CS LAB s.c.
ElectronicLaboratory

Manual • User • Guide

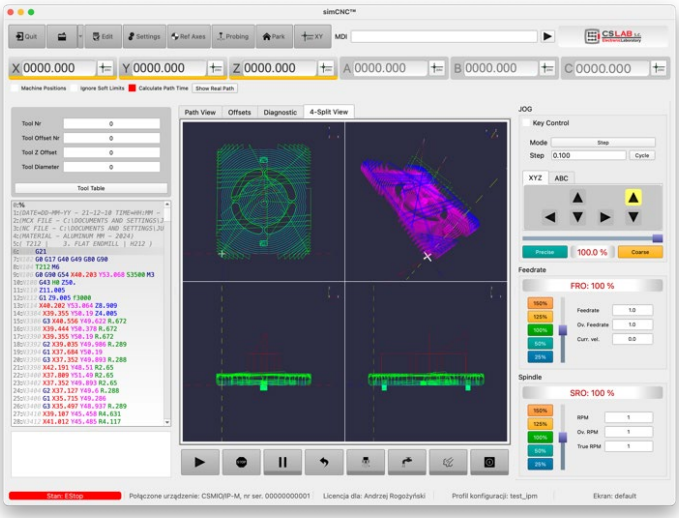
simCNC
Motion Control Software

User Interface Editor Guide

Content

1. General

simCNC software is equipped with an advanced graphical user interface editor, allowing you to create your own, original operator interfaces that are precisely tailored to customer requirements.
In conjunction with the Python scripting language and styling using the popular CSS, the editor allows you to create functional and visually attractive interfaces. The software code has been optimized in terms of performance to ensure responsiveness and comfort of use. SimCNC has a system of auto-layout and scaling graphic elements, which makes the designed interface more dynamic and able to adapt over a wide range to different sizes and resolutions of displays.
Since the simCNC software is multi-platform, the screen designs can be used without modification on Windows, macOS, and Linux.



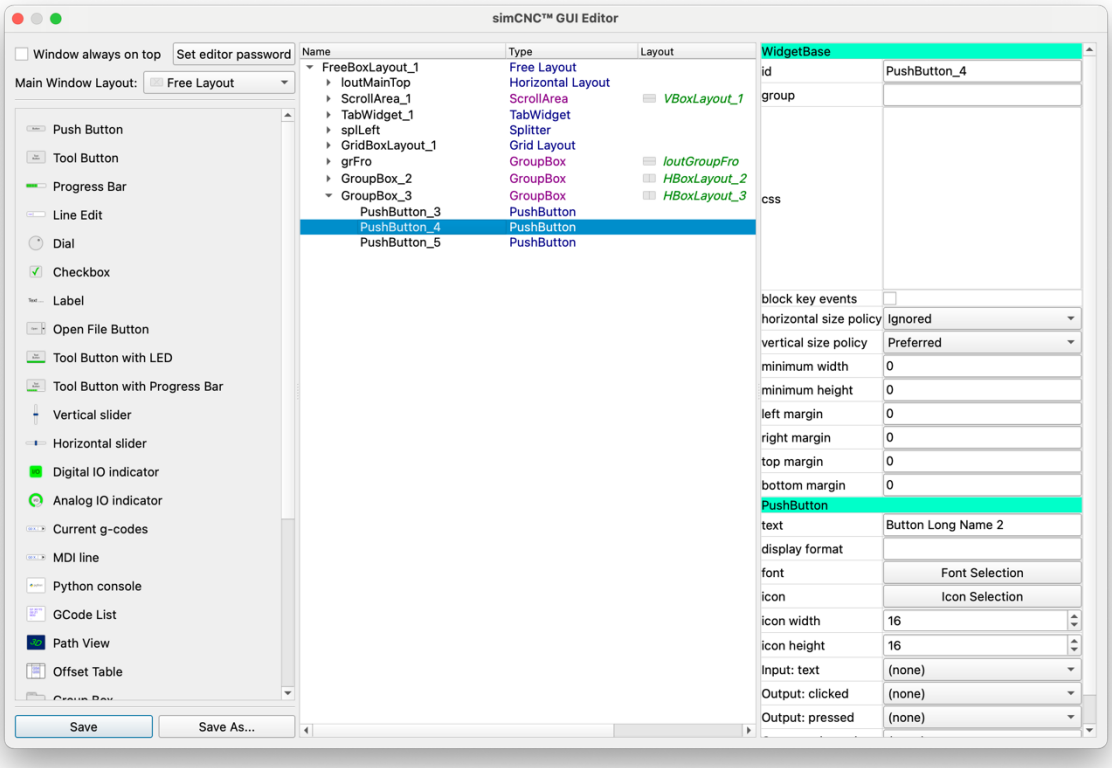
1.1. Recommendations and system requirements

The simCNC software and the integrated graphic editor do not have high hardware requirements. The software will even run on a RaspberryPI4 with 4GB of RAM. However, for comfortable work, especially if we create an interface "from scratch," it is good to be equipped with a good monitor with a higher resolution, e.g., 27" 2560x1440.
It is very convenient to have two monitors and a computer with more memory to be able to simultaneously use tools such as **Affinity Designer** or **Photoshop** to prepare graphic elements or **Visual Studio Code** - to edit style sheets (CSS) and Python scripts.

2. General conditions of working with the GUI editor

2.1. The editor window

Open the editor window from the menu: „Configuration → Open GUI editor".



The window is divided into three main vertical panels. In the left panel, there are the following elements (from the top):

- **Window always on top** – checking it makes the editor window is always on top
- **Set editor password** – password protection for screen editing
- **Main Window Layout** – choosing a layout of the main window container (see the description of the auto-layout system later in the manual)
- **Widget list** – to place a widget in a project, click your mouse and drag it to the designed simCNC screen
- **Save** – save changes
- **Save As** – save the project under a different name

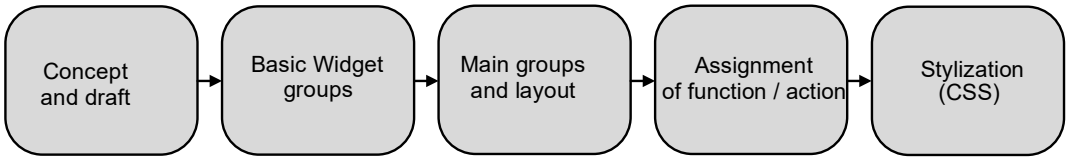
The middle panel shows a tree of all elements of the designed interface, while the right panel displays a list of properties of the selected widget or container.
The properties are described in the widgets section of the manual.

