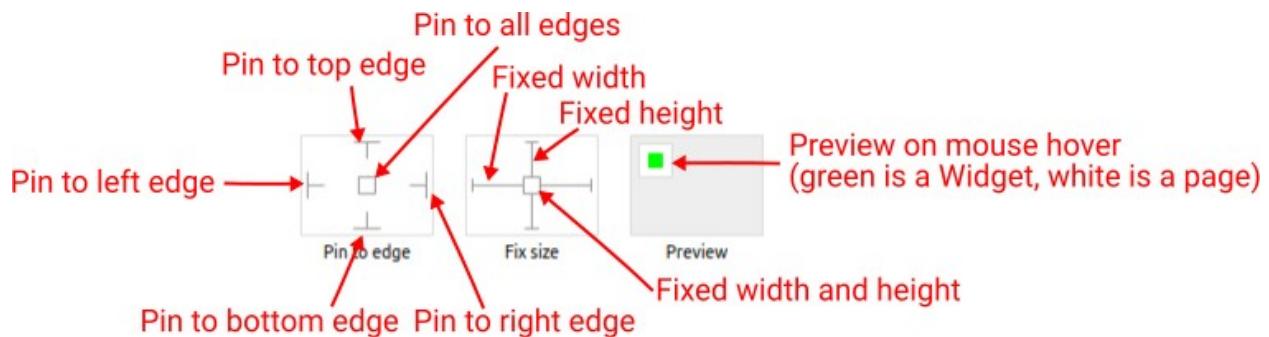
W40.2.1. Visible EXPRESSION (boolean)

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

W40.2.2. 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:



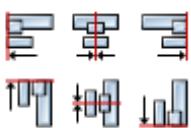
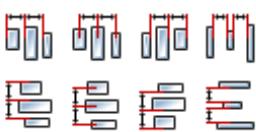
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

W40.2.3. 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****W40.2.4. Center widget Any**

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

**W40.2.5. 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.`

W40.2.6. Top Number

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

W40.2.7. Width Number

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

W40.2.8. Height Number

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

W40.2.9. Absolute position String

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

Style**W40.2.10. Default Object**

Style used when rendering the background of the Widget.

Events**W40.2.11. 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

W40.2.12. 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.

W40.2.13. 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.

W40.2.14. 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

- *Dashboard Widgets Demo*

W41. Rectangle (EEZ-GUI)



W41.1. 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).

W41.2. Properties

Specific

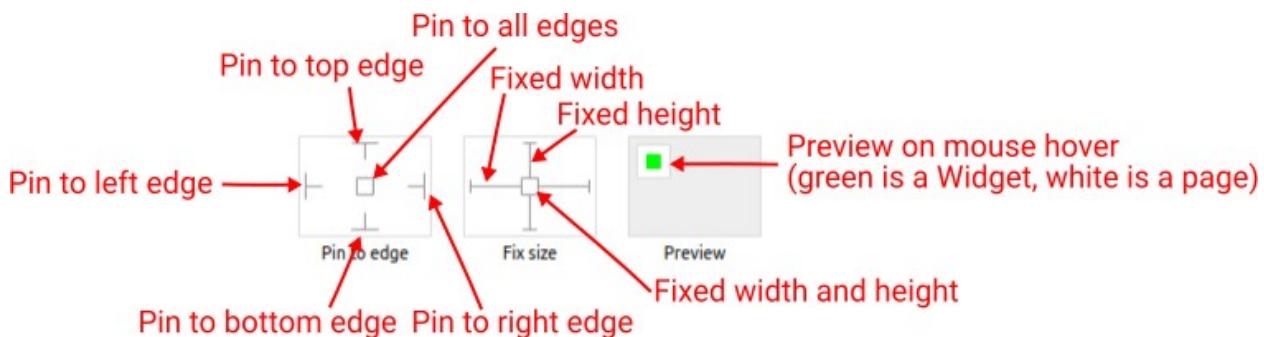
W41.2.1. Visible EXPRESSION (boolean)

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

W41.2.2. 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.

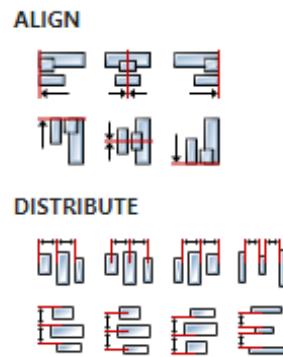
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W41.2.3. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W41.2.4. 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.

**W41.2.5. Center widget Any**

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

**W41.2.6. 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.

W41.2.7. Top Number

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

W41.2.8. Width Number

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

W41.2.9. Height Number

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

W41.2.10. Absolute position String

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

Style**W41.2.11. Default Object**

Style used when rendering the background of the Widget.

Events**W41.2.12. 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

W41.2.13. 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.

W41.2.14. 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.

W41.2.15. 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.

W42. Roller (EEZ-GUI)



W42.1. Description

This Widget allows us to select one option from a list using touch based scrolling.

W42.2. Properties

Specific

W42.2.1. Data *EXPRESSION (integer)*

The variable in which the selected value in the range of [Min, Max] is saved.

W42.2.2. Min *EXPRESSION (any)*

The minimum value that can be selected.

W42.2.3. Max *EXPRESSION (any)*

The maximum value that can be selected.

W42.2.4. 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].

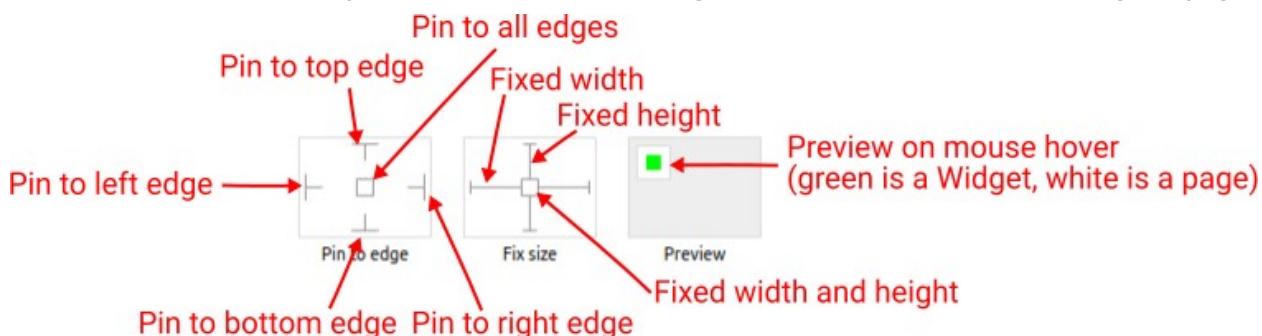
W42.2.5. Visible *EXPRESSION (boolean)*

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

W42.2.6. 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.

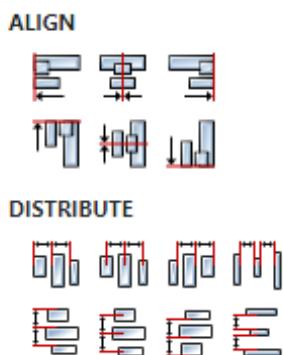
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W42.2.7. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W42.2.8. 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.



W42.2.9. Center widget Any

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



W42.2.10. 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`.

W42.2.11. Top Number

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

W42.2.12. Width Number

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

W42.2.13. Height Number

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

W42.2.14. Absolute position String

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

Style

W42.2.15. Default Object

Style used when rendering the background of the Widget.

W42.2.16. Selected value Object

Style used to render selected value.

W42.2.17. Unselected value Object

Style used to render other (unselected) values.

Events

W42.2.18. 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

W42.2.19. 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.

W42.2.20. 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.

W42.2.21. 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.

W42.3. Inputs

W42.3.1. 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.

W42.4. Examples

- *eez-gui-widgets-demo*

W43. Roller (LVGL)



W43.1. Description

This Widget allows us to select one option from a list using touch based scrolling.
More info ([link](#))

W43.2. Properties

Specific

W43.2.1. Options *EXPRESSION (array:string)*

List of options.

W43.2.2. 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.

W43.2.3. Selected *EXPRESSION (integer)*

The zero-based index of the selected option.

W43.2.4. 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.

W43.2.5. Mode *Enum*

Roller mode options:

- NORMAL – normal roller.
- INFINITE – makes the roller circular.

General

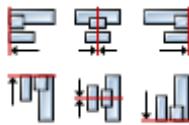
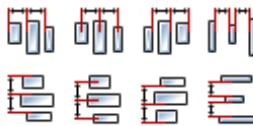
W43.2.6. Name *String*

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

W43.2.7. 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****W43.2.8. Center widget Any**

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

**W43.2.9. 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.

W43.2.10. 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.

W43.2.11. Top Number

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

W43.2.12. 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.

W43.2.13. Width Number

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

W43.2.14. 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.

W43.2.15. Height *Number*

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

W43.2.16. 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.

Style**W43.2.17. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W43.2.18. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W43.2.19. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W43.2.20. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W43.2.21. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W43.2.22. Click focusable *Boolean*

Add focused state to the object when clicked.

W43.2.23. Checkable *Boolean*

Toggle checked state when the object is clicked.

W43.2.24. Scrollable *Boolean*

Make the object scrollable.

W43.2.25. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W43.2.26. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W43.2.27. Scroll one *Boolean*

Allow scrolling only one snappable children.

W43.2.28. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W43.2.29. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W43.2.30. Scroll chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W43.2.31. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W43.2.32. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W43.2.33. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W43.2.34. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W43.2.35. Event bubble Boolean

Propagate the events to the parent too.

W43.2.36. Gesture bubble Boolean

Propagate the gestures to the parent.

W43.2.37. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W43.2.38. Ignore layout Boolean

Make the object positionable by the layouts.

W43.2.39. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W43.2.40. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States

W43.2.41. Checked EXPRESSION (boolean)

Toggled or checked state.

W43.2.42. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W43.2.43. Disabled *EXPRESSION (boolean)*

Disabled state

W43.2.44. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W43.2.45. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W43.2.46. Pressed *Boolean*

Being pressed.

Events**W43.2.47. 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**W43.2.48. 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.

W43.2.49. 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.

W43.2.50. 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.

W43.3. Examples

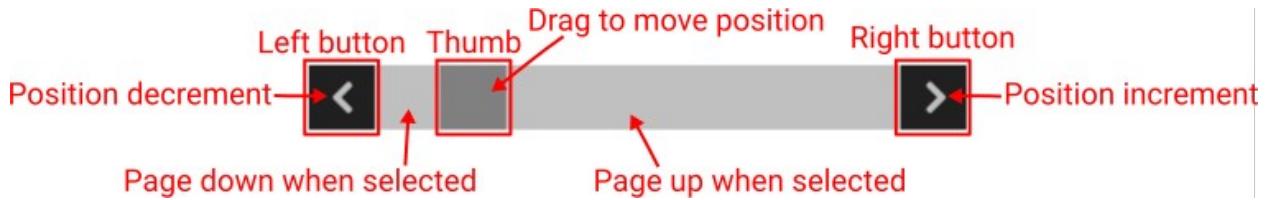
- *LVGL Widgets Demo*

W44. ScrollBar



W44.1. 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").

W44.2. Properties

Specific

W44.2.1. Data *EXPRESSION (struct:\$ScrollbarState)*

Set here the name of the `struct:$ScrollbarState` type variable.

W44.2.2. 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.

W44.2.3. 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.

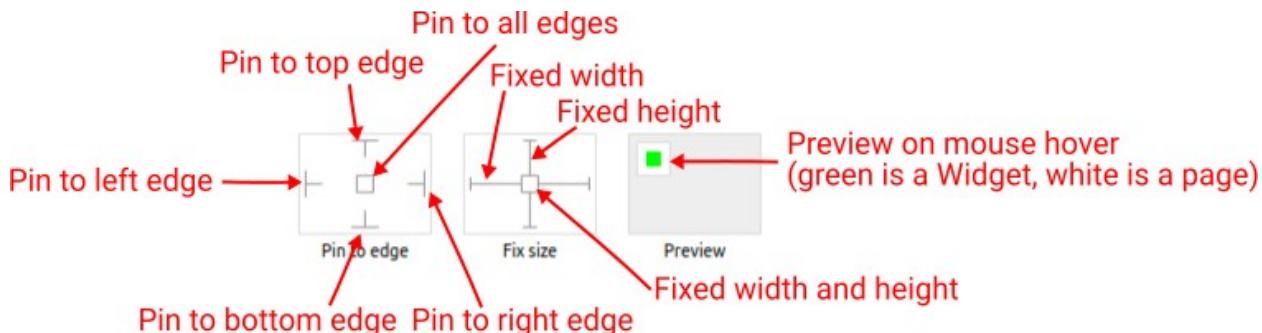
W44.2.4. Visible *EXPRESSION (boolean)*

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

W44.2.5. 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*.

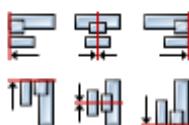
W44.2.6. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

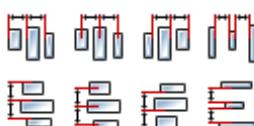
W44.2.7. Align and distribute Any

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

**W44.2.8. Center widget** Any

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

**W44.2.9. 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.

W44.2.10. Top Number

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

W44.2.11. Width Number

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

W44.2.12. Height Number

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

W44.2.13. Absolute position String

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

Style**W44.2.14. Default** Object

Style used when rendering the background of the Widget.

W44.2.15. Thumb Object

Style that will be used to render the scrollbar thumb.

W44.2.16. Buttons Object

Style used to render the left and right buttons.

Events**W44.2.17. 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**W44.2.18. 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.

W44.2.19. 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.

W44.2.20. 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.

W44.3. Examples

- `eez-gui-widgets-demo`

W45. Select



W45.1. 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.

W45.2. Properties

Specific

W45.2.1. 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.

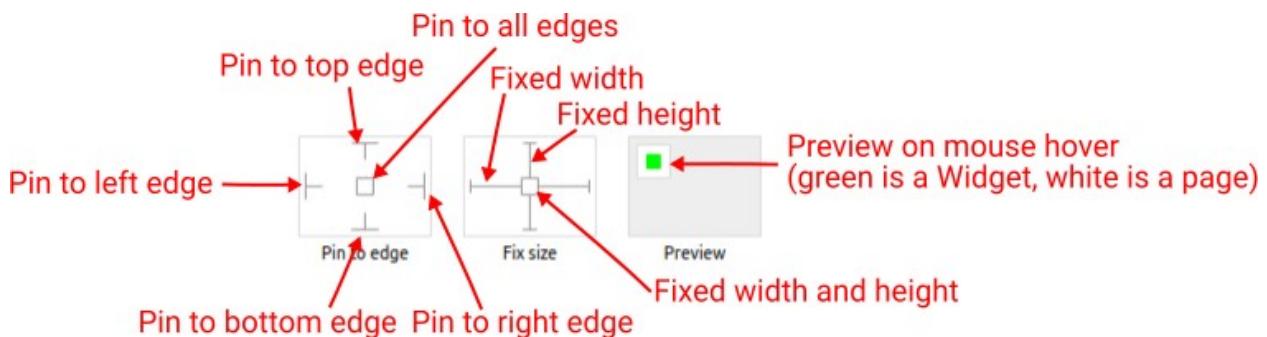
W45.2.2. Visible EXPRESSION (boolean)

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

W45.2.3. 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.

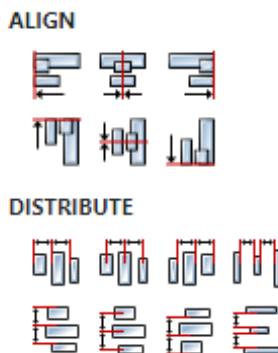
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W45.2.4. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

W45.2.5. 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.

**W45.2.6. Center widget** *Any*

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

**W45.2.7. 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`.

W45.2.8. Top *Number*

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

W45.2.9. Width *Number*

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

W45.2.10. Height *Number*

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

W45.2.11. Absolute position *String*

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

Style**W45.2.12. Default** *Object*

Style used when rendering the background of the Widget.

Events

W45.2.13. 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

W45.2.14. 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.

W45.2.15. 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.

W45.2.16. 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.

W45.3. Examples

- `eez-gui-widgets-demo`

W46. Slider (Dashboard)



W46.1. Description

This Widget allows us to select one value from the list by moving the knob on the slider.

W46.2. Properties

Specific

W46.2.1. Value *EXPRESSION (double)*

The variable in which the selected value in the range of [Min, Max] is saved.

W46.2.2. Min *EXPRESSION (double)*

The minimum value that can be selected.

W46.2.3. Max *EXPRESSION (double)*

The maximum value that can be selected.

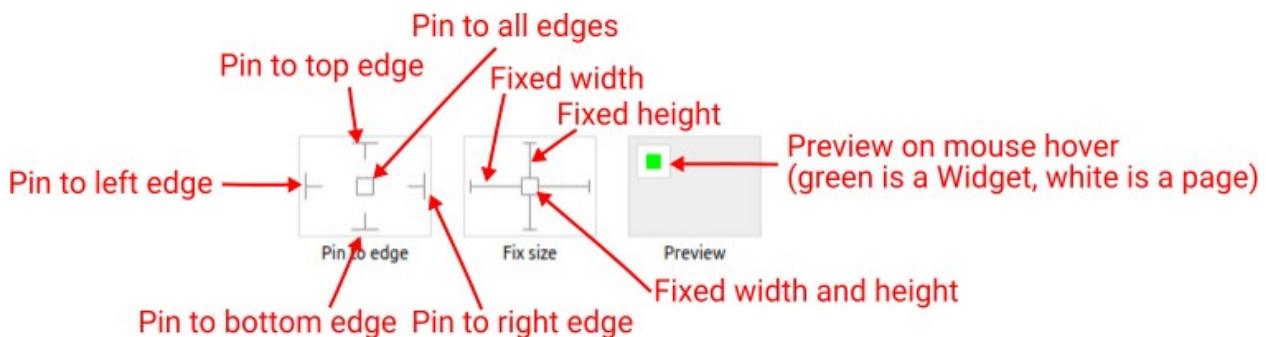
W46.2.4. Visible *EXPRESSION (boolean)*

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

W46.2.5. 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:



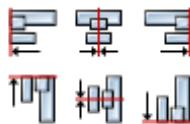
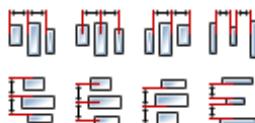
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

W46.2.6. 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****W46.2.7. Center widget Any**

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

**W46.2.8. 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.

W46.2.9. Top Number

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

W46.2.10. Width Number

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

W46.2.11. Height Number

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

W46.2.12. Absolute position String

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

Style**W46.2.13. Default Object**

Style used when rendering of the Widget.

Events

W46.2.14. 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

W46.2.15. 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.

W46.2.16. 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.

W46.2.17. 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.

W46.3. Examples

- *Dashboard Widgets Demo*

W47. Slider (EEZ-GUI)



W47.1. Description

This Widget allows us to select one value from the list by moving the knob on the slider.

W47.2. Properties

Specific

W47.2.1. Data *EXPRESSION (integer)*

The variable in which the selected value in the range of [Min, Max] is saved.

W47.2.2. Min *EXPRESSION (any)*

The minimum value that can be selected.

W47.2.3. Max *EXPRESSION (any)*

The maximum value that can be selected.

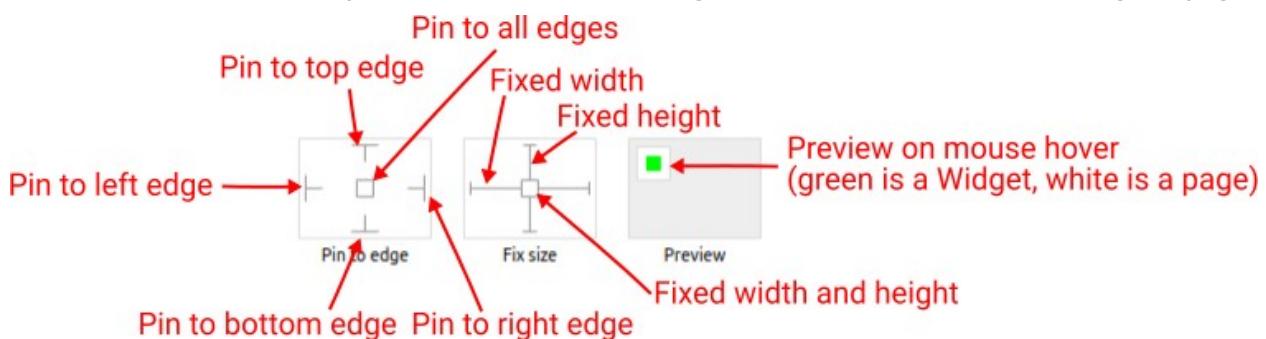
W47.2.4. Visible *EXPRESSION (boolean)*

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

W47.2.5. 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.

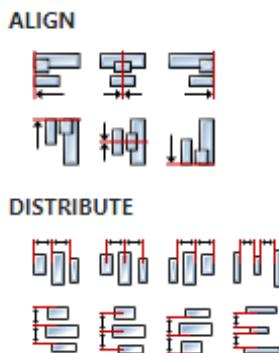
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W47.2.6. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

W47.2.7. 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.

**W47.2.8. Center widget** *Any*

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

**W47.2.9. 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`.

W47.2.10. Top *Number*

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

W47.2.11. Width *Number*

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

W47.2.12. Height *Number*

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

W47.2.13. Absolute position *String*

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

Style**W47.2.14. Default** *Object*

Style used when rendering of the Widget.

Events

W47.2.15. 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

W47.2.16. 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.

W47.2.17. 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.

W47.2.18. 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.

W47.3. Examples

- `eez-gui-widgets-demo`

W48. Slider (LVGL)



W48.1. Description

This Widget allows us to select one or two values from the list by moving the knob on the slider.
More info ([link](#))

W48.2. Properties

Specific

W48.2.1. Min *Number*

The minimum value that can be selected.

W48.2.2. Max *Number*

The maximum value that can be selected.

W48.2.3. 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).

W48.2.4. Value *EXPRESSION (integer)*

The selected value on the slider. If `RANGE` mode is selected then this is selected end value on the slider.

W48.2.5. 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.

W48.2.6. Value left *EXPRESSION (integer)*

If `RANGE` mode is selected then this is selected start value on the slider.

W48.2.7. 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.