2.2. Shortcut keys

SHORTCUT	DESCRIPTION
CTRL-C	Copy selected elements to a clipboard
CTRL-V	Paste the elements from the clipboard into a selected container
CTRL-Z	Undo the last operation (undo)
CTRL-Y	Redo the undone operation (redo)
CTRL-S	Save project

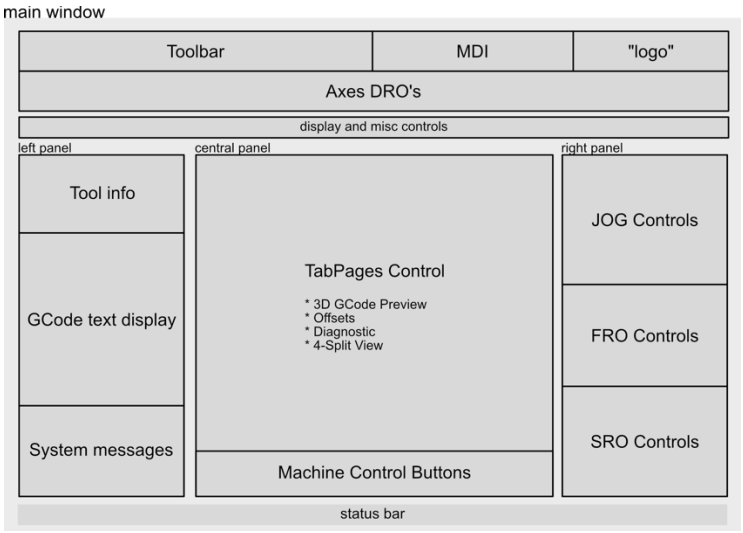
2.3. Stages of designing a new interface (workflow)

This chapter provides an overview of designing a new interface workflow. Do not worry if some of the terms used here are unclear. You can find a detailed description of the individual tools in the following chapters. Here we will focus on general rules.



2.3.1. Concept sketch

To take full advantage of the auto-layout system's capabilities and to not get lost in the growing number of elements, it is good to start with preparing a draft - a general concept of the layout of our interface elements.



The picture above shows an exemplary conceptual sketch of a simCNC default screen (**default**).

Top part:

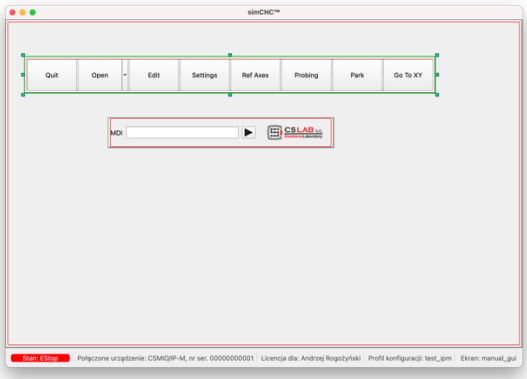
- **Toolbar** - a button bar
- **MDI** - area for quick entry of machine commands
- **"logo"** - company logo
- **Axes DRO's** - indication of a current position of machine axes
- **Display and misc controls** - controls related to displaying coordinates and others

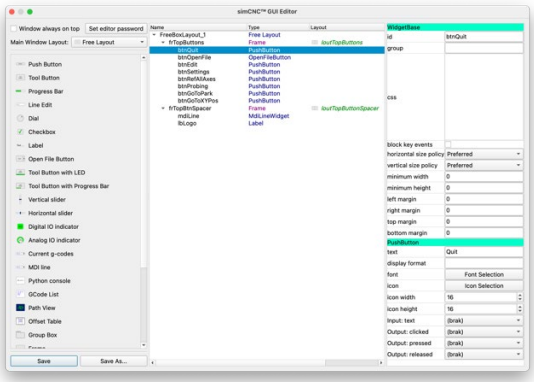
The lower part is divided into three sections:

- Left section
  - **Tool Info** - information about a current tool and offset
  - **Gcode text display** - displaying content of a loaded Gcode file
  - **System messages** - system messages console
- Central section
  - **TabPages Control** - Tab panel
    - 3D preview of Gcode file
    - Work offsets
    - Diagnostics
    - 3D view of a Gcode file in four projections
  - Operation control button bar

- Law section
    - o **JOG Controls** - control panel for manual axis control
    - o **FRO Controls** - feed rate control panel
    - o **SRO Controls** - spindle speed control panel