General

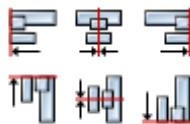
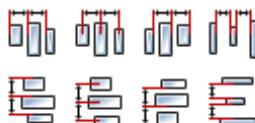
W48.2.8. Name *String*

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

W48.2.9. 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****W48.2.10. Center widget Any**

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

**W48.2.11. 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`.

W48.2.12. 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.

W48.2.13. Top Number

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

W48.2.14. 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.

W48.2.15. Width Number

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

W48.2.16. 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.

W48.2.17. Height *Number*

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

W48.2.18. 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.

Style

W48.2.19. Use style *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags

W48.2.20. Hidden *EXPRESSION (boolean)*

Make the object hidden.

W48.2.21. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W48.2.22. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W48.2.23. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W48.2.24. Click focusable *Boolean*

Add focused state to the object when clicked.

W48.2.25. Checkable *Boolean*

Toggle checked state when the object is clicked.

W48.2.26. Scrollable *Boolean*

Make the object scrollable.

W48.2.27. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W48.2.28. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W48.2.29. Scroll one Boolean

Allow scrolling only one snappable children.

W48.2.30. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W48.2.31. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W48.2.32. Scroll chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W48.2.33. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W48.2.34. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W48.2.35. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W48.2.36. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W48.2.37. Event bubble Boolean

Propagate the events to the parent too.

W48.2.38. Gesture bubble Boolean

Propagate the gestures to the parent.

W48.2.39. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W48.2.40. Ignore layout Boolean

Make the object positionable by the layouts.

W48.2.41. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W48.2.42. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W48.2.43. Checked EXPRESSION (boolean)**

Toggled or checked state.

W48.2.44. Checked state type *Enum*

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W48.2.45. Disabled *EXPRESSION (boolean)*

Disabled state

W48.2.46. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W48.2.47. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W48.2.48. Pressed *Boolean*

Being pressed.

Events**W48.2.49. 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**W48.2.50. 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.

W48.2.51. 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.

W48.2.52. 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.

W48.3. Examples

- *Dashboard Widgets Demo*

W49. Spinner (Dashboard)



W49.1. Description

We use this Widget to show that some operation is in progress or something is loading, etc.

W49.2. Properties

Specific

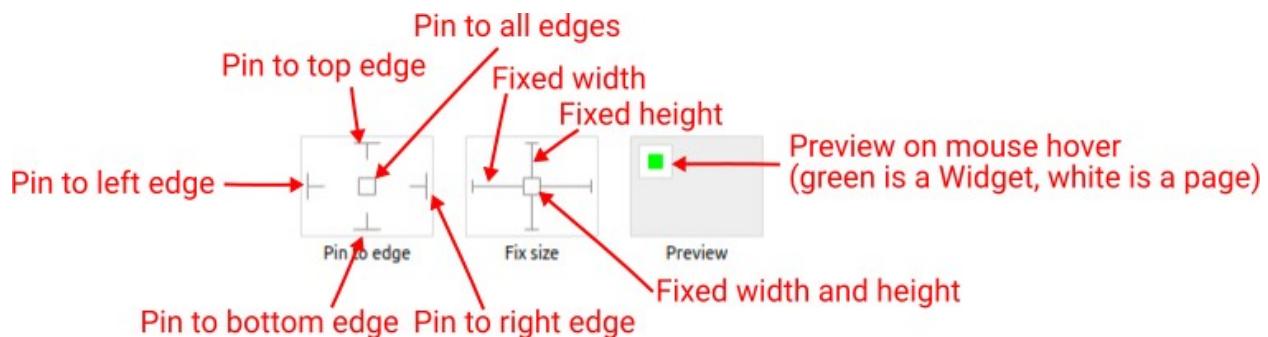
W49.2.1. Visible EXPRESSION (boolean)

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

W49.2.2. 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:



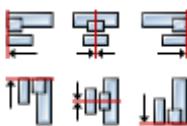
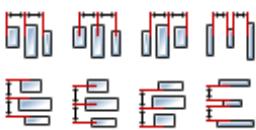
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

W49.2.3. 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****W49.2.4. Center widget Any**

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

**W49.2.5. 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.`

W49.2.6. Top Number

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

W49.2.7. Width Number

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

W49.2.8. Height Number

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

W49.2.9. Absolute position String

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

Style**W49.2.10. Default Object**

Style used when rendering of the Widget.

Events**W49.2.11. 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

W49.2.12. 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.

W49.2.13. 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.

W49.2.14. 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.

W49.3. Examples

- Dashboard Widgets Demo

W50. Spinner (LVGL)



W50.1. Description

Use this Widget to show that some operation is in progress or something is loading, etc.

W50.2. Properties

General

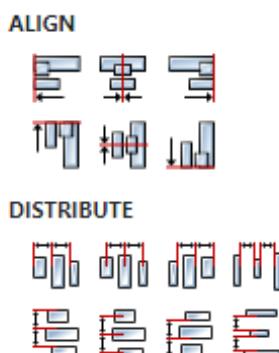
W50.2.1. Name *String*

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.



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.

Style**W50.2.12. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W50.2.13. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W50.2.14. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W50.2.15. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W50.2.16. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W50.2.17. Click focusable Boolean

Add focused state to the object when clicked.

W50.2.18. Checkable Boolean

Toggle checked state when the object is clicked.

W50.2.19. Scrollable Boolean

Make the object scrollable.

W50.2.20. Scroll elastic Boolean

Allow scrolling inside but with slower speed.

W50.2.21. Scroll momentum Boolean

Make the object scroll further when "thrown".

W50.2.22. Scroll one Boolean

Allow scrolling only one snappable children.

W50.2.23. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W50.2.24. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W50.2.25. Scroll chain Boolean

Allow propagating both the horizontal and 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.

States**W50.2.36. Checked EXPRESSION (boolean)**

Toggled or checked state.

W50.2.37. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W50.2.38. Disabled EXPRESSION (boolean)

Disabled state

W50.2.39. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W50.2.40. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W50.2.41. Pressed Boolean

Being pressed.

Events**W50.2.42. 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**W50.2.43. 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.

W50.2.44. 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.

W50.2.45. 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.

W50.3. Examples

- *Dashboard Widgets Demo*

W51. Switch (Dashboard)



W51.1. Description

Switch Widget is used when we want a turn ON or turn OFF option.

W51.2. Properties

Specific

W51.2.1. Value EXPRESSION (any)

Boolean variable in which `true` is stored when the switch is ON and `false` when the switch is OFF.

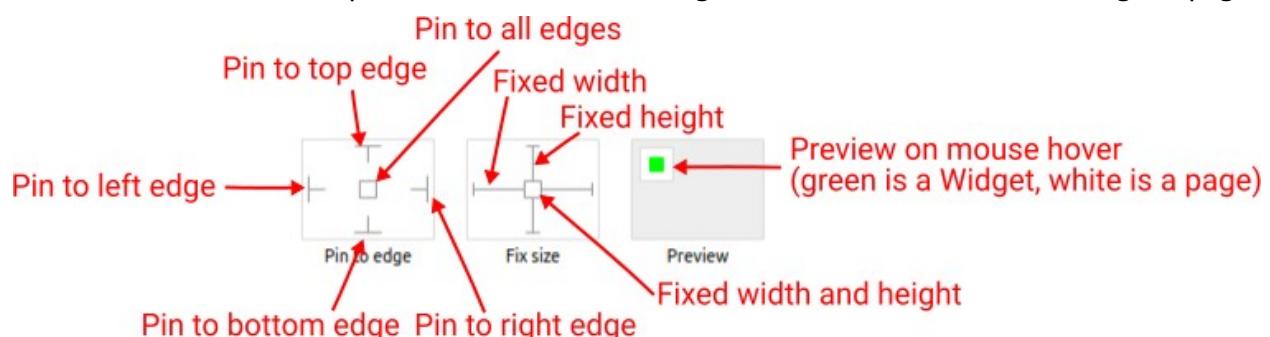
W51.2.2. Visible EXPRESSION (boolean)

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

W51.2.3. 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:



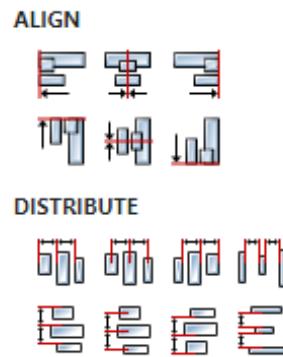
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

W51.2.4. 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.

**W51.2.5. Center widget Any**

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

**W51.2.6. 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.

W51.2.7. Top Number

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

W51.2.8. Width Number

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

W51.2.9. Height Number

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

W51.2.10. Absolute position String

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

Style**W51.2.11. Default Object**

Style used when rendering of the Widget.

Events**W51.2.12. 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

W51.2.13. 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.

W51.2.14. 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.

W51.2.15. 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.

W51.3. Examples

- Dashboard Widgets Demo

W52. Switch (EEZ-GUI)



W52.1. Description

Checkbox Widget is used when we want a turn ON or turn OFF option.

W52.2. Properties

Specific

W52.2.1. Data EXPRESSION (boolean)

Boolean variable in which `true` is stored when the switch is ON and `false` when the switch is OFF.

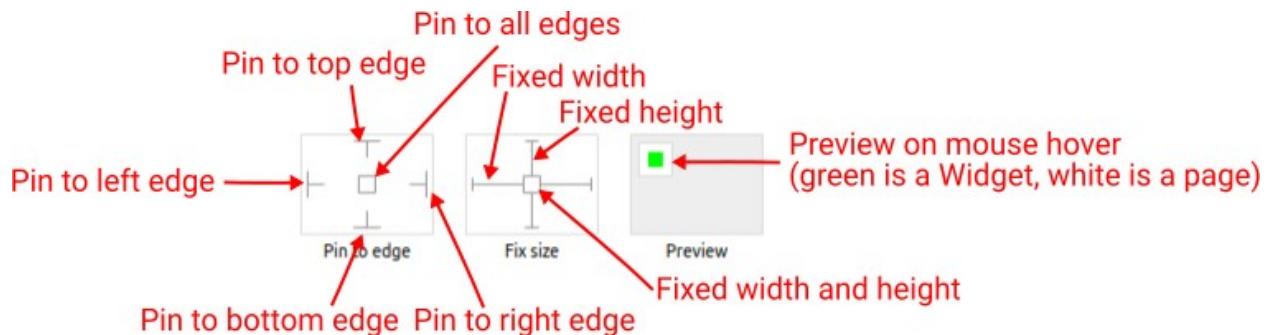
W52.2.2. Visible EXPRESSION (boolean)

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

W52.2.3. 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.

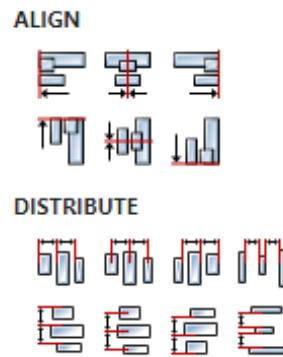
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W52.2.4. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W52.2.5. 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.

**W52.2.6. Center widget Any**

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

**W52.2.7. 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.

W52.2.8. Top Number

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

W52.2.9. Width Number

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

W52.2.10. Height Number

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

W52.2.11. Absolute position String

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

Style**W52.2.12. Default Object**

Style used when rendering of the Widget.

Events**W52.2.13. 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

W52.2.14. 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.

W52.2.15. 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.

W52.2.16. 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.

W52.3. Examples

- *eez-gui-widgets-demo*

W53. Switch (LVGL)



W53.1. Description

Switch Widget is used when we want a turn ON or turn OFF option.
More info ([link](#))

W53.2. Properties

General

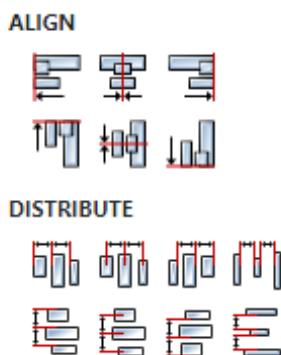
W53.2.1. Name *String*

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

W53.2.2. 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.



W53.2.3. Center widget *Any*

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



W53.2.4. 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.

W53.2.5. 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.

W53.2.6. Top *Number*

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

W53.2.7. 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.

W53.2.8. Width *Number*

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

W53.2.9. 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.

W53.2.10. Height *Number*

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

W53.2.11. 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.

Style**W53.2.12. Use style *ObjectReference***

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W53.2.13. Hidden *EXPRESSION (boolean)***

Make the object hidden.

W53.2.14. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W53.2.15. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W53.2.16. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or

not.

W53.2.17. Click focusable Boolean

Add focused state to the object when clicked.

W53.2.18. Checkable Boolean

Toggle checked state when the object is clicked.

W53.2.19. Scrollable Boolean

Make the object scrollable.

W53.2.20. Scroll elastic Boolean

Allow scrolling inside but with slower speed.

W53.2.21. Scroll momentum Boolean

Make the object scroll further when "thrown".

W53.2.22. Scroll one Boolean

Allow scrolling only one snappable children.

W53.2.23. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W53.2.24. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W53.2.25. Scroll chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W53.2.26. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W53.2.27. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W53.2.28. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W53.2.29. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W53.2.30. Event bubble Boolean

Propagate the events to the parent too.

W53.2.31. Gesture bubble Boolean

Propagate the gestures to the parent.

W53.2.32. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W53.2.33. Ignore layout Boolean

Make the object positionable by the layouts.

W53.2.34. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W53.2.35. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W53.2.36. Checked EXPRESSION (boolean)**

Toggled or checked state.

W53.2.37. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W53.2.38. Disabled EXPRESSION (boolean)

Disabled state

W53.2.39. Disabled state type Enum

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W53.2.40. Focused Boolean

Focused via keypad or encoder or clicked via touchpad/mouse.

W53.2.41. Pressed Boolean

Being pressed.

Events**W53.2.42. 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**W53.2.43. 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.

W53.2.44. 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.

W53.2.45. 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

- *Dashboard Widgets Demo*

W54. Terminal



W54.1. 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.

W54.2. Properties

Specific

W54.2.1. 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.

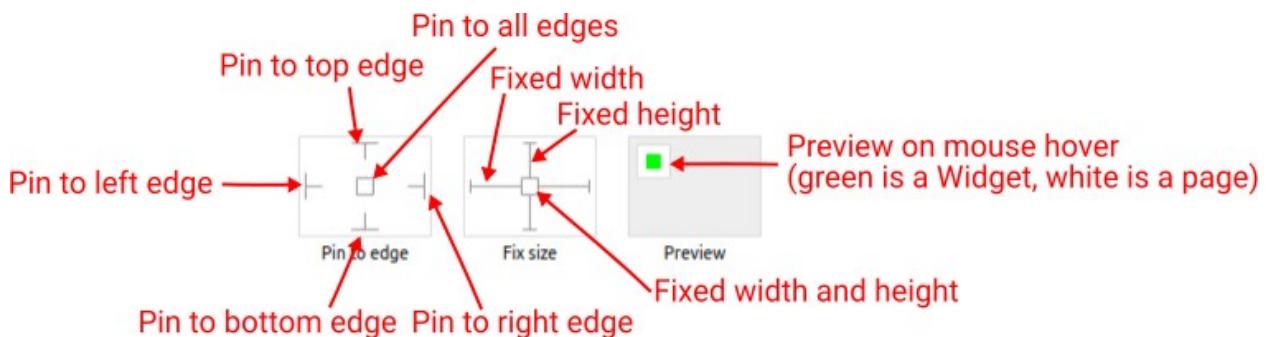
W54.2.2. Visible EXPRESSION (boolean)

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

W54.2.3. 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.

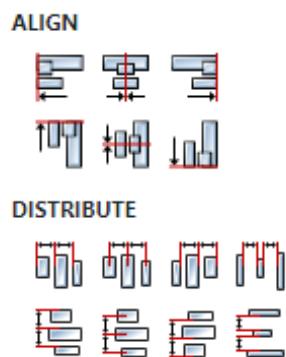
Using the *Fix size* option, we can fix the width/height of the Widget, i.e. if this option is selected the width/height will always be the same, and if not selected the width/height will scale proportionally as the page height scales.

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*.

W54.2.4. 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.



W54.2.5. Center widget Any

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



W54.2.6. 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`.

W54.2.7. Top Number

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

W54.2.8. Width Number

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

W54.2.9. Height Number

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

W54.2.10. Absolute position String

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

Style

W54.2.11. Default Object

Style used when rendering of the Widget.

Events

W54.2.12. 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

W54.2.13. 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.

W54.2.14. 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.

W54.2.15. 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.

W54.3. Examples

- *Dashboard Widgets Demo*

W55. Textarea



W55.1. 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](#))

W55.2. Properties

Specific

W55.2.1. Text *EXPRESSION (string)*

Text to be displayed.

W55.2.2. Text type *Enum*

Here we can choose that the `Text` item is calculated from the Expression.

W55.2.3. Placeholder *String*

A placeholder text can be specified – which is displayed when the `Text` area is empty.

W55.2.4. 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.

W55.2.5. 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.

W55.2.6. Accepted characters *String*

We can set a list of accepted characters with this property. Other characters will be ignored.

W55.2.7. Max text length *Number*

The maximum number of characters can be limited with this property.

General

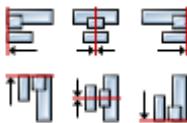
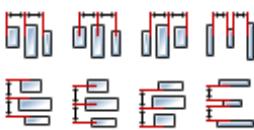
W55.2.8. Name *String*

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.

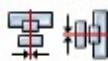
Position and size

W55.2.9. 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****W55.2.10. Center widget Any**

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

**W55.2.11. 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.

W55.2.12. 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.

W55.2.13. Top Number

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

W55.2.14. 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.

W55.2.15. Width Number

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

W55.2.16. 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.

W55.2.17. Height *Number*

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

W55.2.18. 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.

Style**W55.2.19. Use style** *ObjectReference*

Here we can select one of the globally defined Styles so that the Widget uses that Style.

Flags**W55.2.20. Hidden** *EXPRESSION (boolean)*

Make the object hidden.

W55.2.21. Hidden flag type *Enum*

Here we can choose whether the `Hidden` flag state will be calculated from the Expression or not.

W55.2.22. Clickable *EXPRESSION (boolean)*

Make the object clickable by input devices.

W55.2.23. Clickable flag type *Enum*

Here we can choose whether the `Clickable` flag state will be calculated from the Expression or not.

W55.2.24. Click focusable *Boolean*

Add focused state to the object when clicked.

W55.2.25. Checkable *Boolean*

Toggle checked state when the object is clicked.

W55.2.26. Scrollable *Boolean*

Make the object scrollable.

W55.2.27. Scroll elastic *Boolean*

Allow scrolling inside but with slower speed.

W55.2.28. Scroll momentum *Boolean*

Make the object scroll further when "thrown".

W55.2.29. Scroll one *Boolean*

Allow scrolling only one snappable children.

W55.2.30. Scroll chain hor Boolean

Allow propagating the horizontal scroll to a parent.

W55.2.31. Scroll chain ver Boolean

Allow propagating the vertical scroll to a parent.

W55.2.32. Scroll chain Boolean

Allow propagating both the horizontal and the vertical scroll to a parent.

W55.2.33. Scroll on focus Boolean

Automatically scroll object to make it visible when focused.

W55.2.34. Scroll with arrow Boolean

Allow scrolling the focused object with arrow keys.

W55.2.35. Snappable Boolean

If scroll snap is enabled on the parent it can snap to this object.

W55.2.36. Press lock Boolean

Keep the object pressed even if the press slid from the object.

W55.2.37. Event bubble Boolean

Propagate the events to the parent too.

W55.2.38. Gesture bubble Boolean

Propagate the gestures to the parent.

W55.2.39. Adv hittest Boolean

Allow performing more accurate hit (click) test. E.g. accounting for rounded corners.

W55.2.40. Ignore layout Boolean

Make the object positionable by the layouts.

W55.2.41. Floating Boolean

Do not scroll the object when the parent scrolls and ignore layout.

W55.2.42. Overflow visible Boolean

Do not clip the children's content to the parent's boundary.

States**W55.2.43. Checked EXPRESSION (boolean)**

Toggled or checked state.

W55.2.44. Checked state type Enum

Here we can choose whether the `Checked` state will be calculated from the Expression or not.

W55.2.45. Disabled *EXPRESSION (boolean)*

Disabled state

W55.2.46. Disabled state type *Enum*

Here we can choose whether the `Disabled` state will be calculated from the Expression or not.

W55.2.47. Focused *Boolean*

Focused via keypad or encoder or clicked via touchpad/mouse.

W55.2.48. Pressed *Boolean*

Being pressed.

Events**W55.2.49. 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**W55.2.50. 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.

W55.2.51. 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.

W55.2.52. 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.

W55.3. Examples

- *LVGL Widgets Demo*

W56. Text (Dashboard)



W56.1. Description

A widget used to display text.

W56.2. Properties

Specific

W56.2.1. 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.

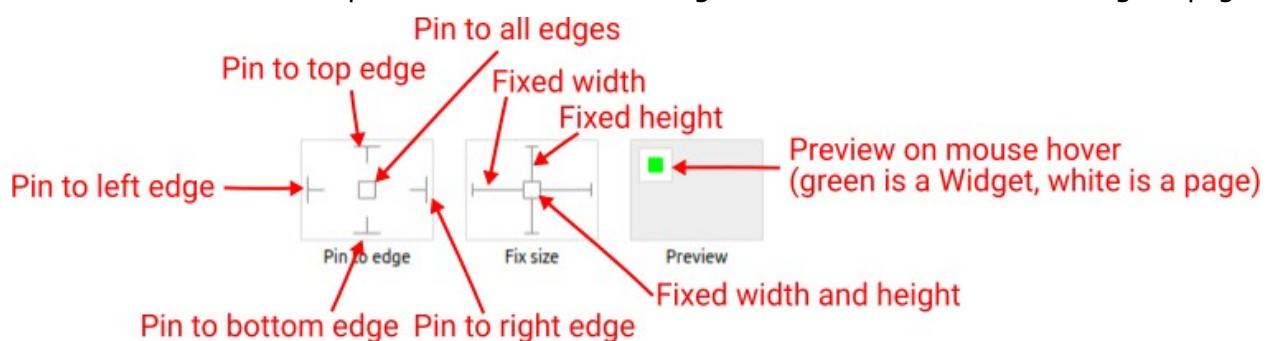
W56.2.2. Visible *EXPRESSION (boolean)*

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

W56.2.3. 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:



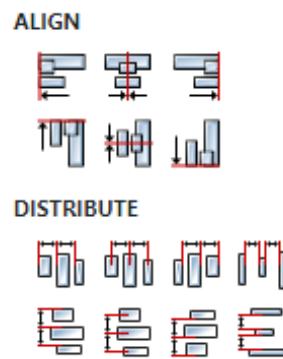
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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*.