2.3.2. Setting up the basic widget groups

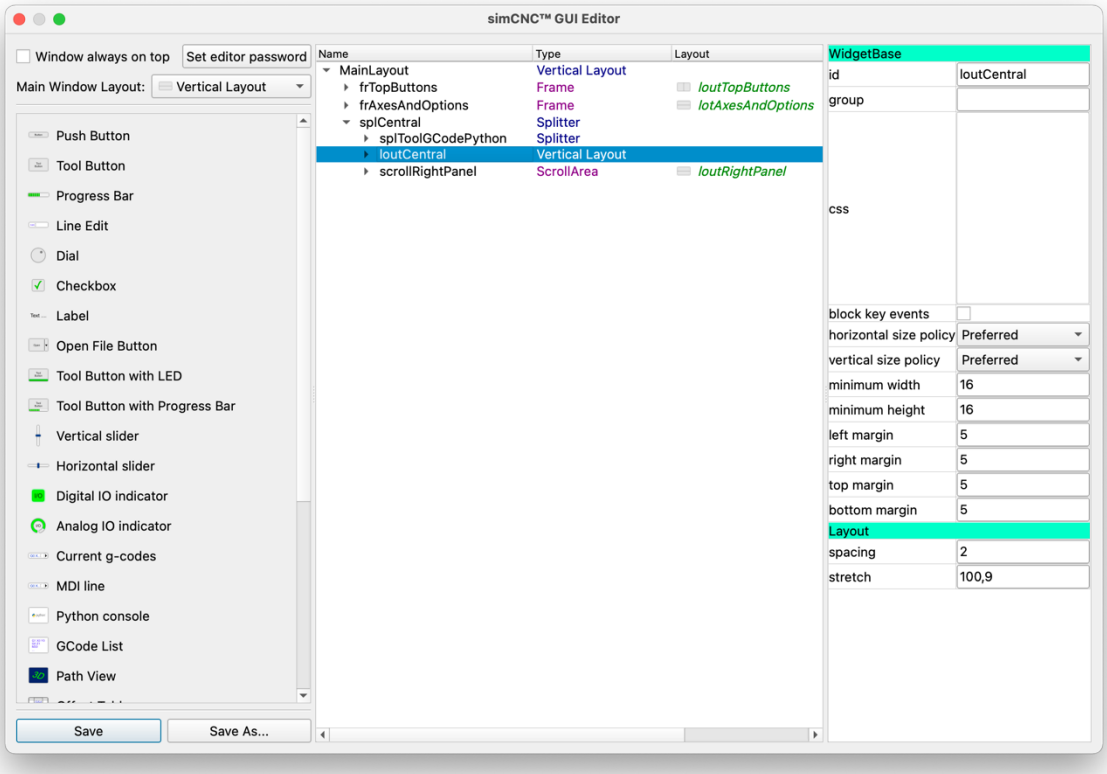




At this stage, we can initially start to create the basic interface widget groups. In the pictures above we can see main simCNC window with a group of buttons and MDI on the left and the editor window on the right.
It is not worth dealing with all the details, such as icons for buttons, styling, etc. at the beginning. There will be time for that later.
It is a good idea to place individual controls in groups and to pre-define the layout (arrangement) rules for them. The auto-layout system will be described in detail in a separate chapter.
It is important to give your widgets names that will allow you to easily find yourself in your project.

2.3.3. Setting up and arranging the main widget groups

The simCNC interface design has a hierarchical structure. You can combine the basic groups into larger ones and finally define a layout of the main software window. You can orientate in the hierarchy of project elements by looking at the middle panel of the editor window - the object tree. In the example below, you can see that the main **MainLayout** container contains three elements: **frTopButtons**, **frAxesAndOptions**, and **splCentral**. The **splCentral** in turn contains: **splToolGCodePython**, **loutCentral** and **scrollRightPanel**.



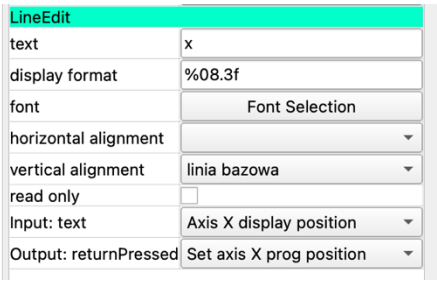
The default simCNC screen has three main groups, arranged vertically:

- **frTopButtons** - a frame with buttons, MDI, and logo
- **frAxesAndOptions** - frame with axis positions and options
- **splCentral** - group with left, center, and right panels

It's good to think over a hierarchy and interface groups during the drafting stage. This makes it easier to modify later and define scaling rules.

2.3.4. Assigning functions to widgets

At this stage, we assign input and output functions to the widgets. For example, for the axis position display widget, we define an input function "**Axis ... display position**" and an output function "**Set axis ... prog position**". The input action updates the displayed value, and the output action triggers the position setting action when an operator edits the widget content. All input and output actions are described in a separate chapter. You can also set Run Script as an output action for buttons and create a Python macro that will run when an operator clicks the button. This allows you to perform more complex tasks or actions that are not on the standard list.



2.3.5. Styling with style sheets (CSS)

The simCNC interface system has built-in support for **CSS** stylesheets. Styling is optional but recommended when you want to create a dynamic and visually attractive interface. Commands can be entered directly in the screen editor in the **css** field, or (recommended) create a separate file with the **.css** extension and put it in the screen directory. It is very convenient to be able to define groups of widgets for styling. To do this, in the edit mode, we enter a group name in the group field of widget properties, and we can later modify the visual attributes in the **css** file for all widgets that have a specific group name set. For example, in the default screen, all buttons on the top bar have **ctrlButtons** group name.

WidgetBase	
id	btnSettings
group	ctrlButtons

Here is a part of the **style.css** file that defines the appearance of the whole group:

```
[group="ctrlButtons"]{
  color: #404040;
  background-color: rgb(170, 170, 170);
  font-size: 12px;
}
[group="ctrlButtons"]:hover {
  color: rgb(112, 14, 14);
  background-color: lightgray;
  font-size: 13px;
}
[group="ctrlButtons"][darkTheme="true"]{
  color: #FFFFFF;
  background-color: rgb(60, 60, 60);
  font-size: 12px;
}
[group="ctrlButtons"][darkTheme="true"]:hover {
  color: rgb(182, 14, 14);
  background-color: lightgray;
  font-size: 13px;
}
```

Shortened notation of the widget "id" when entering "css" in the editor window.

If we enter the content of **CSS** in the editor window, we can use the option of shortening the widget identifier. To do this, enter **#id**, as shown below.

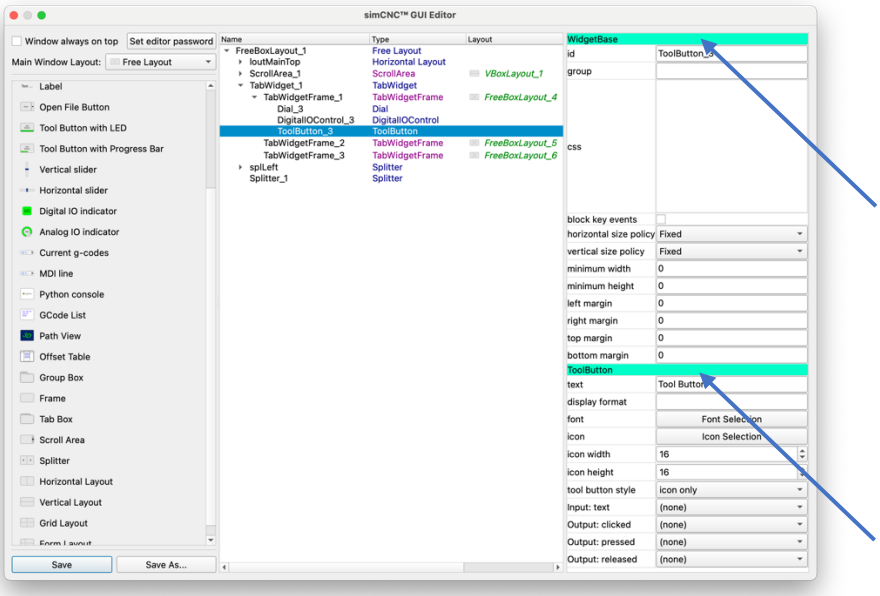
WidgetBase	
id	btnEdit
group	ctrlButtons

css	#id {
	color: yellow;
	background-color: red;
	}

A normal, complete notation would be:
[id="btnEdit"] {
...
}
The standard notation is also allowed.