W56.2.4. 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.



W56.2.5. Center widget Any

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



W56.2.6. 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.

W56.2.7. Top Number

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

W56.2.8. Width Number

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

W56.2.9. Height Number

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

W56.2.10. Absolute position String

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

General

W56.2.11. 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

W56.2.12. Default Object

Style that will be used to render the Widget.

Events

W56.2.13. 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

W56.2.14. 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.

W56.2.15. 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.

W56.2.16. 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.

W56.3. Examples

- *Dashboard Widgets Demo*

W57. Text (EEZ-GUI)



W57.1. Description

A widget used to display single line text.

W57.2. Properties

Specific

W57.2.1. 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.

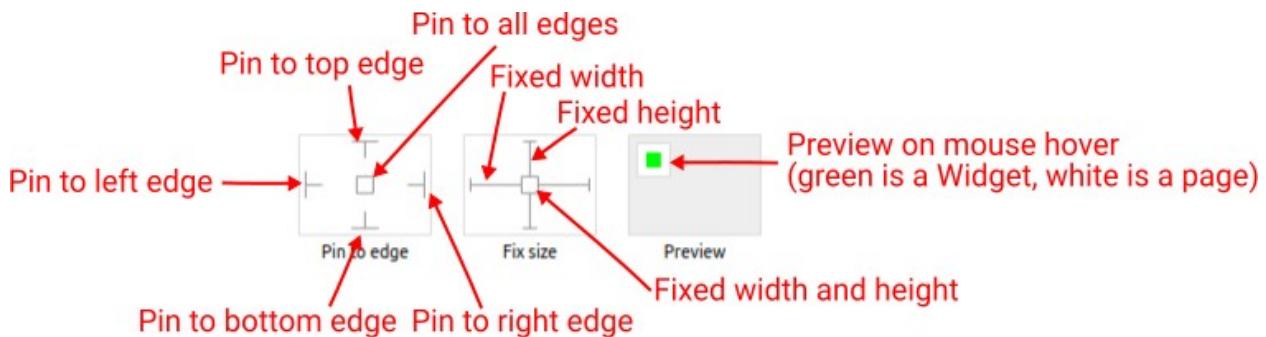
W57.2.2. Visible *EXPRESSION (boolean)*

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

W57.2.3. 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.

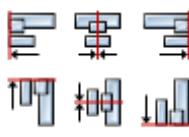
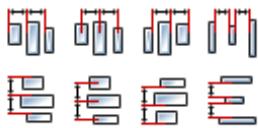
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W57.2.4. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

W57.2.5. 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****W57.2.6. Center widget Any**

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

**W57.2.7. 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.`

W57.2.8. Top Number

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

W57.2.9. Width Number

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

W57.2.10. Height Number

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

W57.2.11. Absolute position String

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

General**W57.2.12. 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**W57.2.13. Default Object**

Style that will be used to render the Widget.

W57.2.14. Focused Object

Style to be used for rendering if the Widget is in focus.

Events

W57.2.15. 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

W57.2.16. 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.

W57.2.17. 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.

W57.2.18. 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.

W57.3. Examples

- `eez-gui-widgets-demo`

W58. TextInput



W58.1. Description

This Widget is used when we want to enter a text.

W58.2. Properties

Specific

W58.2.1. Value *EXPRESSION (any)*

The variable in which the entered text will be stored.

W58.2.2. Placeholder *EXPRESSION (any)*

The text that is displayed at the beginning when nothing has been entered yet.

W58.2.3. Password *Boolean*

If password is entered, then this property should be enabled so that * is displayed instead of characters when entering the password.

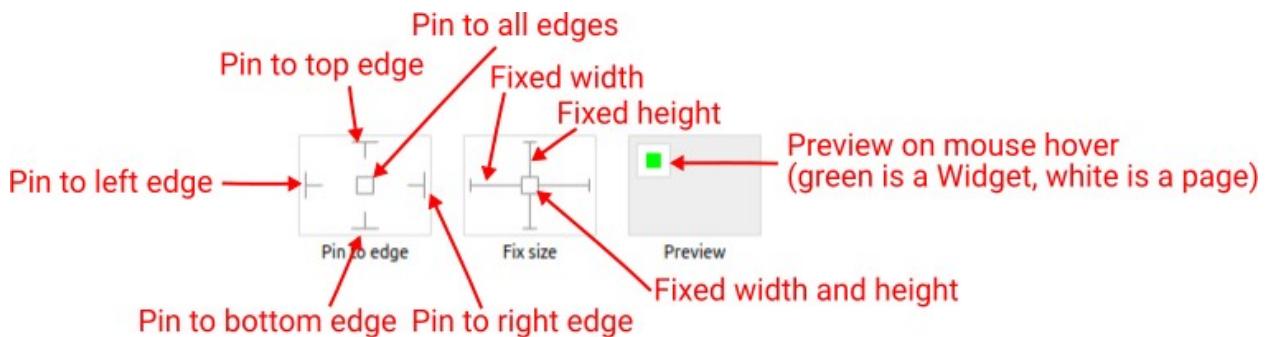
W58.2.4. Visible *EXPRESSION (boolean)*

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

W58.2.5. 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:



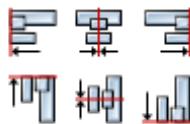
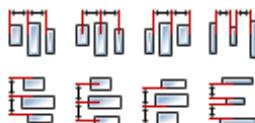
With the *Pin to edge* option we can fix the top, right, bottom and left edge of the Widget in relation to the page when it changes its original dimension because the *Scale to fit* option is selected. E.g. if we selected *Pin to top edge* then the distance between the top edge of the page and the top 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.

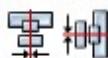
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W58.2.6. 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****W58.2.7. Center widget** *Any*

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

**W58.2.8. 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`.

W58.2.9. Top *Number*

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

W58.2.10. Width *Number*

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

W58.2.11. Height *Number*

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

W58.2.12. Absolute position *String*

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

Style**W58.2.13. Default** *Object*

Style used when rendering of the Widget.

Events

W58.2.14. 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

W58.2.15. 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.

W58.2.16. 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.

W58.2.17. 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.

W58.3. Examples

- *Dashboard Widgets Demo*

W59. ToggleButton



W59.1. Description

A button that can be in two states: `Default` or `Checked`.

W59.2. Properties

Specific

W59.2.1. 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

W59.2.2. Text1 *String*

The text that is displayed when the Widget is in the `Default` state.

W59.2.3. Text2 *String*

The text that is displayed when the Widget is in the `Checked` state.

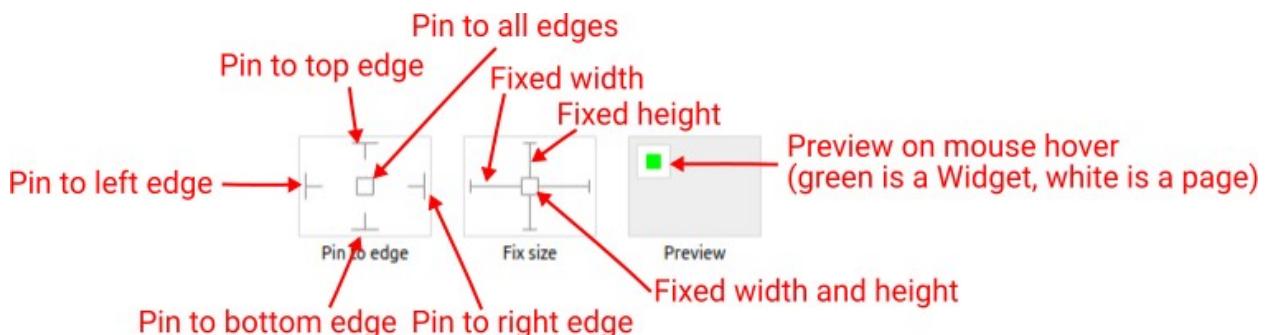
W59.2.4. Visible *EXPRESSION (boolean)*

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

W59.2.5. 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*.

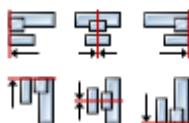
W59.2.6. Hide "Widget is outside of its parent" warning *Boolean*

Check when we want to hide "Widget is outside of its parent" warning message(s).

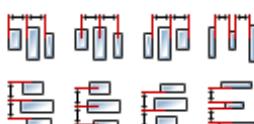
W59.2.7. Align and distribute *Any*

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

**W59.2.8. Center widget** *Any*

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

**W59.2.9. 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`.

W59.2.10. Top *Number*

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

W59.2.11. Width *Number*

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

W59.2.12. Height *Number*

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

W59.2.13. Absolute position *String*

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

Style**W59.2.14. Default** *Object*

Style to be used for rendering if the Widget is the `Default` state.

W59.2.15. Checked Object

Style to be used for rendering if the Widget is the `Checked` state.

Events**W59.2.16. 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**W59.2.17. 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.

W59.2.18. 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.

W59.2.19. 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.

W59.3. Examples

- `eez-gui-widgets-demo`

W60. UpDown



W60.1. Description

This Widget allows us to select a single value using the decrement and increment buttons.

W60.2. Properties

Specific

W60.2.1. Data *EXPRESSION (integer)*

The variable in which the selected value in the range of [Min, Max] is saved.

W60.2.2. Down button text *String*

The text that is displayed inside the button for the decrement value.

W60.2.3. Up button text *String*

The text that is displayed inside the button for the increment value.

W60.2.4. Min *EXPRESSION (any)*

The minimum value that can be selected.

W60.2.5. Max *EXPRESSION (any)*

The maximum value that can be selected.

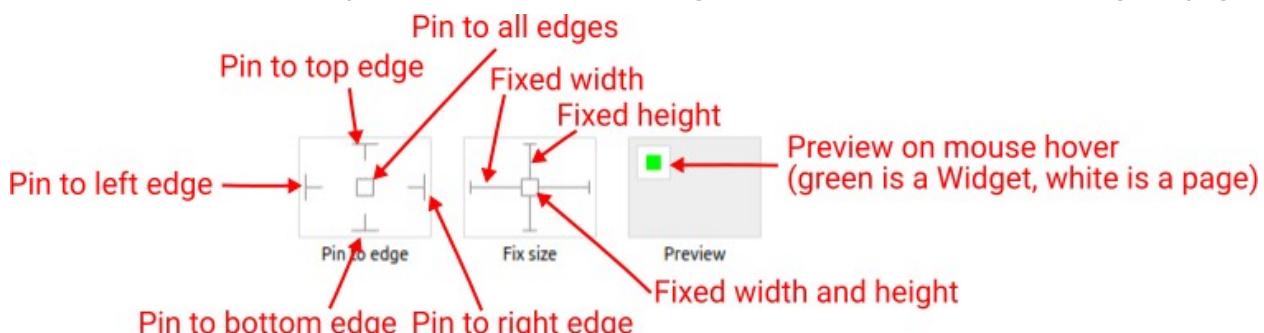
W60.2.6. Visible *EXPRESSION (boolean)*

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

W60.2.7. 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.

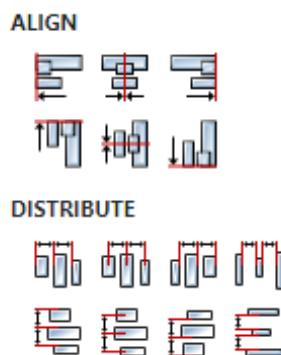
Note: If *Pin to left edge* and *Pin to right edge* are selected, then the *Fix width* option will be disabled, and conversely if *Fix width* is selected, then both *Pin to left edge* and *Pin to right edge* cannot be selected, because both cannot be satisfied. The same applies to *Pin to top edge* and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

W60.2.8. Hide "Widget is outside of its parent" warning Boolean

Check when we want to hide "Widget is outside of its parent" warning message(s).

W60.2.9. 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.



W60.2.10. Center widget Any

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



W60.2.11. 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`.

W60.2.12. Top Number

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

W60.2.13. Width Number

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

W60.2.14. Height Number

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

W60.2.15. Absolute position String

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

Style

W60.2.16. Default Object

Style used when rendering the background of the Widget and text.

W60.2.17. Buttons Object

Style used to render the button.

Events

W60.2.18. 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

W60.2.19. 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.

W60.2.20. 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.

W60.2.21. 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.

W60.3. Examples

- [eez-gui-widgets-demo](#)

*EEZ Studio
Instrument*

Table of Contents

I1.Home page instrument sections.....	I.5
I1.1.History.....	I.5
I1.2.Shortcuts and Groups.....	I.5
I1.3.Notebooks.....	I.6
I1.4.Items purge and restore.....	I.11
I1.5.Instrument Extension (IEXT) Manager.....	I.12
I1.6.Add instrument.....	I.13
I1.7.Establishing a connection with the instrument.....	I.15
I2.Instrument activity bar.....	I.17
I2.1.Start page (EEZ BB3 only).....	I.17
I2.2.Dashboard.....	I.19
I2.3.Terminal.....	I.19
I2.4.Scripts.....	I.20
I2.4.1.Edit script shortcut.....	I.22
I2.5.Shortcuts.....	I.23
I2.6.Lists.....	I.24
I2.6.1.Editing a list using a table.....	I.25
I2.6.2.Editing a list using an envelope.....	I.26
I2.6.3.List view options.....	I.28
I2.6.4.List help.....	I.28

I1. Home page instrument sections

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

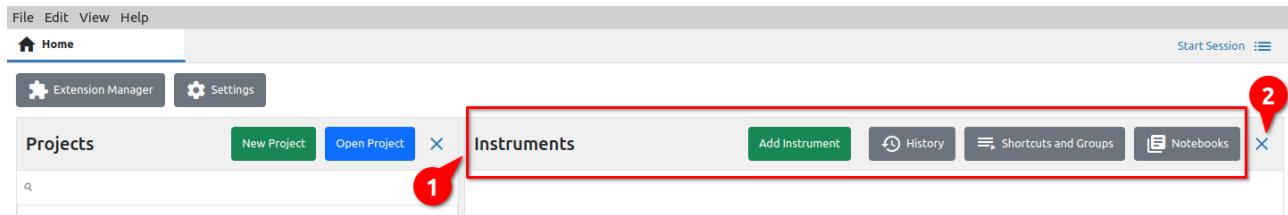


Fig. 1: Home page instrument options



Fig. 2: Home page "Show instruments" option

I1.1. History

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

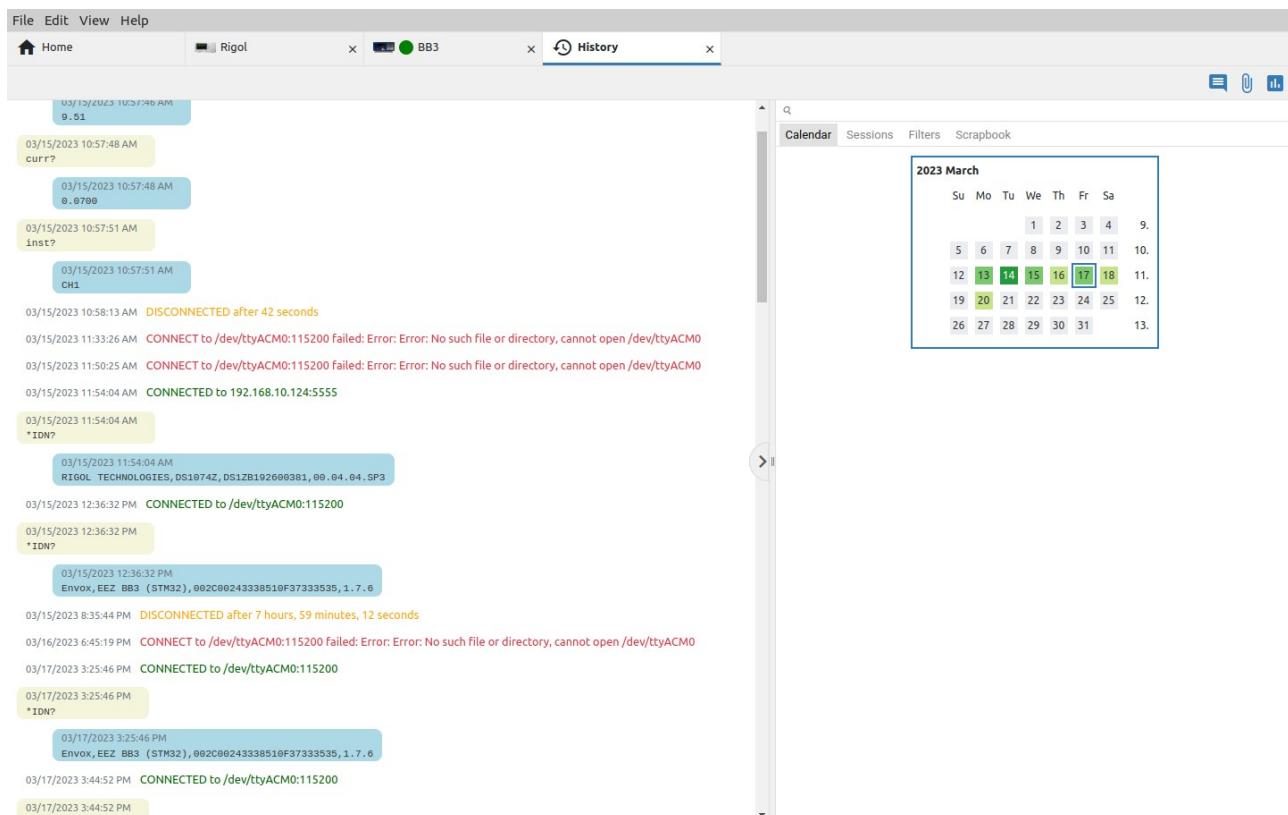


Fig. 3: Instruments History view

I1.2. Shortcuts and Groups

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

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

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	F5	JavaScript	✓	✓	5
Dlog abort	EEZ BB3 STM32 EEZ BB3 Simulator EEZ H24005 r3B4	—	SCPI	✓	✓	15
Dlog start	EEZ BB3 STM32 EEZ BB3 Simulator	—	JavaScript	✓	✓	13
Dlog start	EEZ H24005 r3B4	—	JavaScript	✓	✓	13
Dlog upload	EEZ BB3 STM32 EEZ BB3 Simulator	—	JavaScript	✓	✓	14
Dlog upload	EEZ H24005 r3B4	—	JavaScript	✓	✓	14
Init	EEZ BB3 STM32 EEZ BB3 Simulator EEZ H24005 r3B4	F8	SCPI	✓	✓	8
Outputs OFF	EEZ BB3 STM32 EEZ BB3 Simulator	F1	SCPI	✓	✓	1
Outputs OFF	EEZ H24005 r3B4	F1	SCPI	✓	✓	1
Outputs ON	EEZ BB3 STM32 EEZ BB3 Simulator	F2	SCPI	✓	✓	2
Outputs ON	EEZ H24005 r3B4	F2	SCPI	✓	✓	2

Fig. 4: Instruments Shortcuts and Groups view

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.

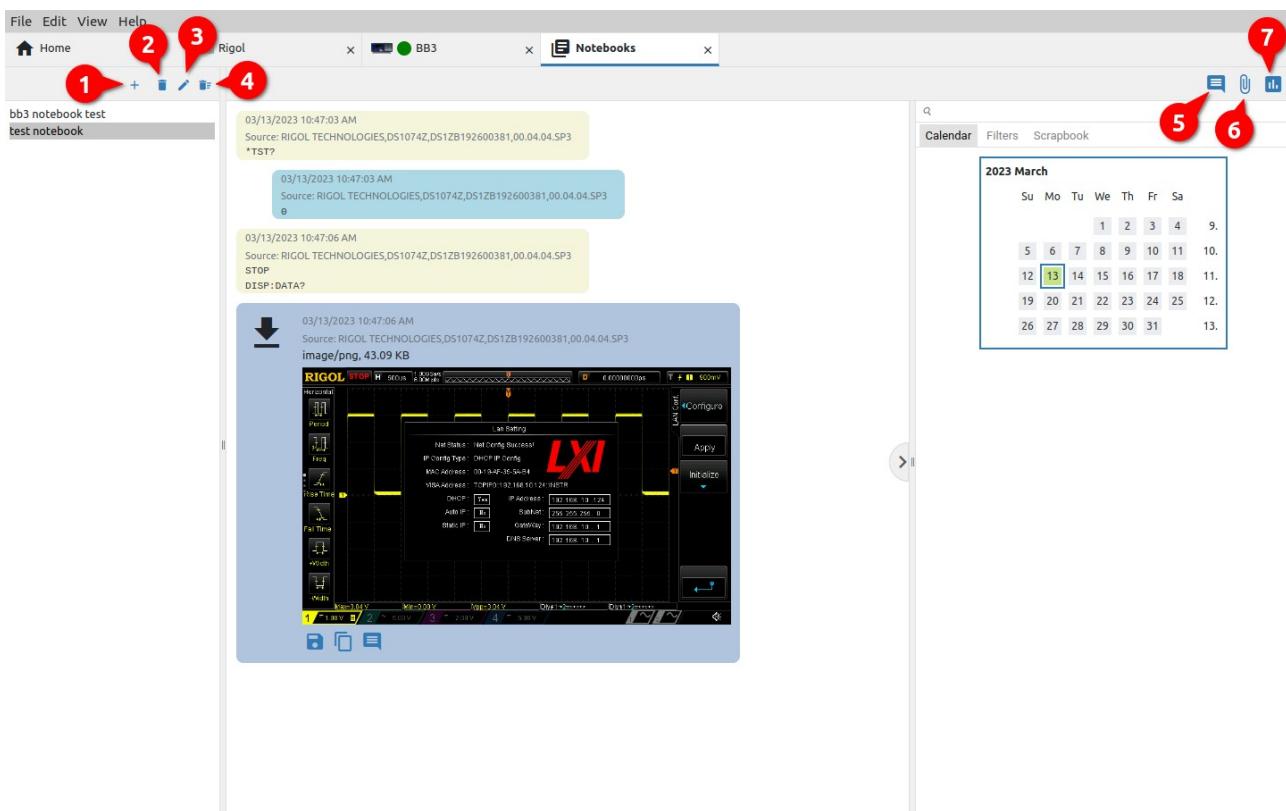


Fig. 5: Instrument Notebooks view

Option

1 Add / Import notebook

Description

Create a new blank notebook or import a notebook file. When creating a new notebook, you will need to enter a name. To import data into a notebook, use the Notebook option in the instrument's *Terminal*, as shown in Fig. 7: (1) go to the *Terminal* tab in the *Action bar*, (2) select one or more items and (3) export them to a notebook file, a new notebook or an already created notebook.