3. Widgets

Widget selection in edit mode displays a list of its properties in the right panel of the editor window.



In the above example of the **Tool Button** widget, you can see that the properties are grouped. **WidgetBase** is a group of properties common to all widgets. **ToolButton** is a properties group for a specific **Tool Button** widget.

The table below describes the properties common to all widgets.

PROPERTY	DESCRIPTION
Id	Widget identifier. You should assign intuitive identifiers, especially if you later use css styling functions or plan to refer to widgets from Python macros. For example, **btnStartGCode** is a much better name than **buton_123**.
Group	Widget group identifier. Very useful for styling (css). By giving the same group name to multiple widgets, you can significantly reduce the amount of code in the css stylesheet.
Css	Quick commands typing field for styling a widget. It's useful, especially when we are experimenting with different properties. Ultimately, it is better to create a **.css** file (or files) in a directory of the designed screen.
block key events	Selecting this option means that the widget does not forward information about buttons pressed on a keyboard. It's useful, for example, in edit fields like MDI. Thanks to this, changing a cursor position in the edit field does not cause a machine movement when JOG control is activated from a keyboard.
Horizontal size policy	Widget horizontal scaling policy. Detailed description in the chapter on auto-layout.
Vertical size policy	Widget vertical scaling policy. Detailed description in the chapter on auto-layout.
Minimum width	Minimum allowed width of a widget.
Minimum height	Minimum allowed height of a widget.
Left margin	Left margin of a widget.
Right margin	Right margin of a widget.
Top margin	Top margin of a widget.
Bottom margin	Bottom margin of a widget.

3.1. Push Button and Tool Button

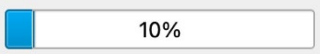
Both these button widgets have almost identical functionality. They slightly differ visually. The ToolButton has an additional option of selecting a position of a displayed icon. Both widgets are functionally identical.

Properties:

PROPERTY	DESCRIPTION
Text	Text displayed on a button
Display format	A text string that defines a format of displayed values. Compliant with the "printf" standard: https://en.wikipedia.org/wiki/Printf_format_string For example, **"% .3f"** displays a floating-point value to three decimal places.
Font	Setting a widget font
Icon	Selection of a picture for a button icon
Icon width	Width of an icon
Icon height	Height of an icon
Tool buton style	Style of the ToolButton. • **Icon only** • **Text only** • **Text beside icon** • **Text under icon** • **Follow style** (as defined in a style sheet)
Input: text	Widget input - selection of a parameter that will update a text displayed on the button
Output: clicked	Widget output - selection of an action that will be activated after clicking on the button
Output: pressed	Widget output - selection of an action that will be activated after pressing a left mouse button on the button
Output: released	Widget output - selection of an action that will be activated when releasing a left mouse button on the button

3.2. Progress Bar

The Progress bar can be used to visualize many different values, such as FRO, SRO, etc.

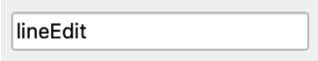


Properties:

PROPERTY	DESCRIPTION
Text	Text displayed on a widget
Display format	A text string that defines a format of displayed values. Compliant with the "printf" standard: https://en.wikipedia.org/wiki/Printf_format_string For example, **"% .3f"** displays a floating-point value to three decimal places.
Font	Setting a widget font
Horizontal alignment	The horizontal position of the widget content (text) • **Left** – left align • **Right** – right align • **Center** – centering • **Justify** – justification
Vertical alignment	The vertical position of the widget content (text) • **Top** – top align • **Bottom** – bottom align • **Center** – centering • **Baseline** – align to baseline
Read only	Selecting this option means that the widget content is read-only and cannot be edited.
Input: value	Widget input - selection of a parameter that will update the bar progress value

3.3. Line Edit

Text display and editing field.



Properties:

PROPERTY	DESCRIPTION
Text	Displayed text - widget content
Display format	A text string that defines a format of displayed values. Compliant with the "printf" standard: https://en.wikipedia.org/wiki/Printf_format_string For example, **"% .3f"** displays a floating-point value to three decimal places.
Font	Setting a widget font
Horizontal alignment	The horizontal position of the widget content (text) • **Left** – left align • **Right** – right align • **Center** – centering • **Justify** – justification
Vertical alignment	The vertical position of the widget content (text) • **Top** – top align • **Bottom** – bottom align • **Center** – centering • **Baseline** – align to baseline
Read only	Selecting this option means that the widget content is read-only and cannot be edited.
Input: text	Widget input - selection of a parameter that will update the text displayed in the edit field
Output: returnPressed	Widget output - selection of an action that will be activated when the return key is pressed, when editing the widget content is completed

3.4. Dial

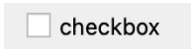
Adjustment Dial – setting numerical values within a specified range.

Properties:

PROPERTY	DESCRIPTION
Wrapping	Setting the rotation range to full 360° with no dead zone.
Notches visible	Notches display enable
Notch target	Notch target
Value	The current value
Minimum	Minimum value
Maximum	Maximum value
Single step	Regular adjustment pitch (pressing the up or down arrow)
Page step	Rapid adjustment stroke (pressing page up or page down)
Inverted appearance	Inverted dial display. Swap the minimum and the maximum.
Inverted controls	Reversing the adjustment direction
Input: value	Widget input - selection of a parameter that will update a position of the dial
Output: valueChanged	Widget output - selection of an action that will be activated when a user changes position of the dial

3.5. Checkbox

Selection button. Used to enable/disable options, e.g., enable/ disable software limits.



Properties:

PROPERTY	DESCRIPTION
Text	Displayed text
Checkbox state	Sets a selection state
Display format	A text string that defines a format of displayed values. Compliant with the "printf" standard: https://en.wikipedia.org/wiki/Printf_format_string For example, **"% .3f"** displays a floating-point value to three decimal places.
Font	Setting a widget font
Input: checkbox state	Widget input - selection of a parameter that will update the widget state
Input: text	Widget input - selection of a parameter that will update the widget text
Output: stateChanged	Widget output - selection of an action that will be activated when the widget state changes

3.6. Label

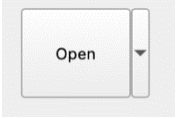
The label widget is used to display text or graphics. It does not activate any actions.

Properties:

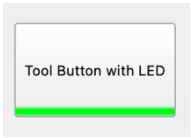
PROPERTY	DESCRIPTION
Text	Displayed text
Display format	A text string that defines a format of displayed values. Compliant with the "printf" standard: https://en.wikipedia.org/wiki/Printf_format_string For example, **"% .3f"** displays a floating-point value to three decimal places.
Font	Setting a widget font
Word wrap	Words braking enable
Horizontal alignment	The horizontal position of the widget content (text) • **Left** – left align • **Right** – right align • **Center** – centering • **Justify** – justification
Vertical alignment	The vertical position of the widget content (text) • **Top** – top align • **Bottom** – bottom align • **Center** – centering • **Baseline** – align to baseline
Pixmap	Select a bitmap to display
Scale mode	Select a bitmap scaling type • **Normal** - without keeping the aspect ratio • **KeepAspect** – keeps aspect ratio
Input: text	Widget input - selection of a parameter that will update the widget text

3.7. Open File Button

A special button for loading GCode files. Similar to the **Tool Button** but additionally displays a list of recently opened files. It does not require setting input and output actions to run. The properties are the same as for the **Tool Button**.



3.8. Tool Button with LED

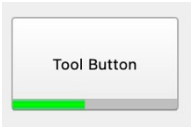


It's a **Tool Button** variant. It also provides a state indicator light display to define an action that will update the **LED** state.

It supports all the Tool Button properties plus the listed below:

PROPERTIES	DESCRIPTION
LED visible	Enables/disables the indicator LED on a button
LED state	LED state – enable / disable
LED interval	If the value is higher than zero, it activates LED flashing. The value is milliseconds, and it determines half of the period. For example 500 means *T = 500x2 = 1s ; f = 1/T = 1Hz*
Color	The LED indicator color selection
Input: LED state	Widget input – selection of a parameter that will update a state of the LED indicator displayed on the button.
Input: LED interval	Widget input – selection of a parameter that will update flashing interval of the LED indicator.

3.9. Tool Button with Progress Bar

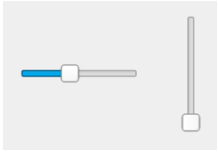


It's another **Tool Button** variant. It also provides a progress bar display to define a separate action that will update a value.

It supports all the Tool Button properties plus the listed below:

PROPERTIES	DESCRIPTION
Value	The current value
Minimum	The minimal value
Maximum	The maximal value
Color	The bar color selection
Input: value	Widget input – selection of a parameter that will update the progress bar value.

3.10. Horizontal Slider and Vertical Slider



There are two variants – horizontal and vertical slider for setting a value within a specific range.

Properties:

PROPERTIES	DESCRIPTION
Value	The current value
Minimum	The minimal value
Maximum	The maximal value
Single step	Regular adjustment pitch (pressing the up or down arrow)
Page step	Rapid adjustment stroke (pressing page up or page down)
Inverted appearance	Inverted slider display. Swap the minimum and the maximum.
Inverted controls	Reversing the adjustment direction
Input: value	Widget input – selection of a parameter that will update a slider position
Output: valueChanged	Widget output – selection of an action that will be activated when a user changes the slider position

3.11. Digital IO indicator

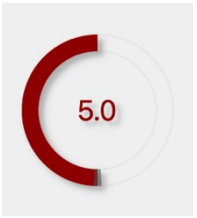
An indicator for displaying logic values (0, 1) state. It is used, for example, to signalize hardware inputs/outputs state. It may also activate an action after clicking.

Properties:

PROPERTIES	DESCRIPTION
Text	Displayed text
State	Sets the indicator state
Display format	A text string that defines a format of displayed values. Compliant with the "printf" standard: https://en.wikipedia.org/wiki/Printf_format_string For example, **"% .3f"** displays a floating-point value to three decimal places.
Clickable	Enable indicator state control by mouse clicking
Color	Indicator color selection
Input: text	Widget input – selection of a parameter that will update the widget text
Input: state	Widget input – selection of a parameter that will update the widget state
Output: stateChanged	Widget output – selection of an action that will be activated when a user changes the widget state

3.12. Analog IO indicator

An indicator for displaying number values. It's used, for example, for presenting hardware values of analog inputs and outputs. It can set values as well.



Properties:

PROPERTIES	DESCRIPTION
Text	Displayed text
Display format	A text string that defines a format of displayed values. Compliant with the "printf" standard: https://en.wikipedia.org/wiki/Printf_format_string For example, **"% .3f"** displays a floating-point value to three decimal places.
Value	The current value
Clickable	Enable indicator state control by mouse clicking
Color	Indicator color selection
Maximum	Maximal value
Input: text	Widget input – selection of a parameter that will update the widget text
Input: value	Widget input – selection of a parameter that will update the indicator text
Output: valueChanged	Widget output – selection of an action that will be activated when a user changes the indicator value

3.13. Current G-Codes

A list of current modal commands state (machine state).

Properties:

PROPERTIES	DESCRIPTION
Font	Widget font settings
Horizontal alignment	Horizontal widget content (text) position. • Left – left align • Right – right align • Center – centering • Justify – justification
Vertical alignment	Vertical widget content (text) position. • Top – top align • Bottom – bottom align • Center – centering • Baseline – base line align

3.14. MDI Line

A field for quick machine commands entering (MDI).

3.15. Python Console

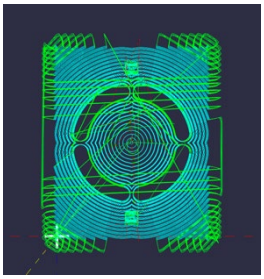
A text panel with Python scripts messages and other system information and warning messages.