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

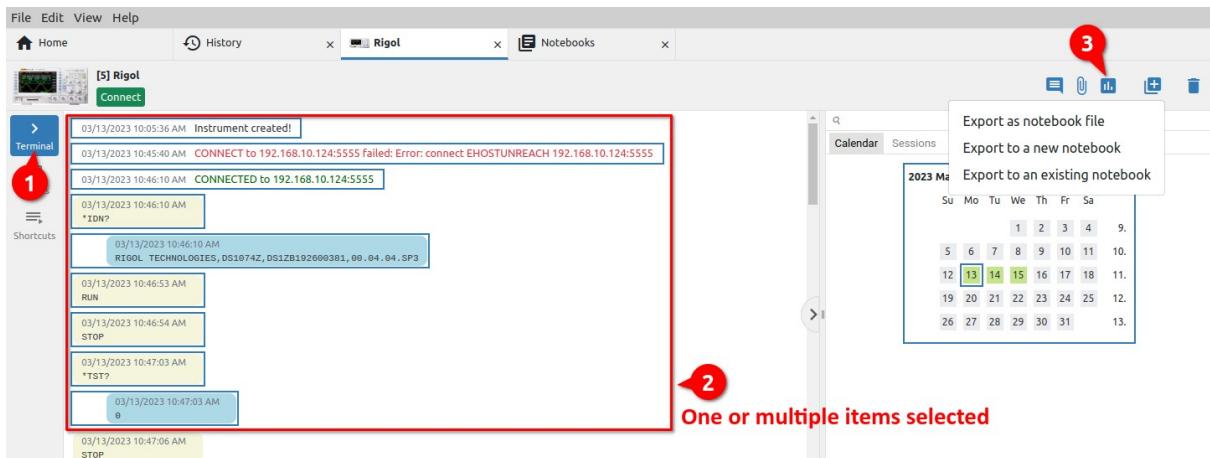


Fig. 6: Adding items to the notebook

2 Remove notebook

Remove the notebook from the list.

3 Change notebook name

Change notebook name.

4 Show deleted notebooks

Notebooks that have been removed from the list are not immediately deleted from the database. This option enables the display of all notebooks (Fig. 7) that have been removed from the list and offers the possibility to restore (return to the list) or permanently delete the notebook.

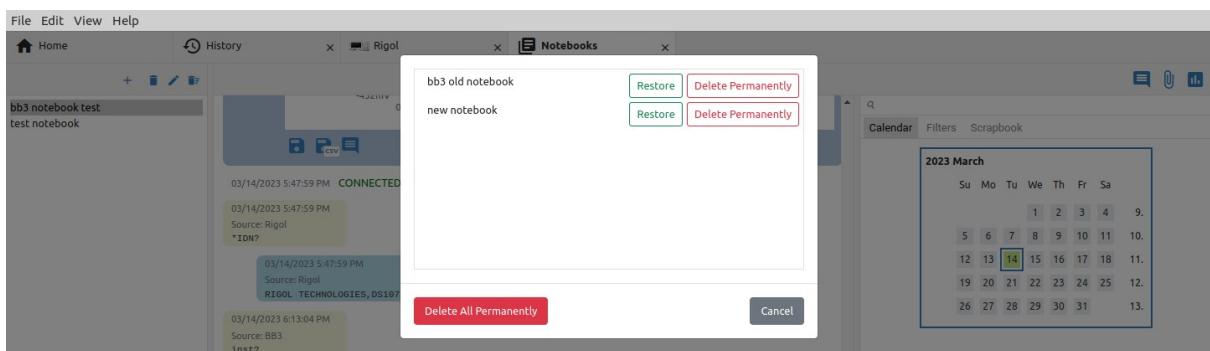


Fig. 7: Deleted notebooks

5 Add note

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

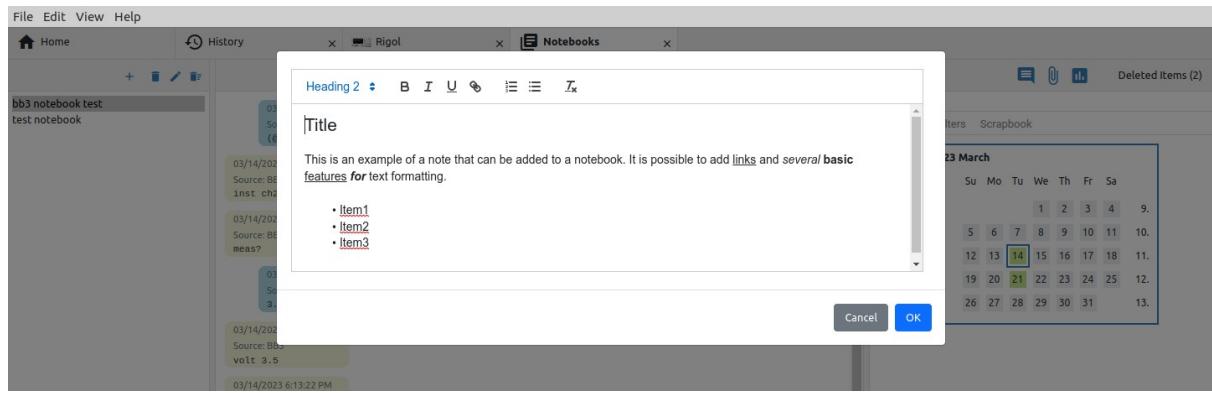


Fig. 8: Adding a new note to the notebook

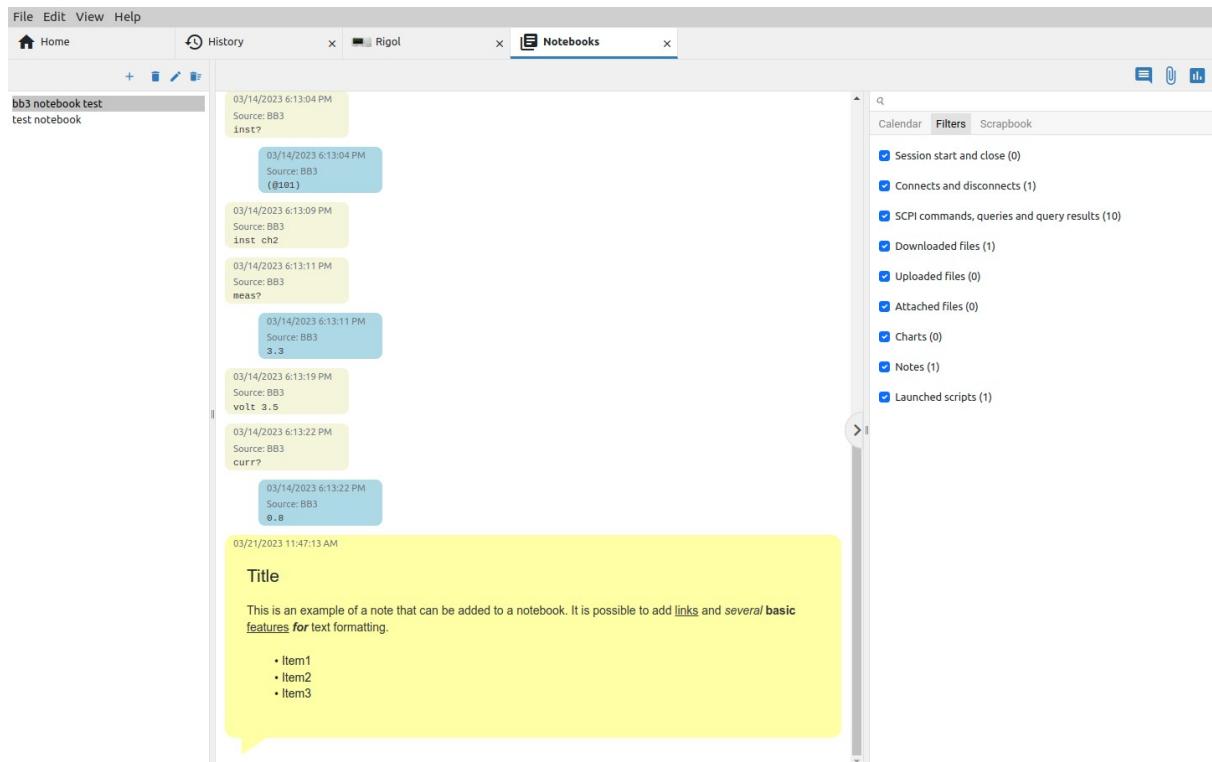


Fig. 9: Newly added note in the notebook

6 Attach file

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

All imported files are marked with a paper clip icon in the upper left corner. It also displays the full path from where the file was imported as well as its size (Fig. 10).

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

11. Home page instrument sections

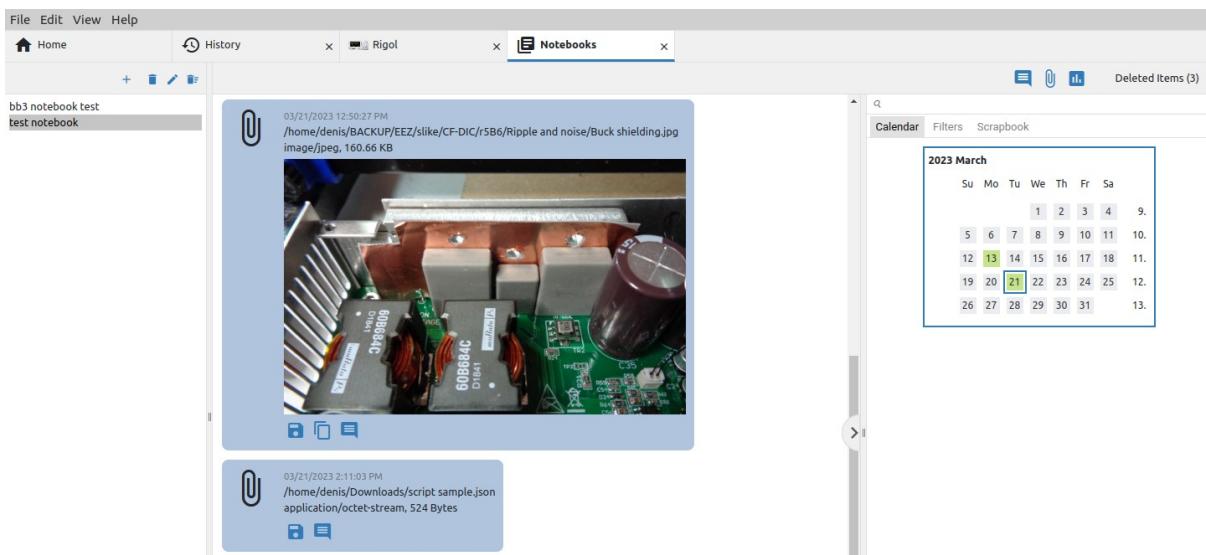


Fig. 10: Files imported into the notebook

7 Add chart

This option allows you to create a new graph from two or more existing ones and add it to the notebook. To create a new graph, you will need to select at least two of the found graphs in the currently selected notebook (1, 2) and add it to the notebook (3) as shown in Fig. 11.

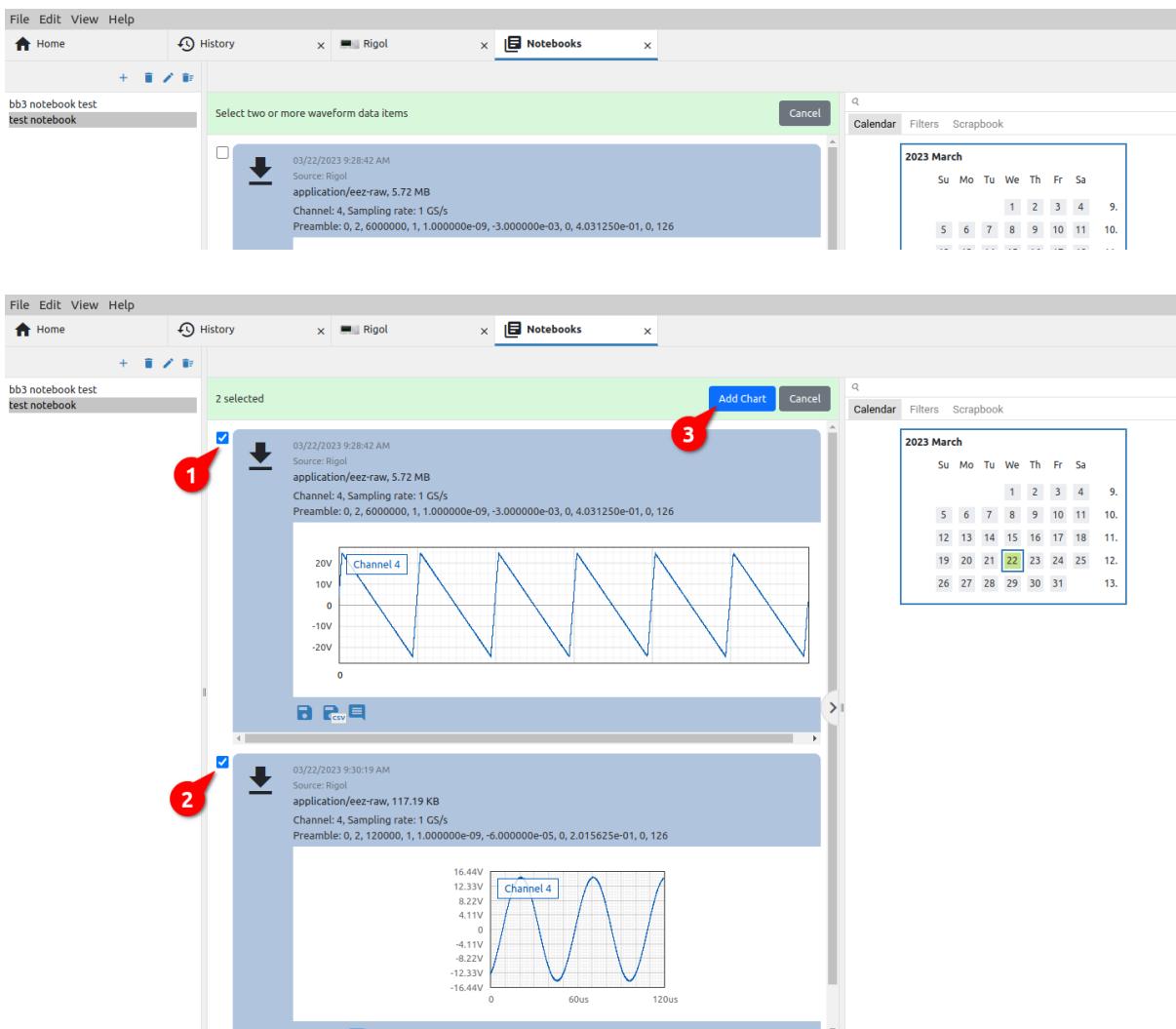


Fig. 11: A selection of graphs to add to the notebook

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

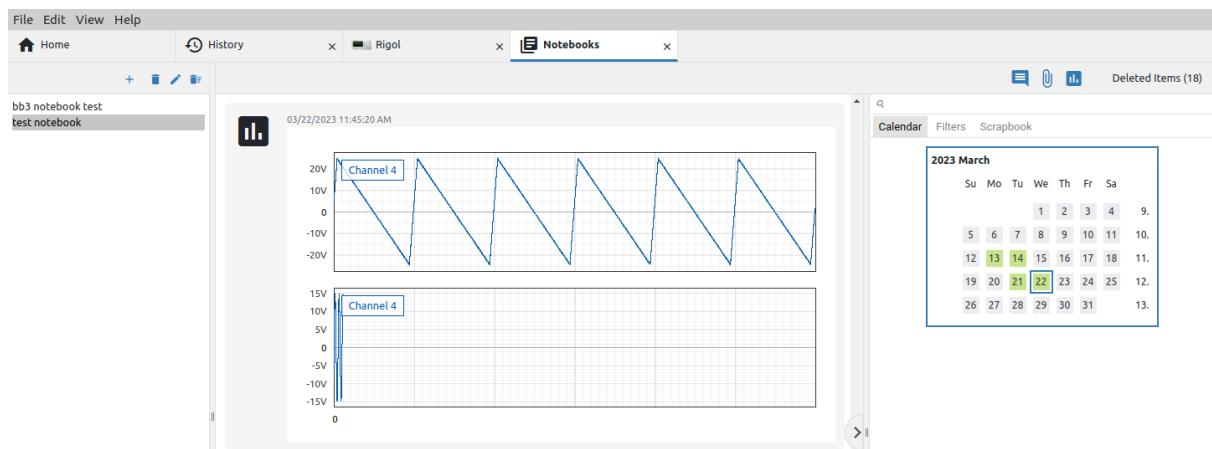


Fig. 12: Newly created graph added to notebook

I1.4. Items purge and restore

Items that are removed from the list are not immediately deleted from the database, which leaves the possibility to restore them if needed. The counter of deleted items that can be restored appears in the right corner as shown in Fig. 13.

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

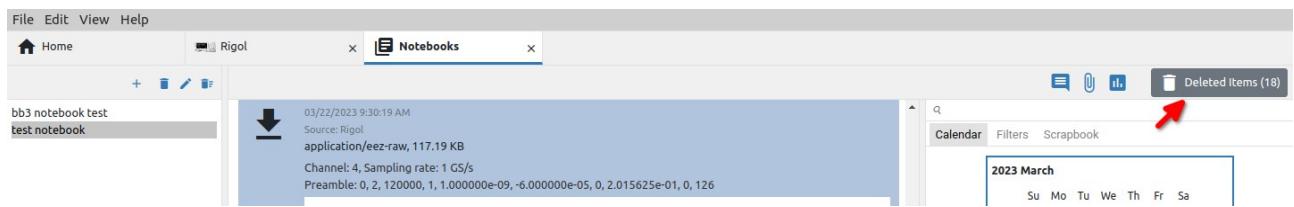


Fig. 13: Deleted items counter

When there are items to delete, they can be accessed by clicking on the counter, when the option to purge all items will first appear (Fig. 14).

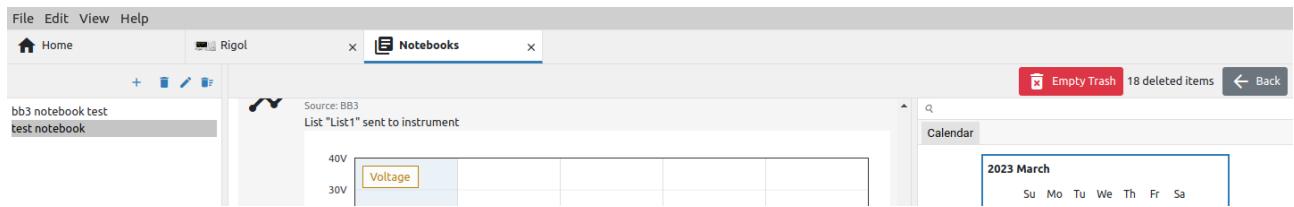


Fig. 14: Empty trash option (no selected items)