3.16. GCode List

Displays the current content of a loaded gcode file in text form and currently executed line during machining. It provides by double-click selection of a line at which the machining starts.

3.17. Path View

It displays the content of a currently loaded gcode file and a current machine axis position in graphic form (3D).

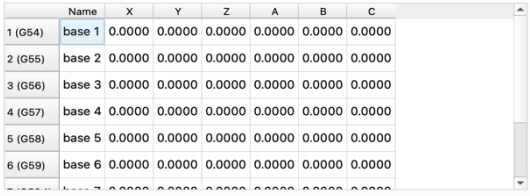


Properties:

PROPERTY	DESCRIPTION
Perspective view	Enables perspective view
Soft limits visible	Enables soft limits visualization
Default view	Default view type: • **Top View** • **Bottom View** • **Right View** • **Left View** • **Front View** • **Rear View** • **Isometric view**
Default z height mapping	Default height (Z) mapping type with colors: • **None** – without coloring • **Color** – color mapping • **Grayscale** – greyscale mapping

3.18. Offset Table

It's a widget for displaying work offsets table and for editing them.



	Name	X	Y	Z	A	B	C
1 (G54)	base 1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2 (G55)	base 2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3 (G56)	base 3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
4 (G57)	base 4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5 (G58)	base 5	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
6 (G59)	base 6	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

3.19. Group Box

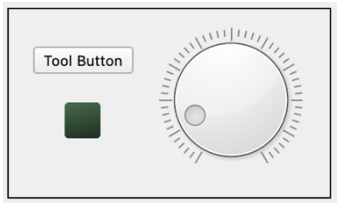
It's one of the basic containers for widgets grouping.

Properties:

PROPERTY	DESCRIPTION
Layout type	Type of auto-layout in a group
Title	Displayed name of a group

3.20. Frame

The frame is also one of the basic containers for widgets grouping. It looks slightly different from the Group Box (it doesn't display the title).

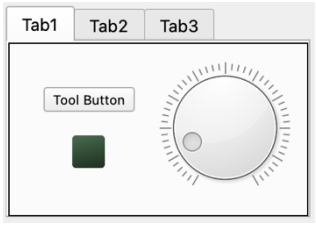


Properties:

PROPERTY	DESCRIPTION
Layout type	Type of auto-layout in a frame
Shadow	Frame shadow type: • **Plain** – standard • **Raised** • **Sunken**
Shape	Frame shape type: • **No frame** • **Box** – rectangular • **Panel** – raised or sunken panel • **Styled panel** – a panel according to actual gui style • **Horizontal line** • **Vertical line** • **Windows styled panel** – style as Windows 2000
Line width	Width of the line
Mid line width	Width of the midline

3.21. Tab Box

A container with tabs for widgets grouping.

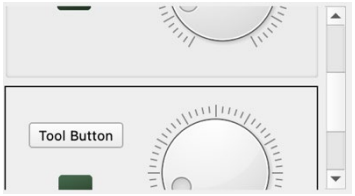


Properties:

PROPERTY	DESCRIPTION
Tabs quantity	Number of tabs
Current tab	The current (default) tab

3.22. Scroll Area

A container with scroll bars. Helpful when we have a limited area, and we want to include more elements in our interface. It's also a good idea to use it if you want the interface design to display correctly on screens with lower resolutions



Properties:

PROPERTY	DESCRIPTION
Spacing	Space between widgets
Stretch	Scaling rate of each element in the container
Layout type	Auto-layout type: • **Free Layout** – no auto-layout • **Horizontal Layout** • **Vertical Layout** • **Grid Layout** • **Form Layout** – vertical pairs layout
Horizontal scroll bar policy	Horizontal content scrolling bar display policy: • **As Needed** • **Always Off** • **Always On**
Vertical scroll bar policy	Vertical content scrolling bar display policy: • **As Needed** • **Always Off** • **Always On**
Shadow	Frame shadow type: • **Plain** – standard • **Raised** • **Sunken**
Shape	Frame shape type: • **No frame** • **Box** – rectangular • **Panel** – raised or sunken panel • **Styled panel** – a panel according to actual gui style • **Horizontal line** • **Vertical line** • **Windows styled panel** – style as Windows 2000
Line width	Width of the line
Mid line width	Width of the midline

3.23. Horizontal Layout

Horizontal auto-layout system container. It groups widgets and defines their layout and scaling way. The auto-layout system is described in a separate chapter.

Properties:

PROPERTY	DESCRIPTION
Spacing	Space between widgets
Stretch	Scaling rate of each element in the container

3.24. Vertical Layout

Vertical auto-layout system container. It groups widgets and defines their layout and scaling way. The auto-layout system is described in a separate chapter.

Properties:

PROPERTY	DESCRIPTION
Spacing	Space between widgets
Stretch	Scaling rate of each element in the container

3.25. Grid Layout

Grid auto-layout system container. It groups widgets and defines their layout and scaling way. The auto-layout system is described in a separate chapter.

Properties:

PROPERTY	DESCRIPTION
Horizontal spacing	Horizontal space between widgets
Vertical spacing	Vertical space between widgets
Column stretch	Columns scaling rate in the container
Row stretch	Rows scaling rate in the container

3.26. Form Layout

Form auto-layout system container. It groups widgets and defines their layout and scaling way. The auto-layout system is described in a separate chapter.

Properties

PROPERTY	DESCRIPTION
Horizontal spacing	Horizontal space between widgets
Vertical spacing	Vertical space between widgets

3.27. Splitter

Dynamic split and elements size auto-layout system container. It groups widgets and defines their layout and scaling way. The auto-layout system is described in a separate chapter.

Properties:

PROPERTY	DESCRIPTION
Orientation	Orientation of elements • **Horizontal** • **Vertical**
Shadow	Frame shadow type • **Plain** – standard • **Raised** • **Sunken**

Shape	Frame shape type
	• **No frame**
	• **Box** – rectangular
	• **Panel** – raised or sunken panel
	  • **Styled panel** – a panel according to current style
	  • **Horizontal line**
	  • **Vertical line**
	• **Windows styled panel** – style as Windows 2000

Line width	Width of the line
Mid line width	Width of the midline

4. Auto-Layout System

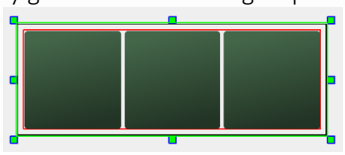
The auto-layout system is a key element of the simCNC user interface. Thanks to it, the created interface is convenient, dynamic, and can adapt to different display sizes to a large extent. It is worth spending some time learning more about the system rules and practicing the operation of this system's elements. After that, you will easily design attractive and functional interfaces quickly and conveniently.

4.1. Container types

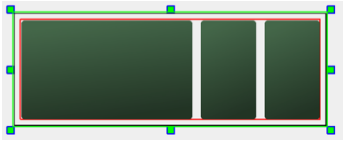
The basic element of the auto-layout system is the so-called container, in which we place widgets and other containers. The type of container determines a general principle of an arrangement of the elements in it.

4.1.1. Horizontal Layout

Container with horizontal layout. As you can easily guess – it is used to group widgets, arranging them horizontally.



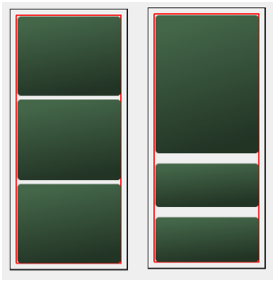
You can see three Digital IO Indicator widgets arranged horizontally in a container in the image above. It is worth paying attention to two properties of the container, which control the distance between widgets (spacing) and division of space for individual elements (**stretch**).



Above is the same container, but the distance **(spacing)** was changed from "1" to "5" and the division of space (stretch) was set to "3,1,1". The notation "**3,1,1**" means that the recommended size of the first widget will be 3x larger than the others. Please pay attention to the word **recommended.** The elements scaling is also affected by the size **policy** parameters described in a separate subsection. In the example above, the **size policy** is set to **Preferred** for all three elements in the container.

4.1.2. Vertical Layout

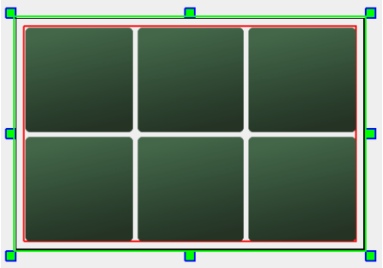
Container with the vertical layout. The principle of operation of this container is identical to **the Horizontal Layout**, and the only difference is the vertical arrangement of the elements in it.



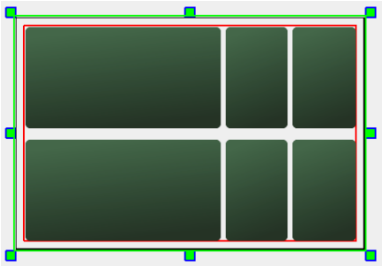
Above, you can see two **Vertical Layout** containers, with the same widgets and two variants of settings, as in the Horizontal **Layout** container description.

4.1.3. Grid Layout

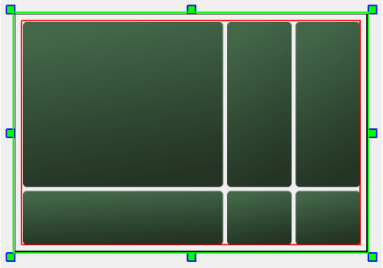
Container with grid arrangement. The elements here are arranged both horizontally and vertically in the form of rows and columns.



You can see six Digital IO Indicator widgets placed in the Grid Layout container in the screenshot above. The control over spacing and **stretch** coefficients principle is identical to the previously described **Horizontal** and **Vertical Layout,** except that we have here the possibility to define properties for columns and rows separately.



Another example shows the change **of column spacing** to "1" and column **stretch** to "3,1,1". As you might expect, the effect is reducing the distance between the elements horizontally, and the first column is three times wider than the others.

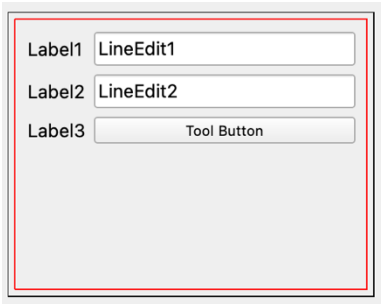


We used here identical settings for **row spacing** (1) and row **stretch** (3,1,1). Now the distances in the rows and columns are the same, and the first row is three times larger than the others.

A common mistake made by less experienced users is to abuse the **Grid Layout** container and place all the interface elements in it. This approach causes issues when the number of elements increases. When we have a lot of columns and rows, it becomes increasingly difficult to control the size and arrangement of elements. A much better approach is to build smaller interface blocks and combine them into larger ones. Remember that another container can also be an element of the container.

4.1.4. Form Layout

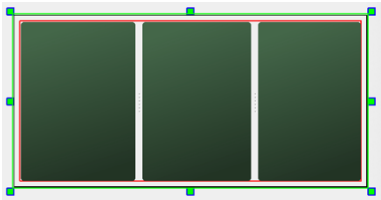
Container with form layout. It is a kind of a variety of **Grid Layout,** specialized for creating forms such as below.



Form Layout arranges pairs of elements in rows, so it is basically a grid made of two columns and a free number of rows. It is great for applications like the example above, where we have repeating rows like "**description→widget**". You can, of course, use **Grid Layout** for it, but Form Layout has form-optimized rules for scaling elements and often gives a better visual effect with less effort.

4.1.5. Splitter

It is a container like **Horizontal** and **Vertical Layout,** but it has an important difference: While, for example, in the **Horizontal Layout,** we define a permanent ratio of the size of elements, the **Splitter** allows a user to modify the size of the elements later. **The splitter** allows you to select the type of layout (vertical/horizontal) by setting the "**Orientation**" property.



Above, we see three widgets, "**Digital IO Indicator,**" arranged horizontally in the Splitter.

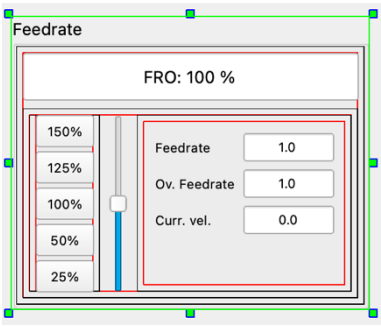
Above is the same container, but after closing the editor and resizing its elements, clicking on the space between widgets and moving the mouse cursor.