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

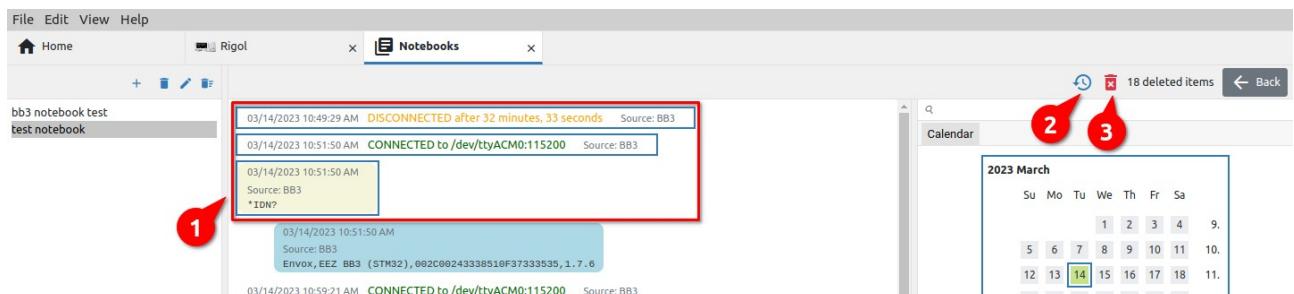


Fig. 15: Selection of deleted items for restore or purge

11.5. Instrument Extension (IEXT) Manager

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

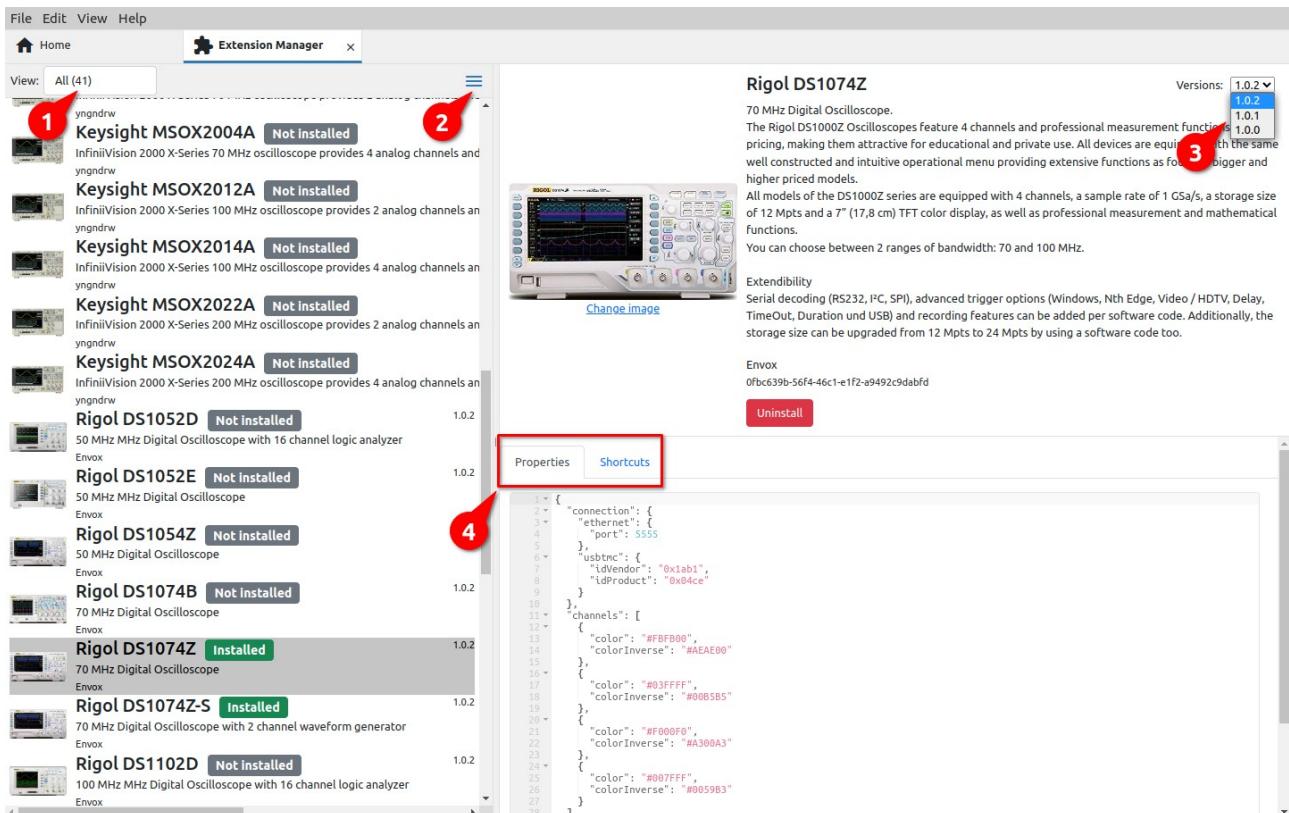


Fig. 16: Instrument extension (IEXT) Manager view

#	Option	Description
1	<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. 17.

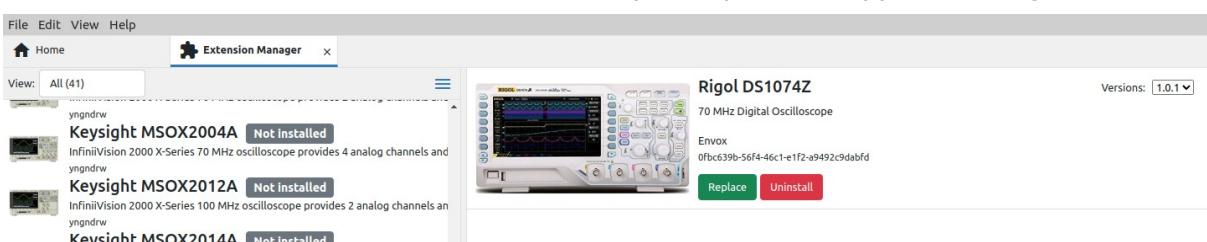


Fig. 17: Changing installed IEXT version

4 Properties

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

11.6. Add instrument

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

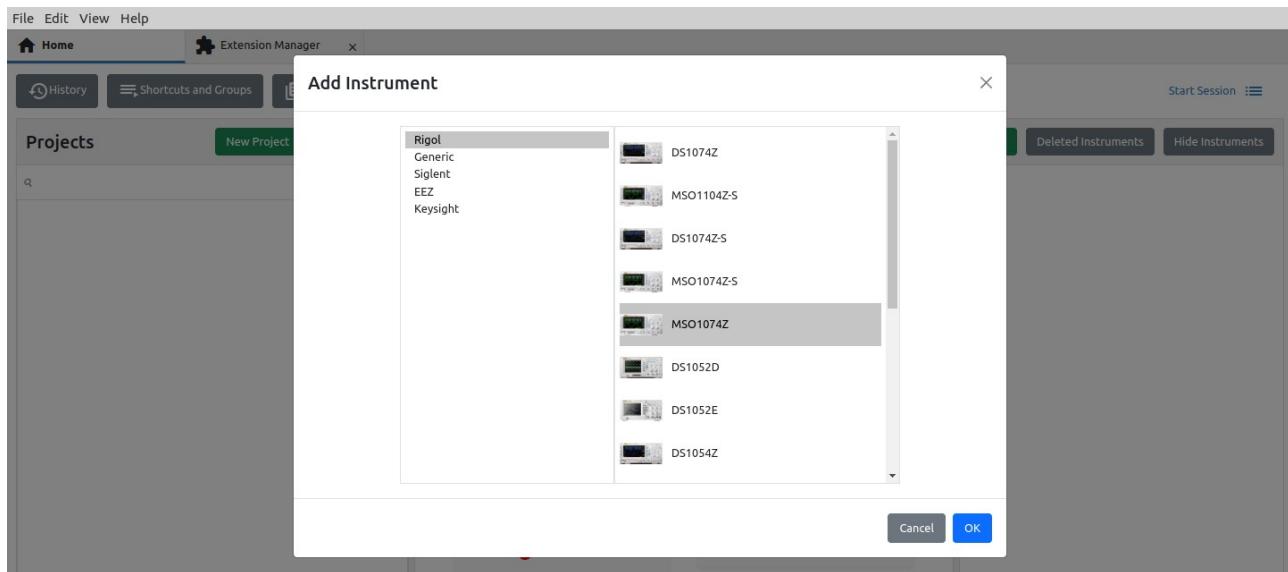


Fig. 18: Add instrument to workbench

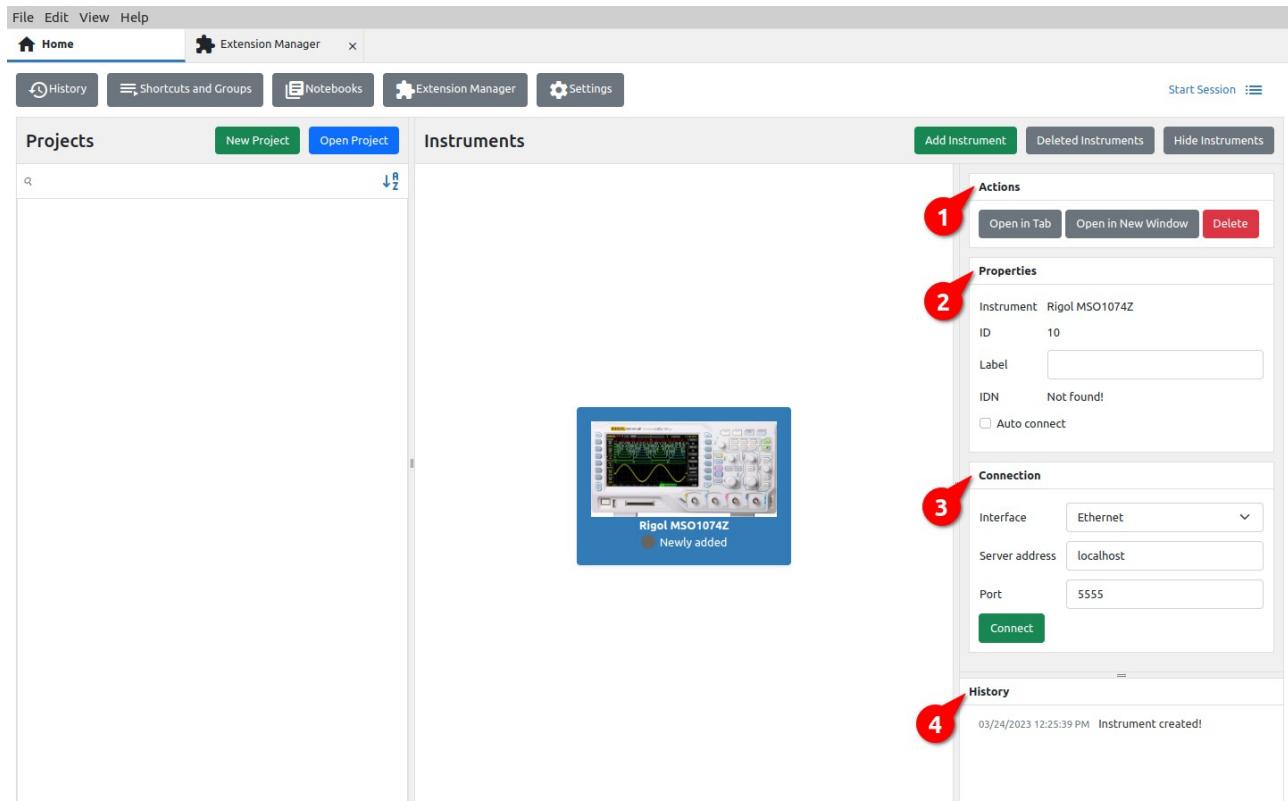


Fig. 19: New added instrument

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

#	Option	Description
1	<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. 20.

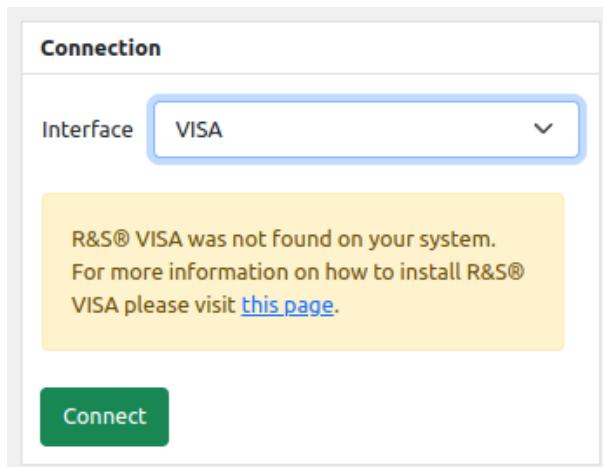


Fig. 20: VISA driver error message

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

I1.7. Establishing a connection with the instrument

Connection to the instrument added to the workbench will be possible as shown in Fig. 21: select the instrument from the workbench (1), select the interface in the Connection section (2) and click the Connect button (3).

If the Instrument tab (1) is open, as shown in Fig. 22 to establish a connection, it will be necessary to click on the *Connect* button (2) when a dialogue for choosing an interface will open in which the connection parameters are defined.

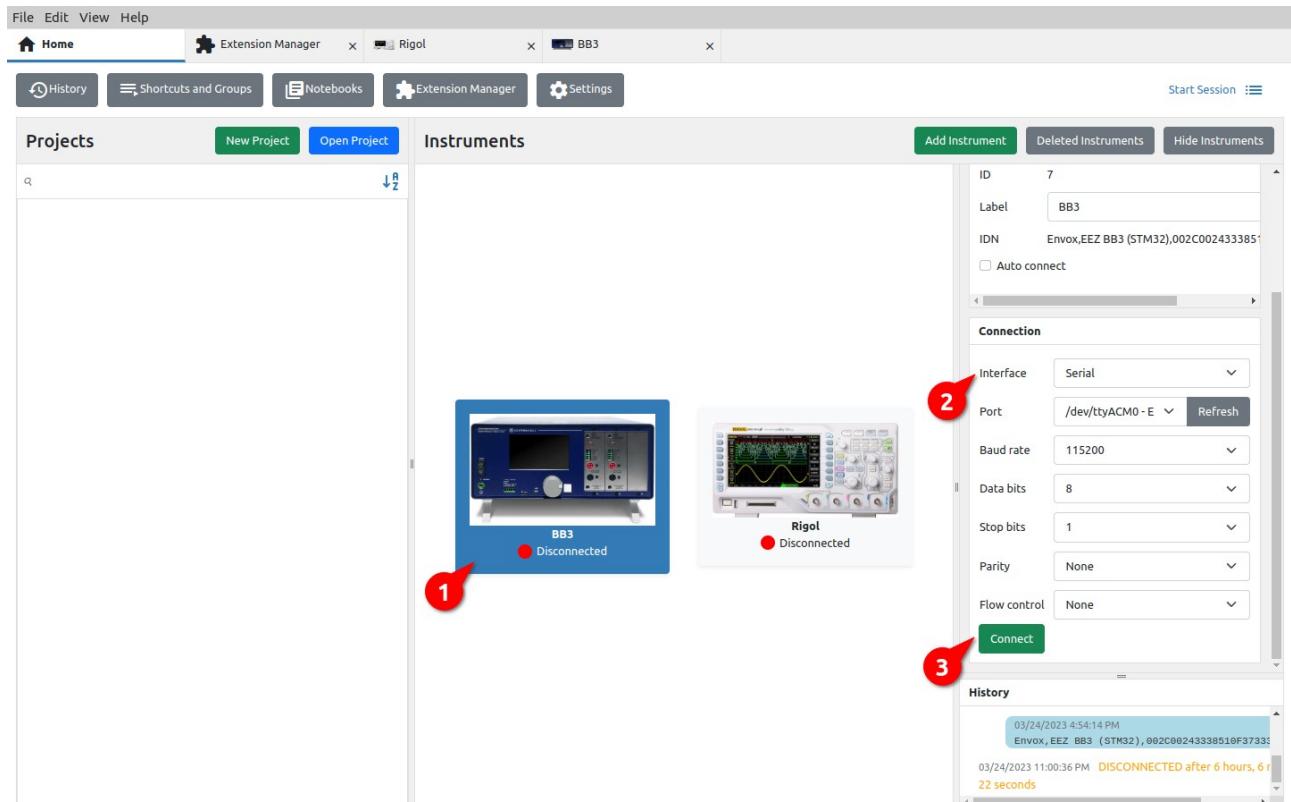


Fig. 21: Selecting an instrument on the workbench to establish a connection

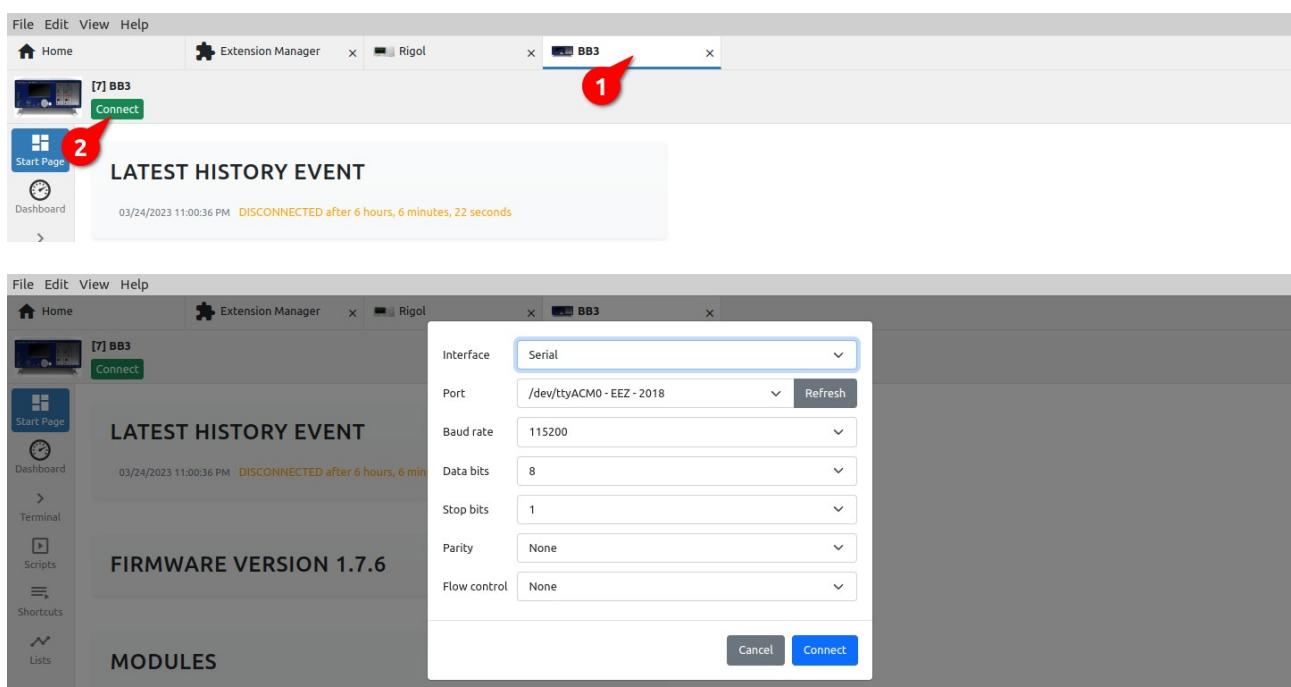


Fig. 22: Establishing a Connection from the Instrument tab

11. Home page instrument sections

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



Fig. 23: Option to close the connection

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)

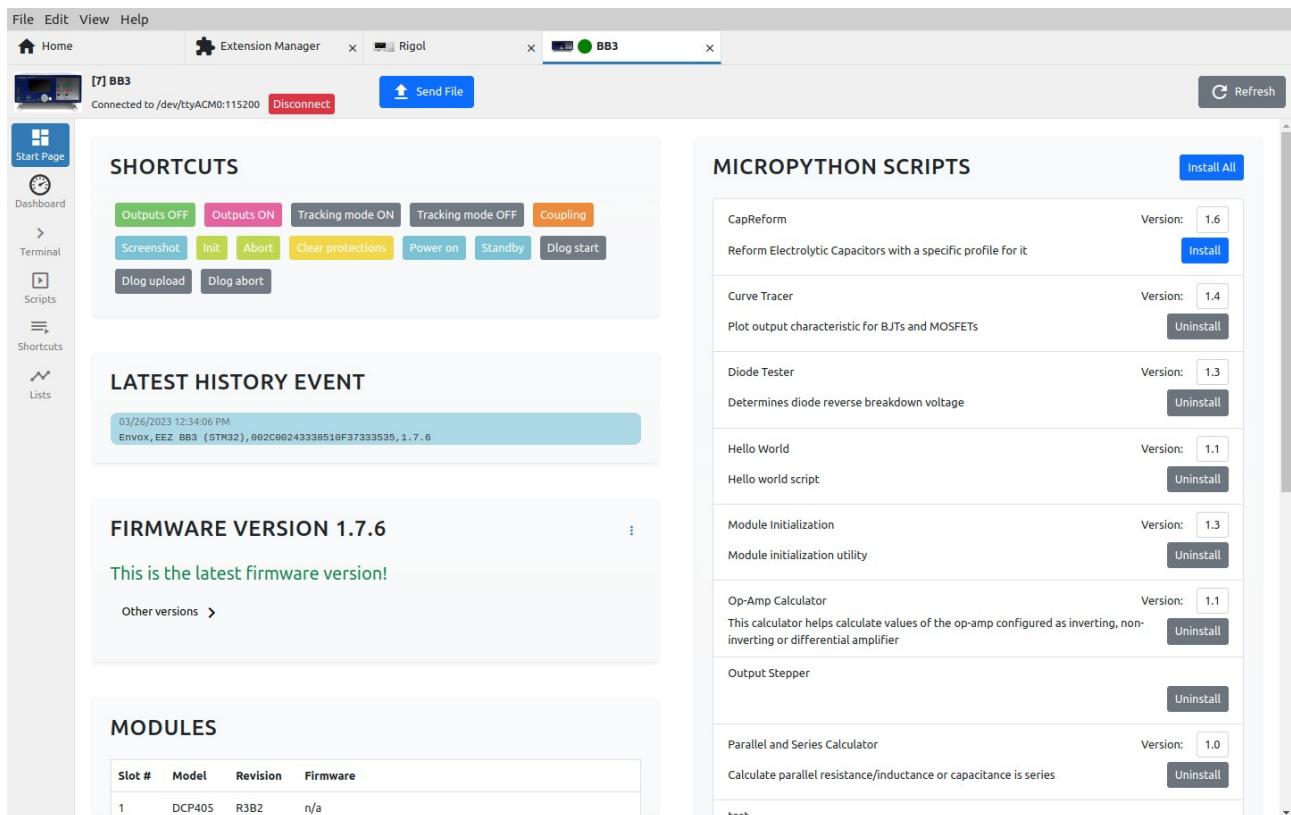


Fig. 24: EEZ BB3 start page

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.

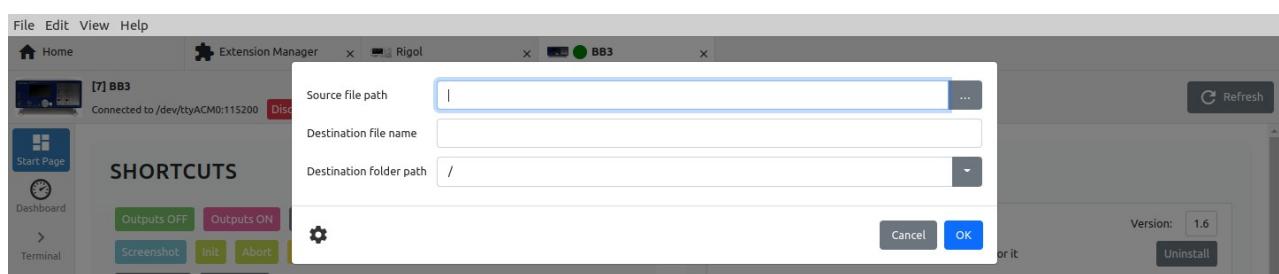


Fig. 25: Sending a file to EEZ BB3

Refresh

Refresh all data displayed on the *Start page*.

Shortcuts

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

Latest history event

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

Firmware version

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

grade option will be offered. It is also possible to manually install another version of the firmware (1) or downgrade the version from the offered list (2) as shown in Fig. 26.

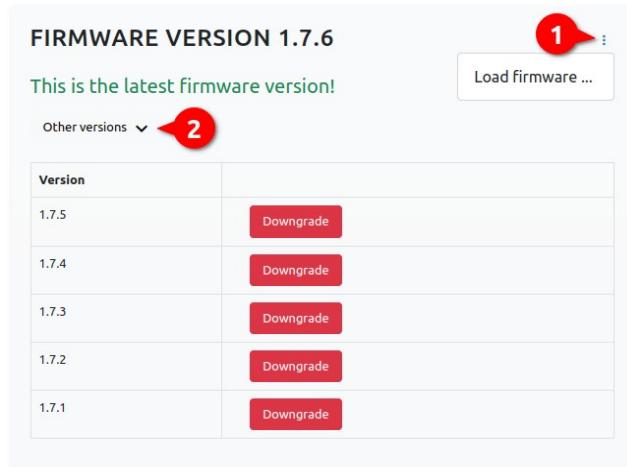


Fig. 26: EEZ BB3 firmware version section

Modules

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

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

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

[Other versions >](#)

[Upload Pinout Pages](#)

Fig. 27: EEZ BB3 modules section

Micropython scripts

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

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

Lists

Program lists created by the user (see Section I2.4.). and which are located on EEZ BB3. Lists can be downloaded, uploaded and edited.

LISTS		Download All	Upload All
3.3V stress test		03/14/2023 10:51:38 AM	Download Upload Edit
List1		03/15/2023 5:08:38 PM	Upload Edit
Test list 1			
New list		03/15/2023 2:33:07 PM	Upload Edit
Another sample			

Fig. 28: EEZ BB3 program lists

12.2. Dashboard

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

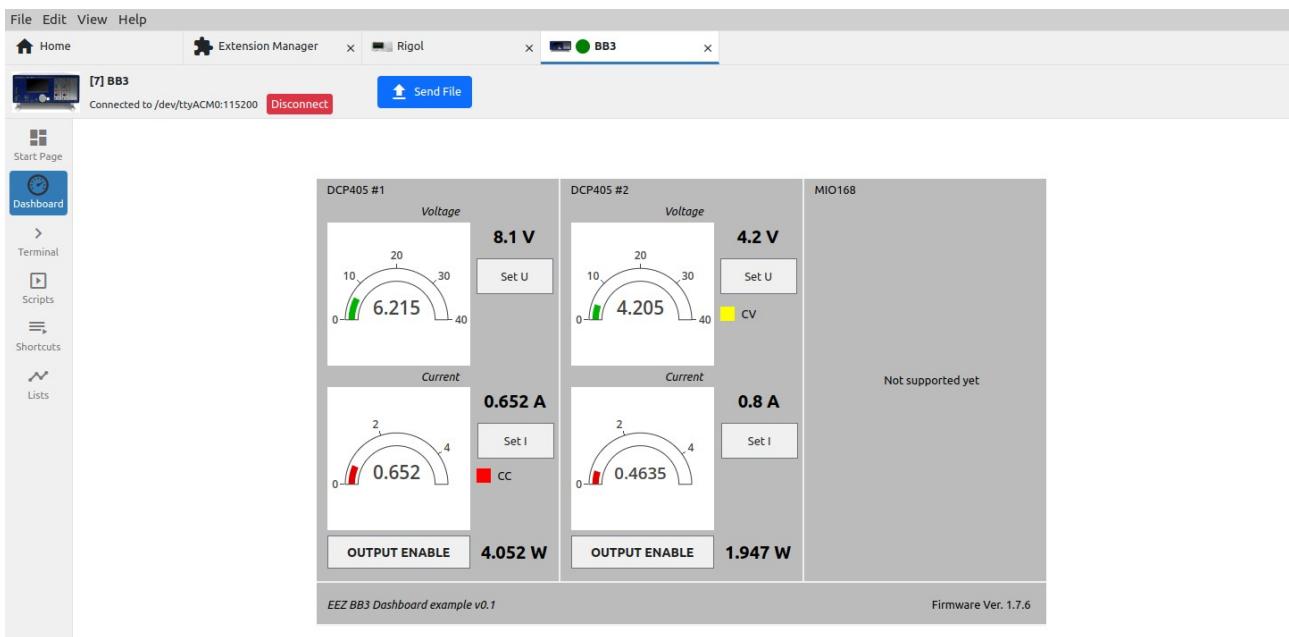


Fig. 29: Instrument dashboard example

12.3. Terminal

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

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

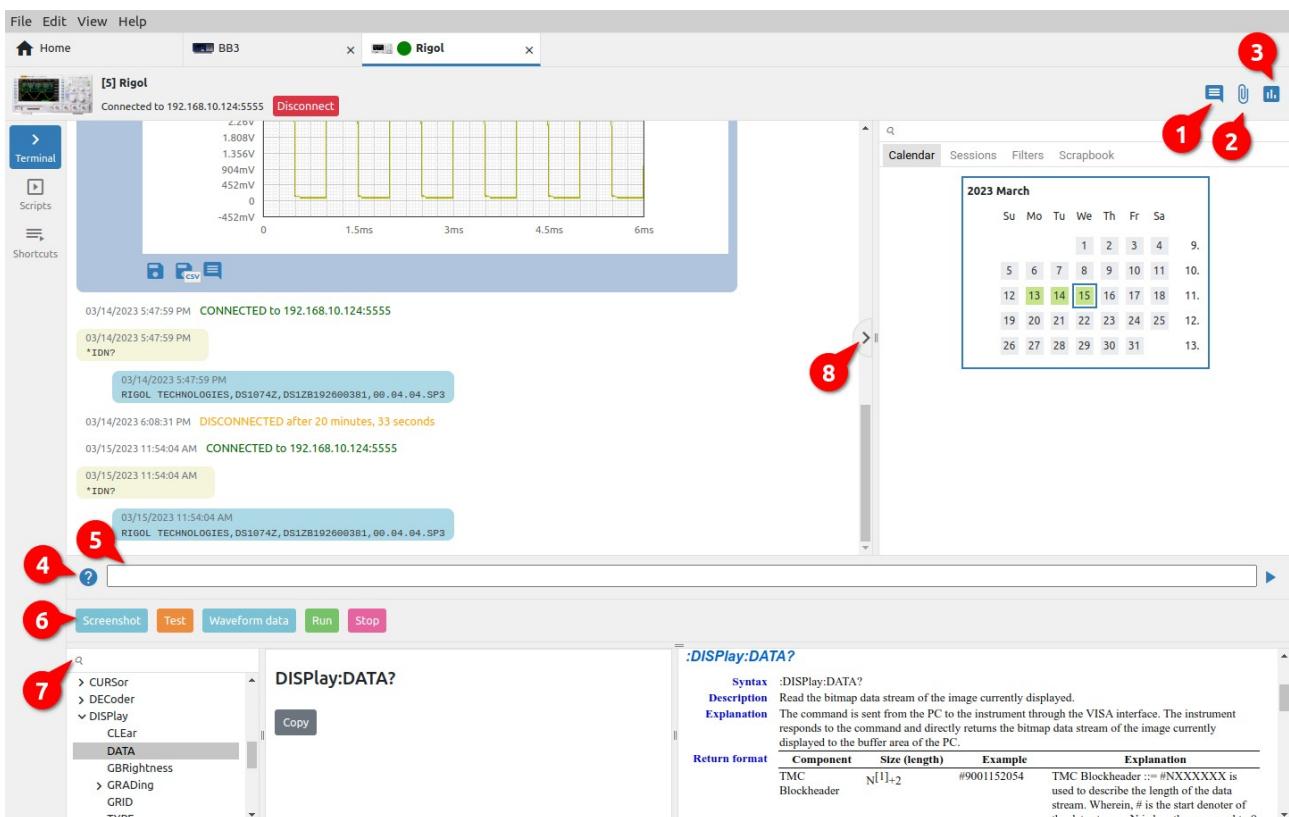


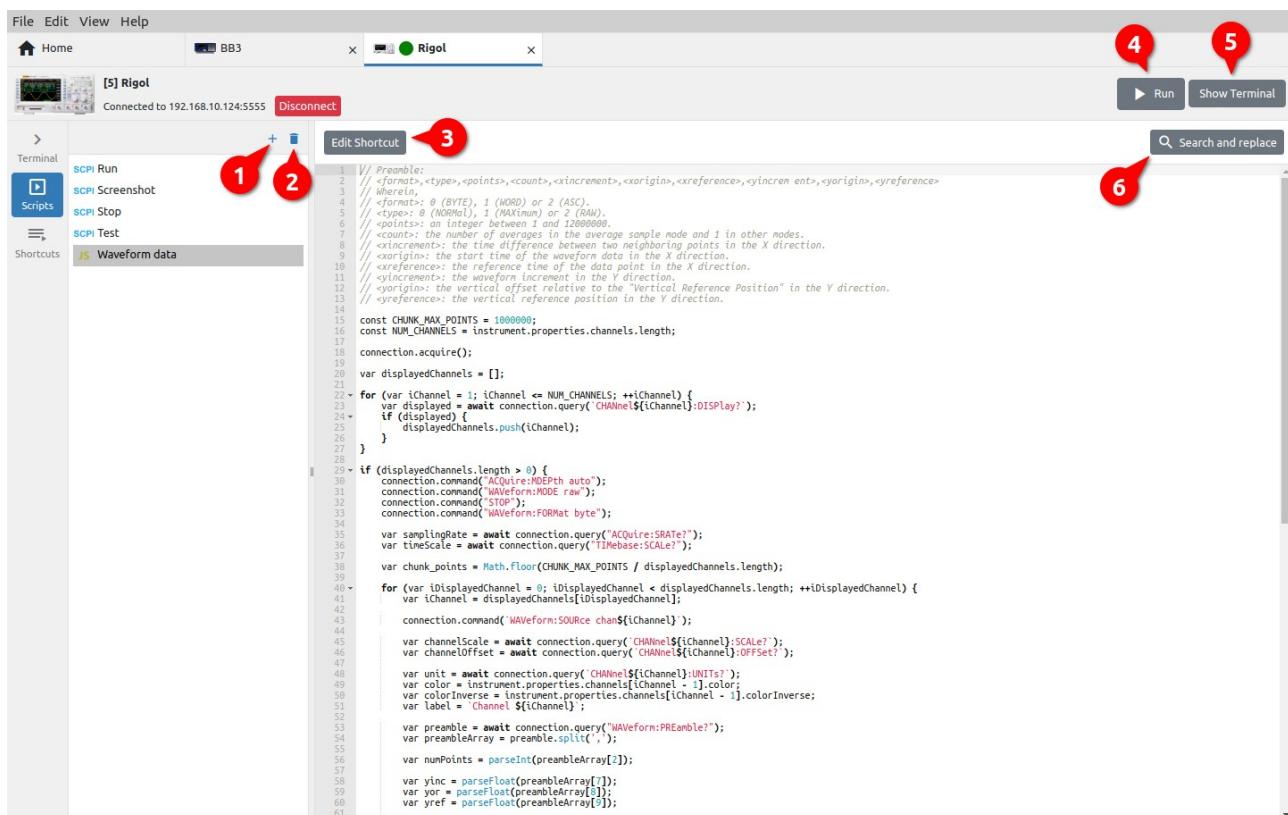
Fig. 30: Instrument terminal

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

12.4. Scripts

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

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



The screenshot shows the EEZ Studio interface with the following annotations:

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

```

File Edit View Help
Home BB3 Rigol [5] Rigol Connected to 192.168.10.124:5555 Disconnect Run Show Terminal
Search and replace

1 Scripts
2 + 3 Edit Shortcut
4 5
6

Terminal
SCPI Run
SCPI Screenshot
SCPI Stop
SCPI Test
US Waveform data

Shortcuts
1 2 3 4 5 6

```

```

1 // Preamble:
2 // <format>, <type>, <points>, <count>, <xincrement>, <xorigin>, <xreference>, <yincrement>, <yorigin>, <yreference>
3 // <format>: 0 (BYTE), 1 (WORD) or 2 (ASC).
4 // <type>: 0 (NORMAL), 1 (MAXIMUM) or 2 (RAW).
5 // <points>: an integer between 1 and 12800000.
6 // <count>: the number of data points to average sample mode and 1 in other modes.
7 // <xincrement>: the time difference between two neighboring points in the X direction.
8 // <xincrement>: the start time of the waveform data in the X direction.
9 // <xorigin>: the reference time of the data point in the X direction.
10 // <xreference>: the waveform increment in the X direction.
11 // <yincrement>: the vertical offset relative to the "Vertical Reference Position" in the Y direction.
12 // <yreference>: the vertical reference position in the Y direction.
13
14 const CHUNK_MAX_POINTS = 1000000;
15 const NUM_CHANNELS = instrument.properties.channels.length;
16
17 connection.acquire();
18
19 var displayedChannels = [];
20
21 for (var iChannel = 1; iChannel <= NUM_CHANNELS; ++iChannel) {
22   var displayed = await connection.query(`CHANnel${iChannel}:DISPlay?`);
23   if (displayed) {
24     displayedChannels.push(iChannel);
25   }
26 }
27
28 if (displayedChannels.length > 0) {
29   connection.command("ACQuire:MODe auto");
30   connection.command("WAveform:MODe raw");
31   connection.command("STOP");
32   connection.command("WAveform:FORMat byte");
33
34 var samplingRate = await connection.query("ACQuire:SRATE?");
35 var timeScale = await connection.query("TImebase:SCALE?");
36
37 var chunk_points = Math.floor(CHUNK_MAX_POINTS / displayedChannels.length);
38
39 for (var iDisplayedChannel = 0; iDisplayedChannel < displayedChannels.length; ++iDisplayedChannel) {
40   var iChannel = displayedChannels[iDisplayedChannel];
41
42   connection.command(`WAveform:SOURce chAnel${iChannel}`);
43
44   var channelscale = await connection.query(`CHANnel${iChannel}:SCALE?`);
45   var channelOffset = await connection.query(`CHANnel${iChannel}:OFFSet?`);
46
47   var unit = await connection.query(`CHANnel${iChannel}:UNITS?`);
48   var color = instrument.palette.channels[iChannel - 1].color;
49   var colorInverse = instrument.properties.channels[iChannel - 1].colorInverse;
50   var label = `CHANnel ${iChannel}`;
51
52   var preamble = await connection.query("WAveform:PREamble");
53   var preambleArray = preamble.split(",");
54
55   var numPoints = parseInt(preambleArray[0]);
56
57   var yinc = parseFloat(preambleArray[1]);
58   var yor = parseFloat(preambleArray[2]);
59   var yref = parseFloat(preambleArray[3]);
60
61   if (unit === "V") {
62     yinc *= 1000000;
63     yor *= 1000000;
64     yref *= 1000000;
65   }
66
67   if (color === "red") {
68     color = "#ff0000";
69   } else if (color === "blue") {
70     color = "#0000ff";
71   } else if (color === "green") {
72     color = "#00ff00";
73   } else if (color === "yellow") {
74     color = "#ffff00";
75   } else if (color === "cyan") {
76     color = "#00ffff";
77   } else if (color === "magenta") {
78     color = "#ff00ff";
79   }
80
81   if (label === "CHANnel 1") {
82     label = "CH1";
83   } else if (label === "CHANnel 2") {
84     label = "CH2";
85   } else if (label === "CHANnel 3") {
86     label = "CH3";
87   } else if (label === "CHANnel 4") {
88     label = "CH4";
89   }
90
91   if (yinc < 0) {
92     yinc = -yinc;
93   }
94
95   if (yor < 0) {
96     yor = -yor;
97   }
98
99   if (yref < 0) {
100    yref = -yref;
101  }
102
103   if (color === "red") {
104     color = "#ff0000";
105   } else if (color === "blue") {
106     color = "#0000ff";
107   } else if (color === "green") {
108     color = "#00ff00";
109   } else if (color === "yellow") {
110     color = "#ffff00";
111   } else if (color === "cyan") {
112     color = "#00ffff";
113   } else if (color === "magenta") {
114     color = "#ff00ff";
115   }
116
117   if (label === "CHANnel 1") {
118     label = "CH1";
119   } else if (label === "CHANnel 2") {
120     label = "CH2";
121   } else if (label === "CHANnel 3") {
122     label = "CH3";
123   } else if (label === "CHANnel 4") {
124     label = "CH4";
125   }
126
127   if (yinc < 0) {
128     yinc = -yinc;
129   }
130
131   if (yor < 0) {
132     yor = -yor;
133   }
134
135   if (yref < 0) {
136     yref = -yref;
137   }
138
139   if (color === "red") {
140     color = "#ff0000";
141   } else if (color === "blue") {
142     color = "#0000ff";
143   } else if (color === "green") {
144     color = "#00ff00";
145   } else if (color === "yellow") {
146     color = "#ffff00";
147   } else if (color === "cyan") {
148     color = "#00ffff";
149   } else if (color === "magenta") {
150     color = "#ff00ff";
151   }
152
153   if (label === "CHANnel 1") {
154     label = "CH1";
155   } else if (label === "CHANnel 2") {
156     label = "CH2";
157   } else if (label === "CHANnel 3") {
158     label = "CH3";
159   } else if (label === "CHANnel 4") {
160     label = "CH4";
161   }
162
163   if (yinc < 0) {
164     yinc = -yinc;
165   }
166
167   if (yor < 0) {
168     yor = -yor;
169   }
170
171   if (yref < 0) {
172     yref = -yref;
173   }
174
175   if (color === "red") {
176     color = "#ff0000";
177   } else if (color === "blue") {
178     color = "#0000ff";
179   } else if (color === "green") {
180     color = "#00ff00";
181   } else if (color === "yellow") {
182     color = "#ffff00";
183   } else if (color === "cyan") {
184     color = "#00ffff";
185   } else if (color === "magenta") {
186     color = "#ff00ff";
187   }
188
189   if (label === "CHANnel 1") {
190     label = "CH1";
191   } else if (label === "CHANnel 2") {
192     label = "CH2";
193   } else if (label === "CHANnel 3") {
194     label = "CH3";
195   } else if (label === "CHANnel 4") {
196     label = "CH4";
197   }
198
199   if (yinc < 0) {
200     yinc = -yinc;
201   }
202
203   if (yor < 0) {
204     yor = -yor;
205   }
206
207   if (yref < 0) {
208     yref = -yref;
209   }
210
211   if (color === "red") {
212     color = "#ff0000";
213   } else if (color === "blue") {
214     color = "#0000ff";
215   } else if (color === "green") {
216     color = "#00ff00";
217   } else if (color === "yellow") {
218     color = "#ffff00";
219   } else if (color === "cyan") {
220     color = "#00ffff";
221   } else if (color === "magenta") {
222     color = "#ff00ff";
223   }
224
225   if (label === "CHANnel 1") {
226     label = "CH1";
227   } else if (label === "CHANnel 2") {
228     label = "CH2";
229   } else if (label === "CHANnel 3") {
230     label = "CH3";
231   } else if (label === "CHANnel 4") {
232     label = "CH4";
233   }
234
235   if (yinc < 0) {
236     yinc = -yinc;
237   }
238
239   if (yor < 0) {
240     yor = -yor;
241   }
242
243   if (yref < 0) {
244     yref = -yref;
245   }
246
247   if (color === "red") {
248     color = "#ff0000";
249   } else if (color === "blue") {
250     color = "#0000ff";
251   } else if (color === "green") {
252     color = "#00ff00";
253   } else if (color === "yellow") {
254     color = "#ffff00";
255   } else if (color === "cyan") {
256     color = "#00ffff";
257   } else if (color === "magenta") {
258     color = "#ff00ff";
259   }
260
261   if (label === "CHANnel 1") {
262     label = "CH1";
263   } else if (label === "CHANnel 2") {
264     label = "CH2";
265   } else if (label === "CHANnel 3") {
266     label = "CH3";
267   } else if (label === "CHANnel 4") {
268     label = "CH4";
269   }
270
271   if (yinc < 0) {
272     yinc = -yinc;
273   }
274
275   if (yor < 0) {
276     yor = -yor;
277   }
278
279   if (yref < 0) {
280     yref = -yref;
281   }
282
283   if (color === "red") {
284     color = "#ff0000";
285   } else if (color === "blue") {
286     color = "#0000ff";
287   } else if (color === "green") {
288     color = "#00ff00";
289   } else if (color === "yellow") {
290     color = "#ffff00";
291   } else if (color === "cyan") {
292     color = "#00ffff";
293   } else if (color === "magenta") {
294     color = "#ff00ff";
295   }
296
297   if (label === "CHANnel 1") {
298     label = "CH1";
299   } else if (label === "CHANnel 2") {
300     label = "CH2";
301   } else if (label === "CHANnel 3") {
302     label = "CH3";
303   } else if (label === "CHANnel 4") {
304     label = "CH4";
305   }
306
307   if (yinc < 0) {
308     yinc = -yinc;
309   }
310
311   if (yor < 0) {
312     yor = -yor;
313   }
314
315   if (yref < 0) {
316     yref = -yref;
317   }
318
319   if (color === "red") {
320     color = "#ff0000";
321   } else if (color === "blue") {
322     color = "#0000ff";
323   } else if (color === "green") {
324     color = "#00ff00";
325   } else if (color === "yellow") {
326     color = "#ffff00";
327   } else if (color === "cyan") {
328     color = "#00ffff";
329   } else if (color === "magenta") {
330     color = "#ff00ff";
331   }
332
333   if (label === "CHANnel 1") {
334     label = "CH1";
335   } else if (label === "CHANnel 2") {
336     label = "CH2";
337   } else if (label === "CHANnel 3") {
338     label = "CH3";
339   } else if (label === "CHANnel 4") {
340     label = "CH4";
341   }
342
343   if (yinc < 0) {
344     yinc = -yinc;
345   }
346
347   if (yor < 0) {
348     yor = -yor;
349   }
350
351   if (yref < 0) {
352     yref = -yref;
353   }
354
355   if (color === "red") {
356     color = "#ff0000";
357   } else if (color === "blue") {
358     color = "#0000ff";
359   } else if (color === "green") {
360     color = "#00ff00";
361   } else if (color === "yellow") {
362     color = "#ffff00";
363   } else if (color === "cyan") {
364     color = "#00ffff";
365   } else if (color === "magenta") {
366     color = "#ff00ff";
367   }
368
369   if (label === "CHANnel 1") {
370     label = "CH1";
371   } else if (label === "CHANnel 2") {
372     label = "CH2";
373   } else if (label === "CHANnel 3") {
374     label = "CH3";
375   } else if (label === "CHANnel 4") {
376     label = "CH4";
377   }
378
379   if (yinc < 0) {
380     yinc = -yinc;
381   }
382
383   if (yor < 0) {
384     yor = -yor;
385   }
386
387   if (yref < 0) {
388     yref = -yref;
389   }
390
391   if (color === "red") {
392     color = "#ff0000";
393   } else if (color === "blue") {
394     color = "#0000ff";
395   } else if (color === "green") {
396     color = "#00ff00";
397   } else if (color === "yellow") {
398     color = "#ffff00";
399   } else if (color === "cyan") {
400     color = "#00ffff";
401   } else if (color === "magenta") {
402     color = "#ff00ff";
403   }
404
405   if (label === "CHANnel 1") {
406     label = "CH1";
407   } else if (label === "CHANnel 2") {
408     label = "CH2";
409   } else if (label === "CHANnel 3") {
410     label = "CH3";
411   } else if (label === "CHANnel 4") {
412     label = "CH4";
413   }
414
415   if (yinc < 0) {
416     yinc = -yinc;
417   }
418
419   if (yor < 0) {
420     yor = -yor;
421   }
422
423   if (yref < 0) {
424     yref = -yref;
425   }
426
427   if (color === "red") {
428     color = "#ff0000";
429   } else if (color === "blue") {
430     color = "#0000ff";
431   } else if (color === "green") {
432     color = "#00ff00";
433   } else if (color === "yellow") {
434     color = "#ffff00";
435   } else if (color === "cyan") {
436     color = "#00ffff";
437   } else if (color === "magenta") {
438     color = "#ff00ff";
439   }
440
441   if (label === "CHANnel 1") {
442     label = "CH1";
443   } else if (label === "CHANnel 2") {
444     label = "CH2";
445   } else if (label === "CHANnel 3") {
446     label = "CH3";
447   } else if (label === "CHANnel 4") {
448     label = "CH4";
449   }
450
451   if (yinc < 0) {
452     yinc = -yinc;
453   }
454
455   if (yor < 0) {
456     yor = -yor;
457   }
458
459   if (yref < 0) {
460     yref = -yref;
461   }
462
463   if (color === "red") {
464     color = "#ff0000";
465   } else if (color === "blue") {
466     color = "#0000ff";
467   } else if (color === "green") {
468     color = "#00ff00";
469   } else if (color === "yellow") {
470     color = "#ffff00";
471   } else if (color === "cyan") {
472     color = "#00ffff";
473   } else if (color === "magenta") {
474     color = "#ff00ff";
475   }
476
477   if (label === "CHANnel 1") {
478     label = "CH1";
479   } else if (label === "CHANnel 2") {
480     label = "CH2";
481   } else if (label === "CHANnel 3") {
482     label = "CH3";
483   } else if (label === "CHANnel 4") {
484     label = "CH4";
485   }
486
487   if (yinc < 0) {
488     yinc = -yinc;
489   }
490
491   if (yor < 0) {
492     yor = -yor;
493   }
494
495   if (yref < 0) {
496     yref = -yref;
497   }
498
499   if (color === "red") {
500     color = "#ff0000";
501   } else if (color === "blue") {
502     color = "#0000ff";
503   } else if (color === "green") {
504     color = "#00ff00";
505   } else if (color === "yellow") {
506     color = "#ffff00";
507   } else if (color === "cyan") {
508     color = "#00ffff";
509   } else if (color === "magenta") {
510     color = "#ff00ff";
511   }
512
513   if (label === "CHANnel 1") {
514     label = "CH1";
515   } else if (label === "CHANnel 2") {
516     label = "CH2";
517   } else if (label === "CHANnel 3") {
518     label = "CH3";
519   } else if (label === "CHANnel 4") {
520     label = "CH4";
521   }
522
523   if (yinc < 0) {
524     yinc = -yinc;
525   }
526
527   if (yor < 0) {
528     yor = -yor;
529   }
530
531   if (yref < 0) {
532     yref = -yref;
533   }
534
535   if (color === "red") {
536     color = "#ff0000";
537   } else if (color === "blue") {
538     color = "#0000ff";
539   } else if (color === "green") {
540     color = "#00ff00";
541   } else if (color === "yellow") {
542     color = "#ffff00";
543   } else if (color === "cyan") {
544     color = "#00ffff";
545   } else if (color === "magenta") {
546     color = "#ff00ff";
547   }
548
549   if (label === "CHANnel 1") {
550     label = "CH1";
551   } else if (label === "CHANnel 2") {
552     label = "CH2";
553   } else if (label === "CHANnel 3") {
554     label = "CH3";
555   } else if (label === "CHANnel 4") {
556     label = "CH4";
557   }
558
559   if (yinc < 0) {
560     yinc = -yinc;
561   }
562
563   if (yor < 0) {
564     yor = -yor;
565   }
566
567   if (yref < 0) {
568     yref = -yref;
569   }
570
571   if (color === "red") {
572     color = "#ff0000";
573   } else if (color === "blue") {
574     color = "#0000ff";
575   } else if (color === "green") {
576     color = "#00ff00";
577   } else if (color === "yellow") {
578     color = "#ffff00";
579   } else if (color === "cyan") {
580     color = "#00ffff";
581   } else if (color === "magenta") {
582     color = "#ff00ff";
583   }
584
585   if (label === "CHANnel 1") {
586     label = "CH1";
587   } else if (label === "CHANnel 2") {
588     label = "CH2";
589   } else if (label === "CHANnel 3") {
590     label = "CH3";
591   } else if (label === "CHANnel 4") {
592     label = "CH4";
593   }
594
595   if (yinc < 0) {
596     yinc = -yinc;
597   }
598
599   if (yor < 0) {
600     yor = -yor;
601   }
602
603   if (yref < 0) {
604     yref = -yref;
605   }
606
607   if (color === "red") {
608     color = "#ff0000";
609   } else if (color === "blue") {
610     color = "#0000ff";
611   } else if (color === "green") {
612     color = "#00ff00";
613   } else if (color === "yellow") {
614     color = "#ffff00";
615   } else if (color === "cyan") {
616     color = "#00ffff";
617   } else if (color === "magenta") {
618     color = "#ff00ff";
619   }
620
621   if (label === "CHANnel 1") {
622     label = "CH1";
623   } else if (label === "CHANnel 2") {
624     label = "CH2";
625   } else if (label === "CHANnel 3") {
626     label = "CH3";
627   } else if (label === "CHANnel 4") {
628     label = "CH4";
629   }
630
631   if (yinc < 0) {
632     yinc = -yinc;
633   }
634
635   if (yor < 0) {
636     yor = -yor;
637   }
638
639   if (yref < 0) {
640     yref = -yref;
641   }
642
643   if (color === "red") {
644     color = "#ff0000";
645   } else if (color === "blue") {
646     color = "#0000ff";
647   } else if (color === "green") {
648     color = "#00ff00";
649   } else if (color === "yellow") {
650     color = "#ffff00";
651   } else if (color === "cyan") {
652     color = "#00ffff";
653   } else if (color === "magenta") {
654     color = "#ff00ff";
655   }
656
657   if (label === "CHANnel 1") {
658     label = "CH1";
659   } else if (label === "CHANnel 2") {
660     label = "CH2";
661   } else if (label === "CHANnel 3") {
662     label = "CH3";
663   } else if (label === "CHANnel 4") {
664     label = "CH4";
665   }
666
667   if (yinc < 0) {
668     yinc = -yinc;
669   }
670
671   if (yor < 0) {
672     yor = -yor;
673   }
674
675   if (yref < 0) {
676     yref = -yref;
677   }
678
679   if (color === "red") {
680     color = "#ff0000";
681   } else if (color === "blue") {
682     color = "#0000ff";
683   } else if (color === "green") {
684     color = "#00ff00";
685   } else if (color === "yellow") {
686     color = "#ffff00";
687   } else if (color === "cyan") {
688     color = "#00ffff";
689   } else if (color === "magenta") {
690     color = "#ff00ff";
691   }
692
693   if (label === "CHANnel 1") {
694     label = "CH1";
695   } else if (label === "CHANnel 2") {
696     label = "CH2";
697   } else if (label === "CHANnel 3") {
698     label = "CH3";
699   } else if (label === "CHANnel 4") {
700     label = "CH4";
701   }
702
703   if (yinc < 0) {
704     yinc = -yinc;
705   }
706
707   if (yor < 0) {
708     yor = -yor;
709   }
710
711   if (yref < 0) {
712     yref = -yref;
713   }
714
715   if (color === "red") {
716     color = "#ff0000";
717   } else if (color === "blue") {
718     color = "#0000ff";
719   } else if (color === "green") {
720     color = "#00ff00";
721   } else if (color === "yellow") {
722     color = "#ffff00";
723   } else if (color === "cyan") {
724     color = "#00ffff";
725   } else if (color === "magenta") {
726     color = "#ff00ff";
727   }
728
729   if (label === "CHANnel 1") {
730     label = "CH1";
731   } else if (label === "CHANnel 2") {
732     label = "CH2";
733   } else if (label === "CHANnel 3") {
734     label = "CH3";
735   } else if (label === "CHANnel 4") {
736     label = "CH4";
737   }
738
739   if (yinc < 0) {
740     yinc = -yinc;
741   }
742
743   if (yor < 0) {
744     yor = -yor;
745   }
746
747   if (yref < 0) {
748     yref = -yref;
749   }
750
751   if (color === "red") {
752     color = "#ff0000";
753   } else if (color === "blue") {
754     color = "#0000ff";
755   } else if (color === "green") {
756     color = "#00ff00";
757   } else if (color === "yellow") {
758     color = "#ffff00";
759   } else if (color === "cyan") {
760     color = "#00ffff";
761   } else if (color === "magenta") {
762     color = "#ff00ff";
763   }
764
765   if (label === "CHANnel 1") {
766     label = "CH1";
767   } else if (label === "CHANnel 2") {
768     label = "CH2";
769   } else if (label === "CHANnel 3") {
770     label = "CH3";
771   } else if (label === "CHANnel 4") {
772     label = "CH4";
773   }
774
775   if (yinc < 0) {
776     yinc = -yinc;
777   }
778
779   if (yor < 0) {
780     yor = -yor;
781   }
782
783   if (yref < 0) {
784     yref = -yref;
785   }
786
787   if (color === "red") {
788     color = "#ff0000";
789   } else if (color === "blue") {
790     color = "#0000ff";
791   } else if (color === "green") {
792     color = "#00ff00";
793   } else if (color === "yellow") {
794     color = "#ffff00";
795   } else if (color === "cyan") {
796     color = "#00ffff";
797   } else if (color === "magenta") {
798     color = "#ff00ff";
799   }
800
801   if (label === "CHANnel 1") {
802     label = "CH1";
803   } else if (label === "CHANnel 2") {
804     label = "CH2";
805   } else if (label === "CHANnel 3") {
806     label = "CH3";
807   } else if (label === "CHANnel 4") {
808     label = "CH4";
809   }
810
811   if (yinc < 0) {
812     yinc = -yinc;
813   }
814
815   if (yor < 0) {
816     yor = -yor;
817   }
818
819   if (yref < 0) {
820     yref = -yref;
821   }
822
823   if (color === "red") {
824     color = "#ff0000";
825   } else if (color === "blue") {
826     color = "#0000ff";
827   } else if (color === "green") {
828     color = "#00ff00";
829   } else if (color === "yellow") {
830     color = "#ffff00";
831   } else if (color === "cyan") {
832     color = "#00ffff";
833   } else if (color === "magenta") {
834     color = "#ff00ff";
835   }
836
837   if (label === "CHANnel 1") {
838     label = "CH1";
839   } else if (label === "CHANnel 2") {
840     label = "CH2";
841   } else if (label === "CHANnel 3") {
842     label = "CH3";
843   } else if (label === "CHANnel 4") {
844     label = "CH4";
845   }
846
847   if (yinc < 0) {
848     yinc = -yinc;
849   }
850
851   if (yor < 0) {
852     yor = -yor;
853   }
854
855   if (yref < 0) {
856     yref = -yref;
857   }
858
859   if (color === "red") {
860     color = "#ff0000";
861   } else if (color === "blue") {
862     color = "#0000ff";
863   } else if (color === "green") {
864     color = "#00ff00";
865   } else if (color === "yellow") {
866     color = "#ffff00";
867   } else if (color === "cyan") {
868     color = "#00ffff";
869   } else if (color === "magenta") {
870     color = "#ff00ff";
871   }
872
873   if (label === "CHANnel 1") {
874     label = "CH1";
875   } else if (label === "CHANnel 2") {
876     label = "CH2";
877   } else if (label === "CHANnel 3") {
878     label = "CH3";
879   } else if (label === "CHANnel 4") {
880     label = "CH4";
881   }
882
883   if (yinc < 0) {
884     yinc = -yinc;
885   }
886
887   if (yor < 0) {
888     yor = -yor;
889   }
890
891   if (yref < 0) {
892     yref = -yref;
893   }
894
895   if (color === "red") {
896     color = "#ff0000";
897   } else if (color === "blue") {
898     color = "#0000ff";
899   } else if (color === "green") {
900     color = "#00ff00";
901   } else if (color === "yellow") {
902     color = "#ffff00";
903   } else if (color === "cyan") {
904     color = "#00ffff";
905   } else if (color === "magenta") {
906     color = "#ff00ff";
907   }
908
909   if (label === "CHANnel 1") {
910     label = "CH1";
911   } else if (label === "CHANnel 2") {
912     label = "CH2";
913   } else if (label === "CHANnel 3") {
914     label = "CH3";
915   } else if (label === "CHANnel 4") {
916     label = "CH4";
917   }
918
919   if (yinc < 0) {
920     yinc = -yinc;
921   }
922
923   if (yor < 0) {
924     yor = -yor;
92
```

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

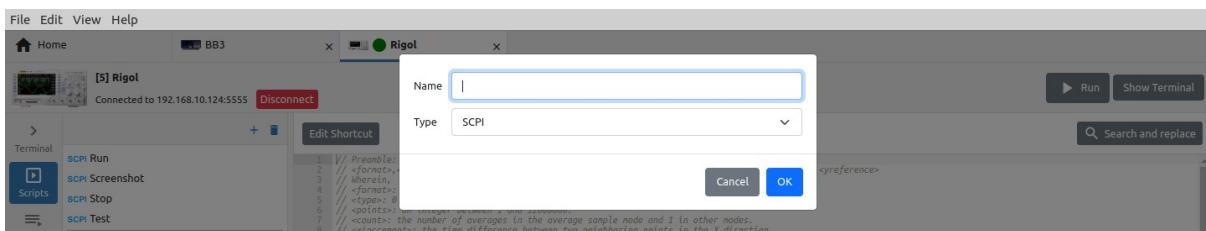


Fig. 32: Adding a new script

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

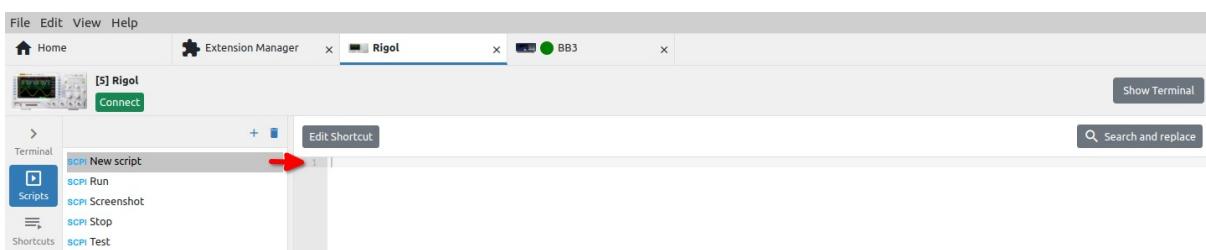


Fig. 33: Script editing

2	Delete script	Deleting the selected script.
3	Edit shortcut	Editing a script shortcut (see Section I2.4.1.)
4	Run	Runs the script on the instrument. This option is only displayed if the connection to the instrument is established.
5	Show / Hide terminal	Show / hide Terminal on the right.
6	Search and replace	Script editor function for searching and replacing text in the script. By default, only the search field is displayed. To replace the found text, it will be necessary to click on the "+" sign.

I2.4.1. Edit script shortcut

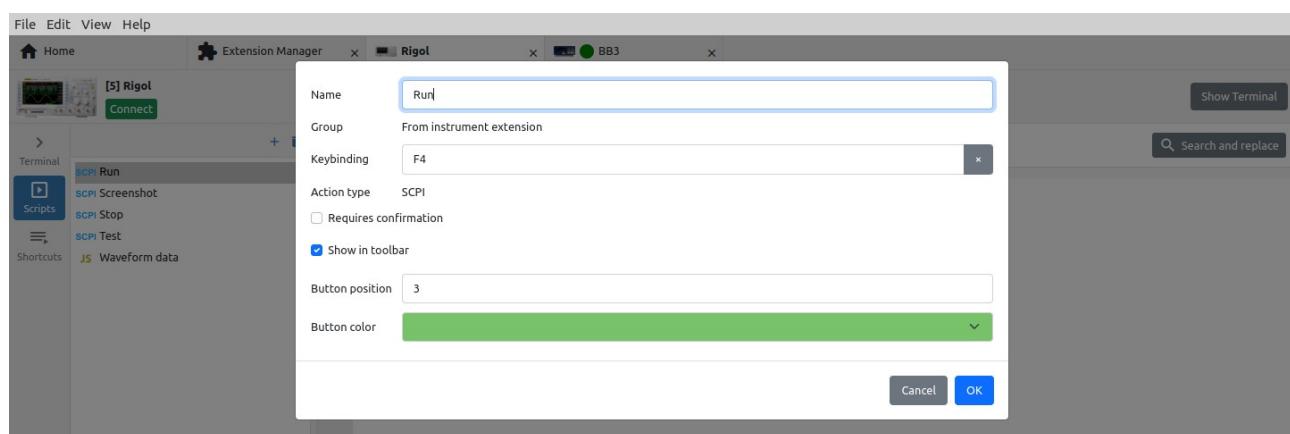


Fig. 34: Script shortcut editing

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

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

12.5. Shortcuts

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

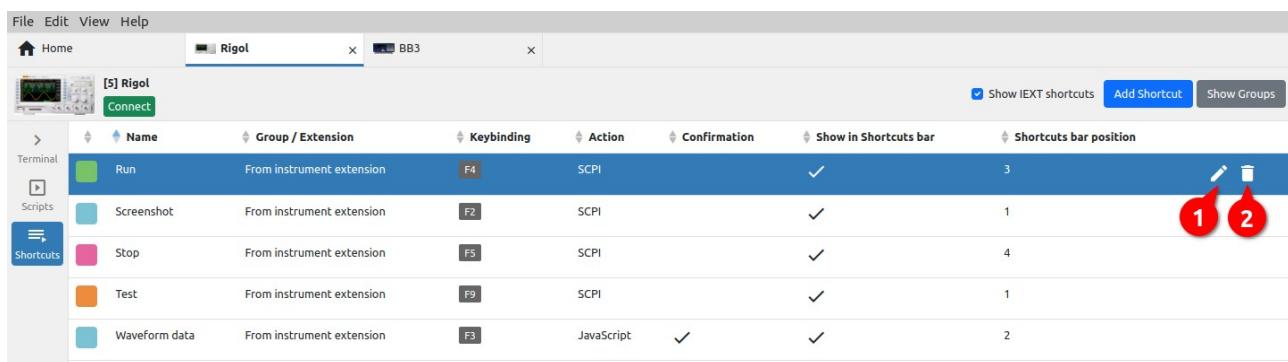


Fig. 35: Instrument shortcuts

#	Option	Description
1	<i>Edit shortcut</i>	Editing the shortcut (see Section 12.4.1.)
2	<i>Delete shortcut</i>	Deleting an existing shortcut.
	<i>Show IEXT shortcuts</i>	Filters the display of Shortcuts belonging to the installed Instrument Extension (IEXT).

Add Shortcut Adding a new shortcut opens the entry form as shown in Fig. 36.

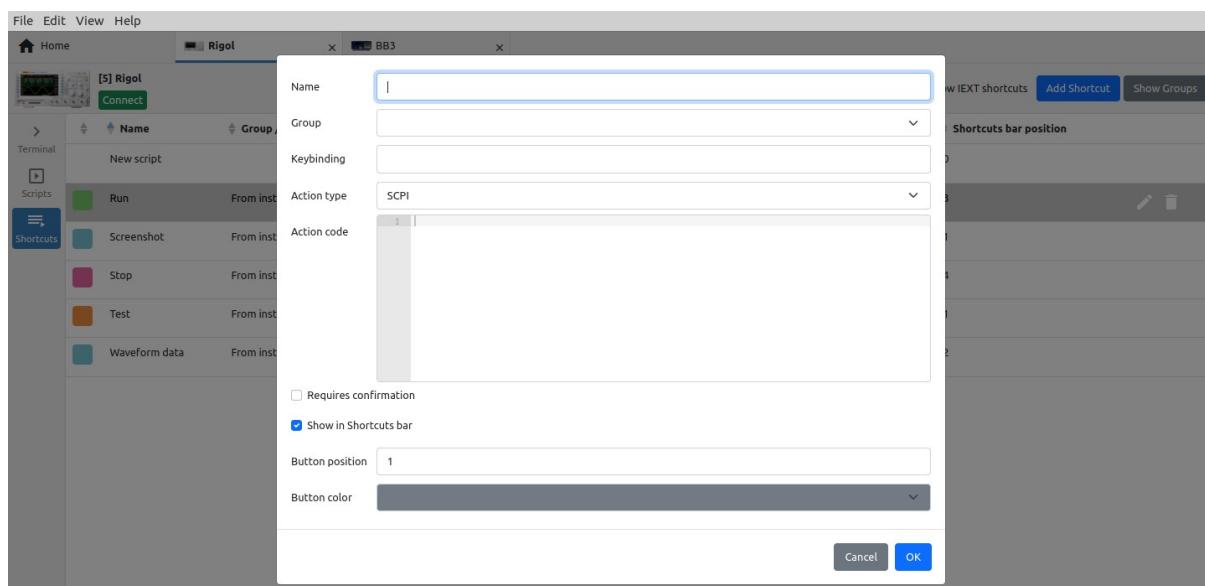


Fig. 36: Add new shortcut

Show Groups / Show Shortcuts

Toggle between displaying a list of shortcuts and groups (Fig. 37) of shortcuts.



Fig. 37: Instrument shortcut groups

I2.6. Lists

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

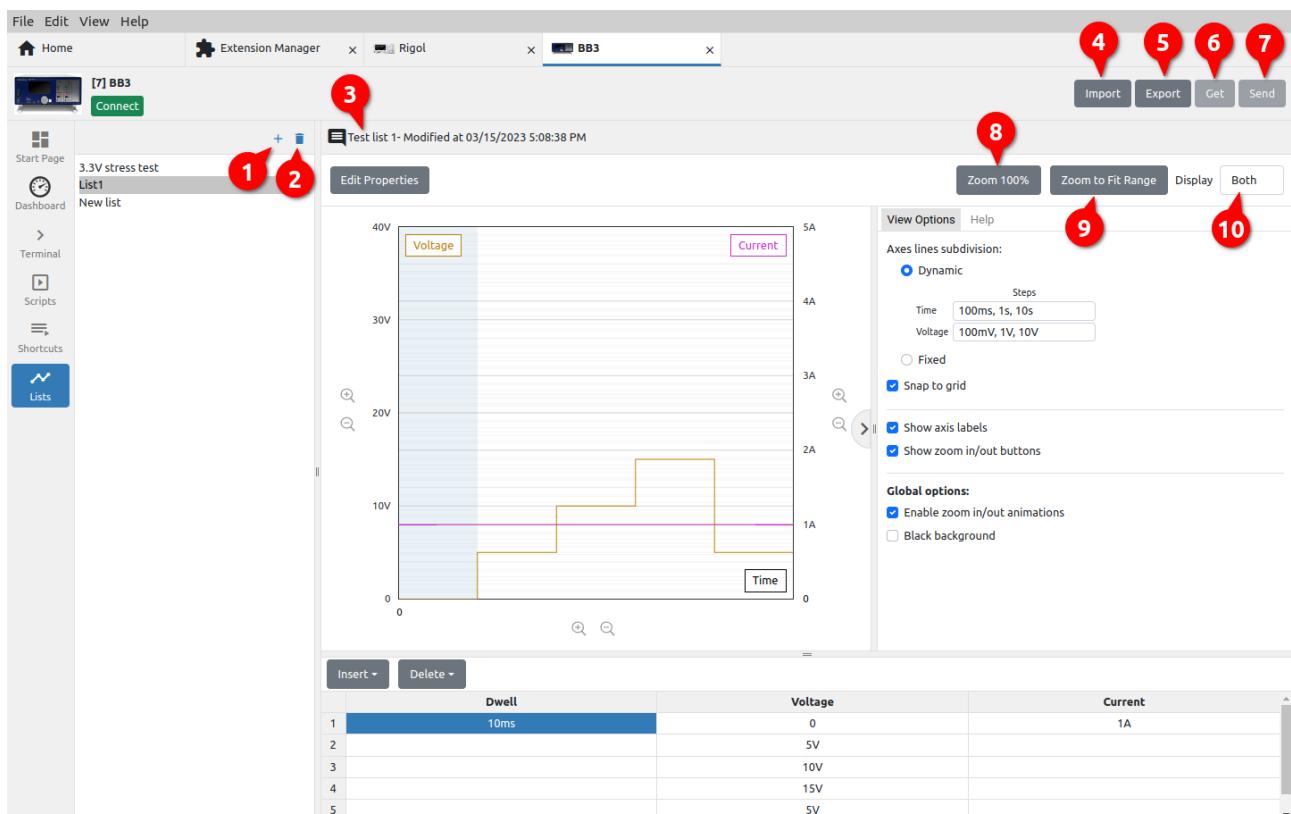


Fig. 38: Instrument programming lists

Option**Description**1 *Add list*

Creating a new list. The parameters of the list can be specified through a table (Fig. 38) or by defining envelope points that show the change of the parameter value over time.
In addition to the list *Type*, it will be necessary to define a *Name* and optionally a *Description*.

2 *Remove list*

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

3 *List info*

List description and datetime of last changes.

4 *Import*

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

5 *Export*

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

6 Get

Receiving a list from the instrument. The option will be disabled if connection is not established with the instrument.
Opens a menu (Fig. 39) where you can choose the source (e.g. channel) from which the list will be received. For the imported list, it is necessary to enter the name and description (Fig. 40).

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

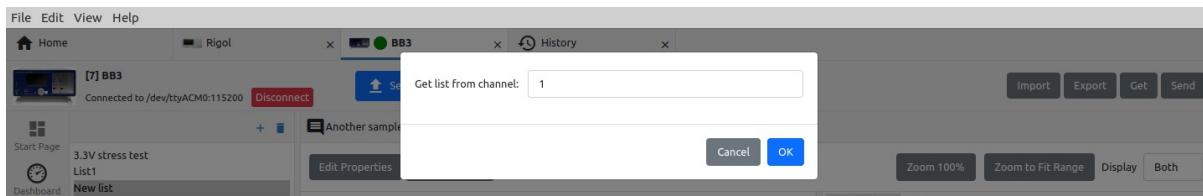


Fig. 39: List source selection

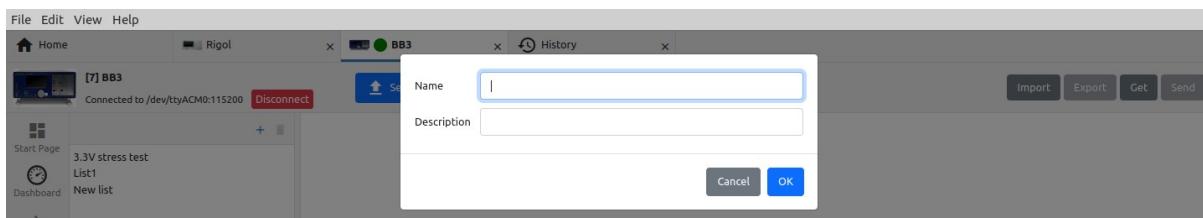


Fig. 40: Imported list parameters

7 Send

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

8 Zoom 100%

Display graph without scaling.

9 Zoom to Fit Range

Graph display scaled according to the largest defined value.

10 Display

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

12.6.1. Editing a list using a table

Editing the list via the table is shown in Fig. 38. The program parameters graph is drawn simultaneously with editing the table at the bottom of the graph. In the case shown, the list contains two program parameters: *Voltage* and *Current*, for which values should be entered as well as duration (*Dwell*). To define the value, it is possible to use the units prefix, e.g. ms for dwell, mV for voltage, and mA for current.

In Fig. 41 and Fig. 42 shows all options for inserting new lines and deleting existing ones.

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

Fig. 41: Table insertion options

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

Fig. 42: Table deletion options

12.6.2. Editing a list using an envelope

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

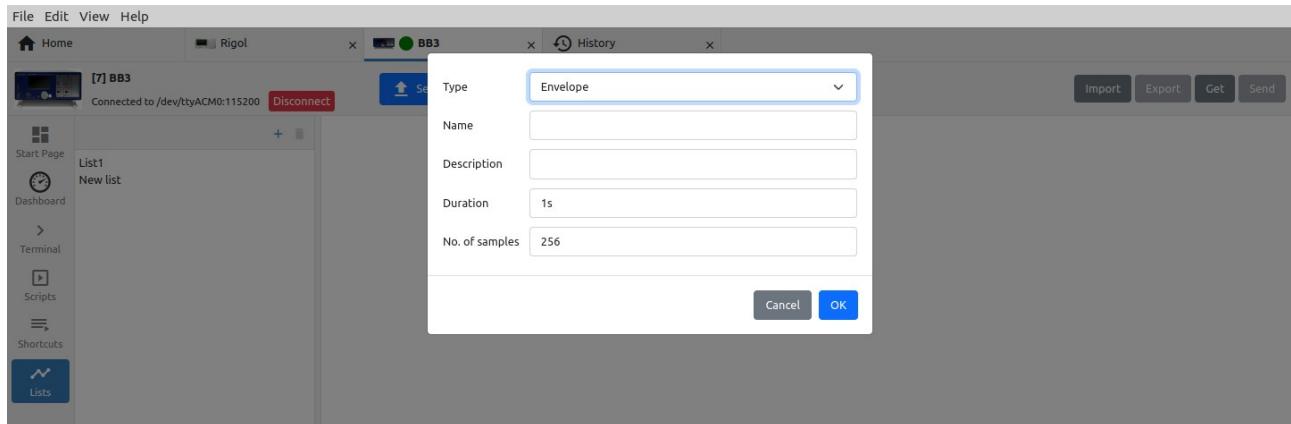


Fig. 43: Adding a new list in envelope mode

When creating a new list in envelope mode, it will be necessary to set two more parameters: the total duration of the program sequence and the number of samples (Fig. 43). The former is needed to be able to display the duration in the graph, and the later is needed to know how many points should be generated in total when sending the list to the instrument.

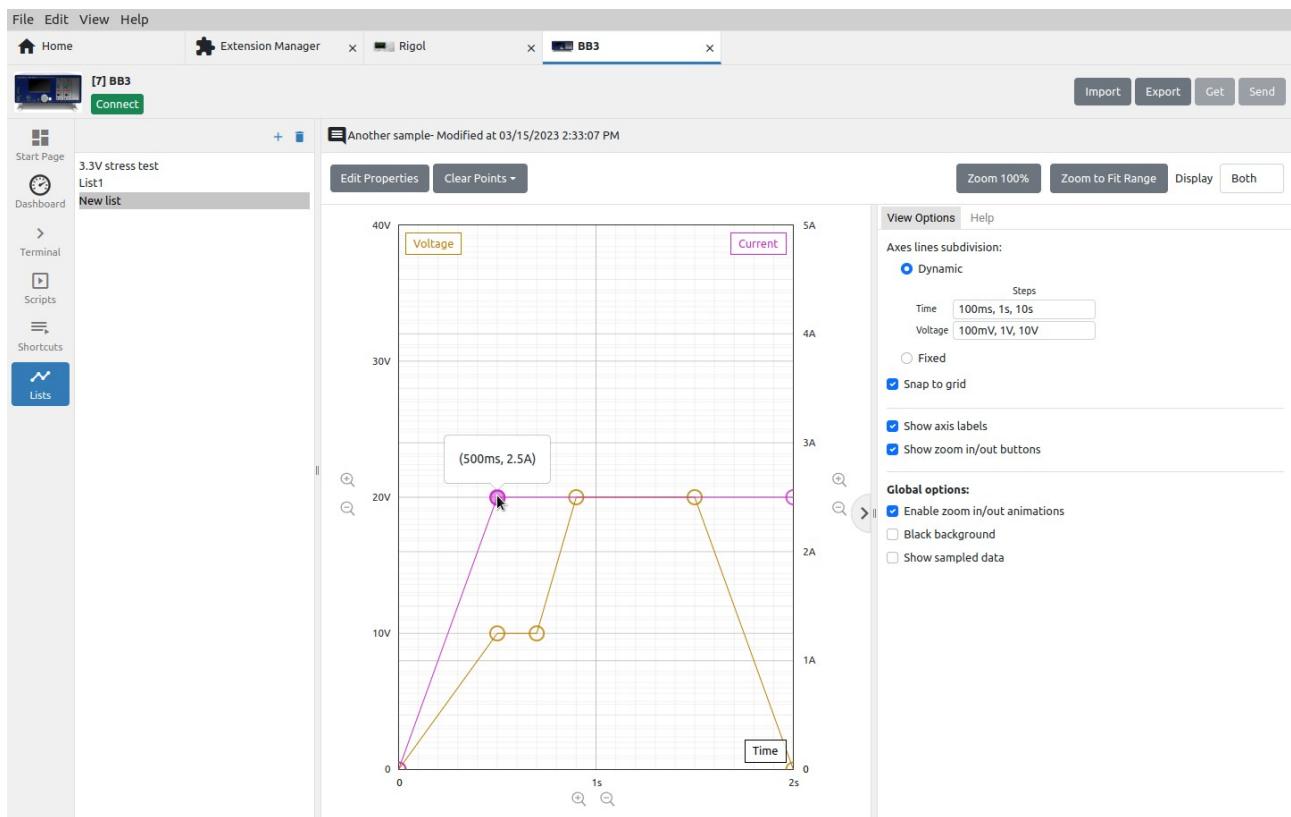


Fig. 44: Graph editing in envelope mode

The example in Fig. 44 contains 6 programming points for setting the voltage (light brown) and 3 for setting the current (magenta).

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

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

EEZ Studio Reference Guide

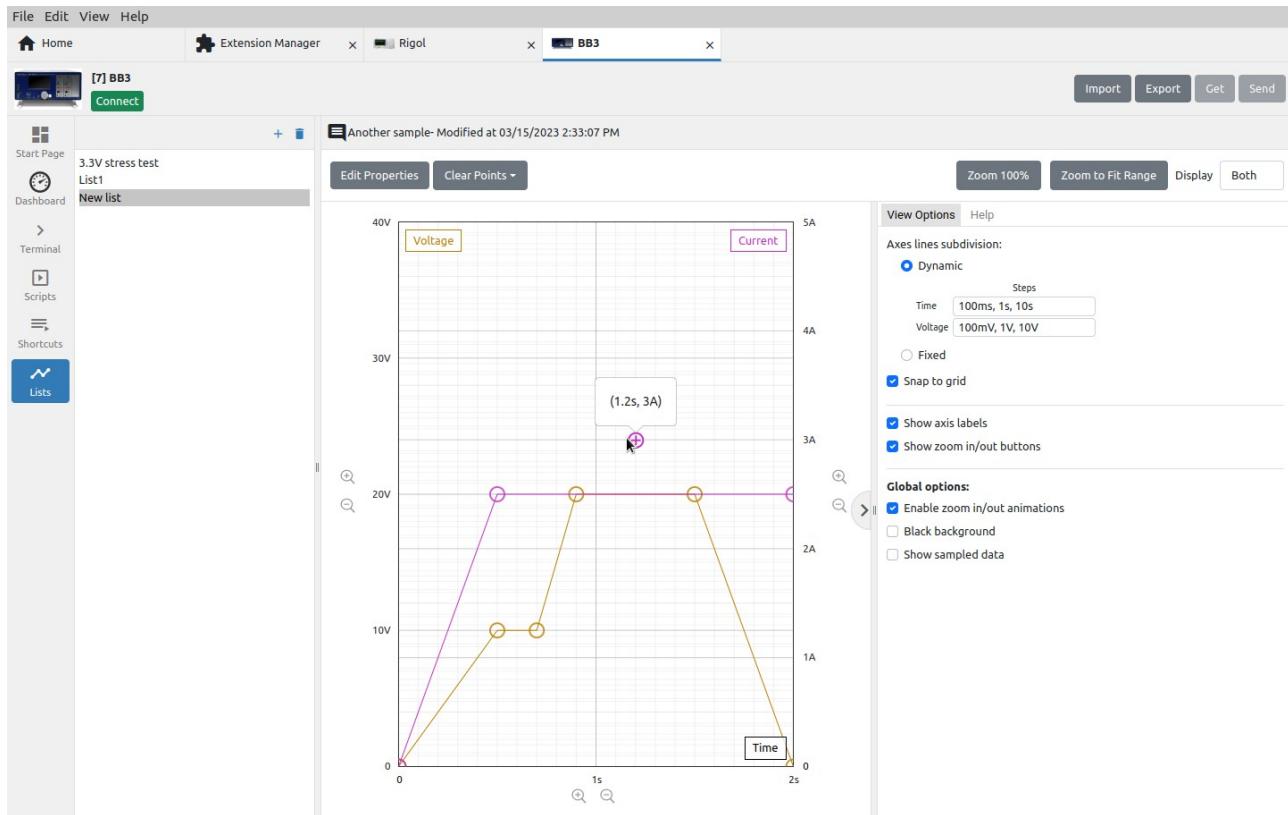


Fig. 45: Adding a new point in envelope mode

If you want to delete an existing point or manually edit its parameters after you have positioned yourself on it, you only need to click once more with the mouse when a dialog box will appear as shown in Fig. 46.

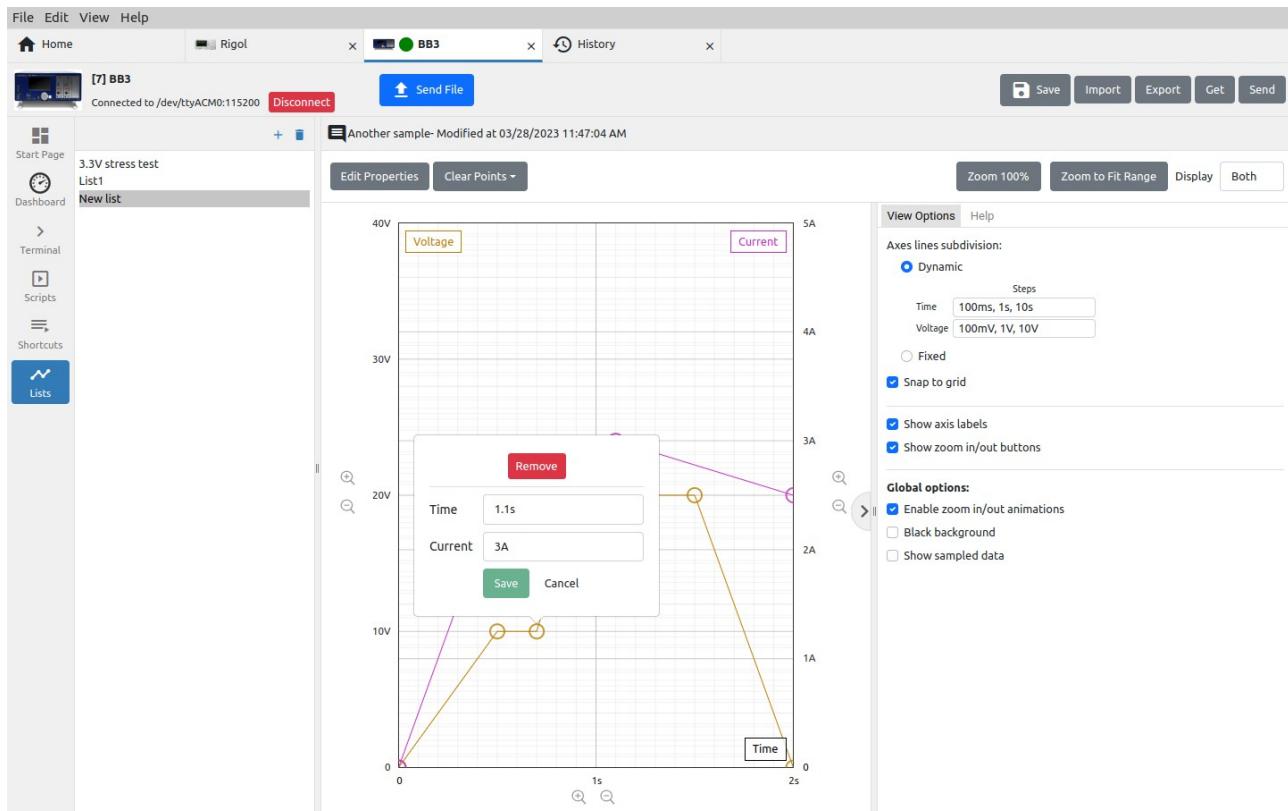


Fig. 46: Program point editing in envelope mode

12.6.3. List view options

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

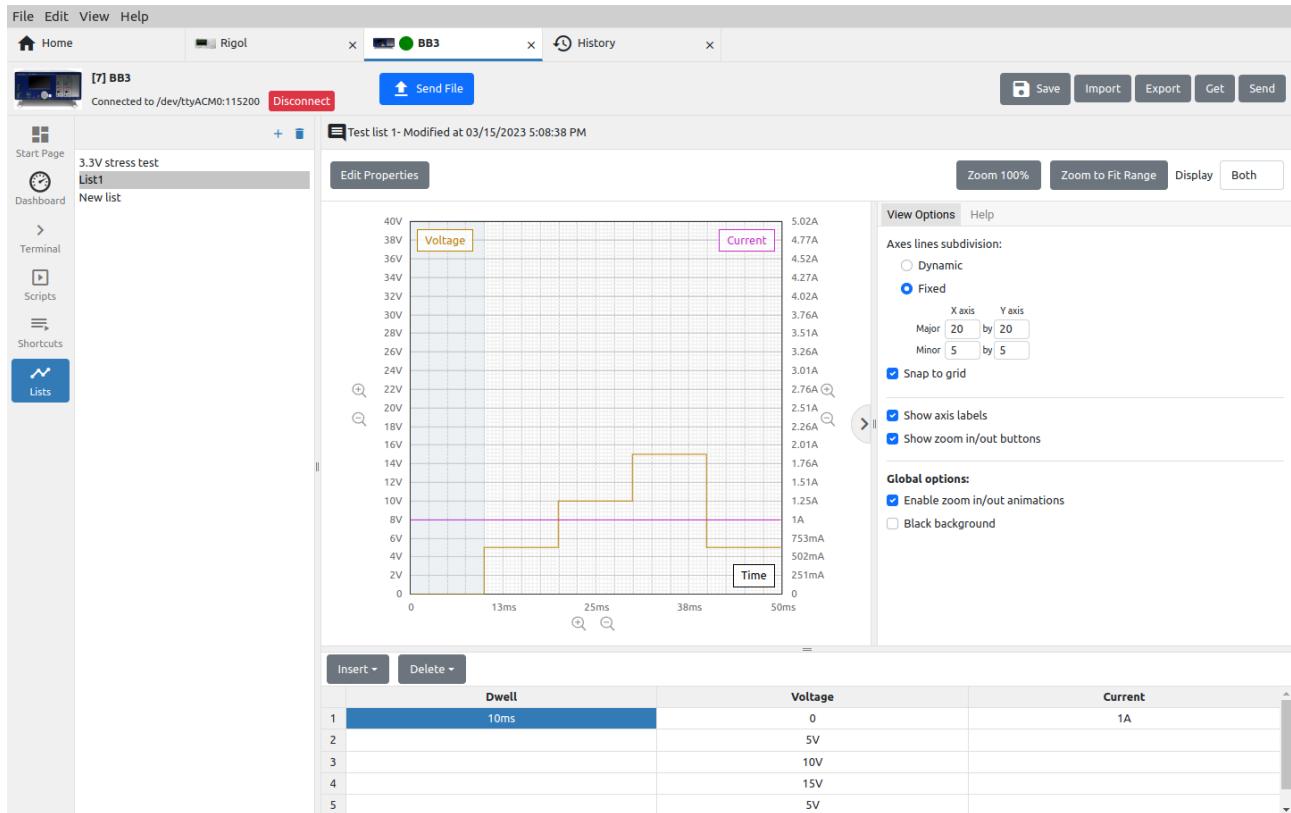


Fig. 47: Fixed graph view

12.6.4. List help

For zooming and navigating the graph, in addition to the zoom options located next to the x- and y-axes of the graph ("+" and "-" magnifier signs), a combination of mouse keys and control keys can be used. These additional options are shown in the Help tab as in Fig. 48.

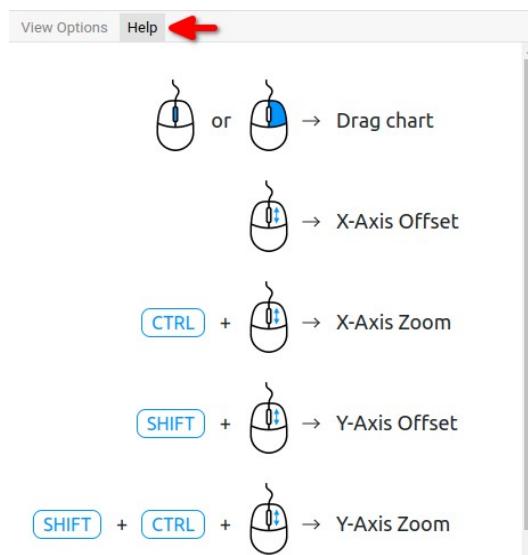


Fig. 48: Graph navigation and zoom help

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

Version: 0.10.3 (M17)
Date: 2023-08-31