With this container, you can give the operator more control over the size of the interface groups, which often translates into a better experience and more convenient work.

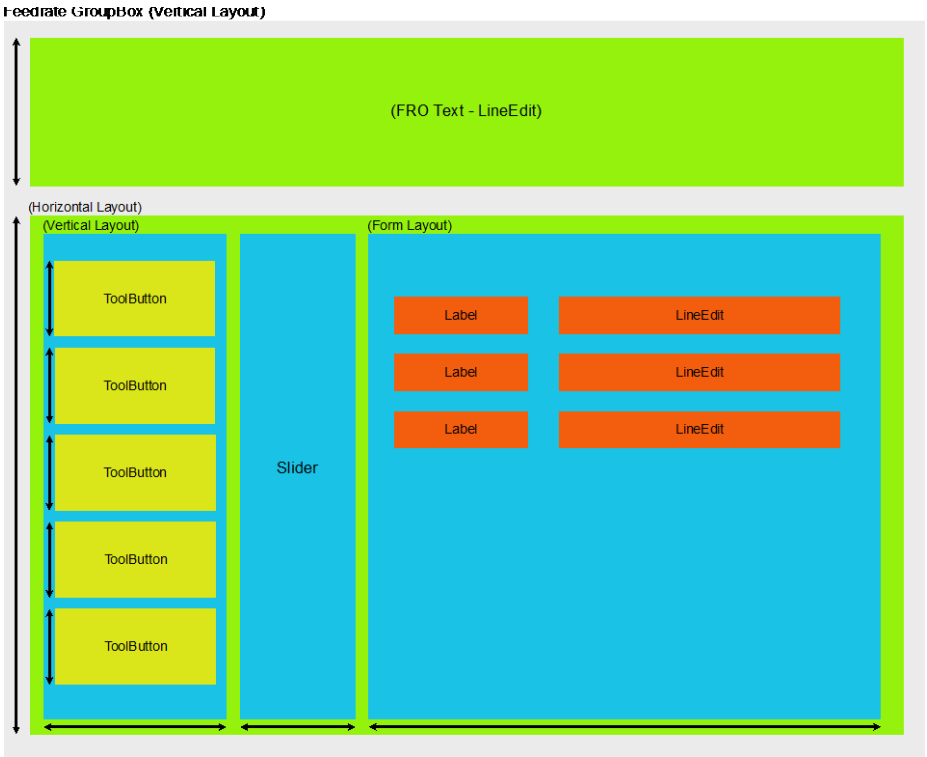
4.2. Containers binding – hierarchical structure

Connecting containers is the basic way to build a complete interface and gives more control over the size of elements in the auto-layout system.

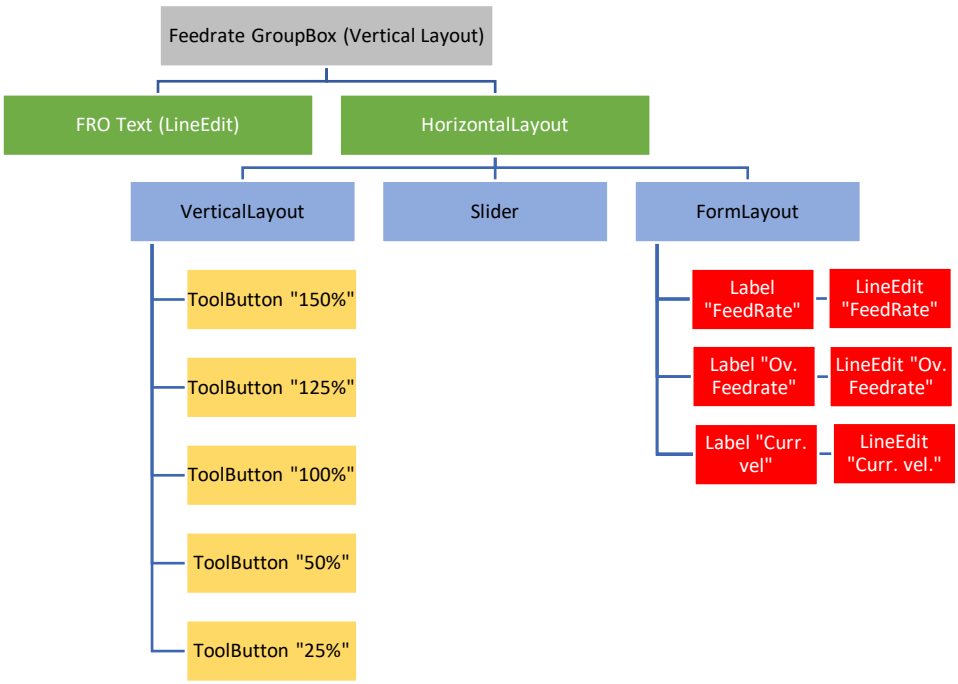
For example, a part of an interface like the following:



When designing this type of block, it is hard to decide at first what kind of layout we should use. The answer is using several containers, as shown in the picture below:



In the form of a tree, it looks like this:



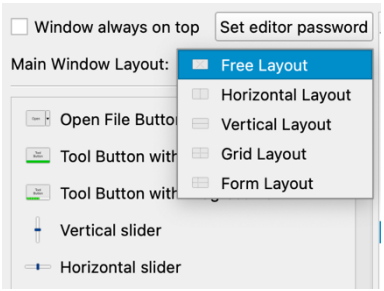
The main container is the **GroupBox** widget with **Vertical Layout**. There are two elements marked in green: the **LineEdit** widget (text "FRO: 100%") and another container with **Horizontal Layout.** In it, in turn, there are three elements marked in blue: a container with 25%-100% buttons (**Vertical Layout),** a speed setting slider **(Slider),** and a container with a form for displaying and editing parameters (**Form Layout**).

At first, it may seem a bit complicated, but such a hierarchical structure has many advantages. Among others, it enforces order in the project and allows you to easily use entire blocks in different places or even other projects. For example, if you want to use a group of buttons from the example above in another project, select only the appropriate container and click CTRL-C. Close the editor, change the screen, enter the edit mode again and click CTRL-V.
With a bit of practice, thinking about the interface in the blocks and groups becomes natural, and the design becomes comfortable and fast.

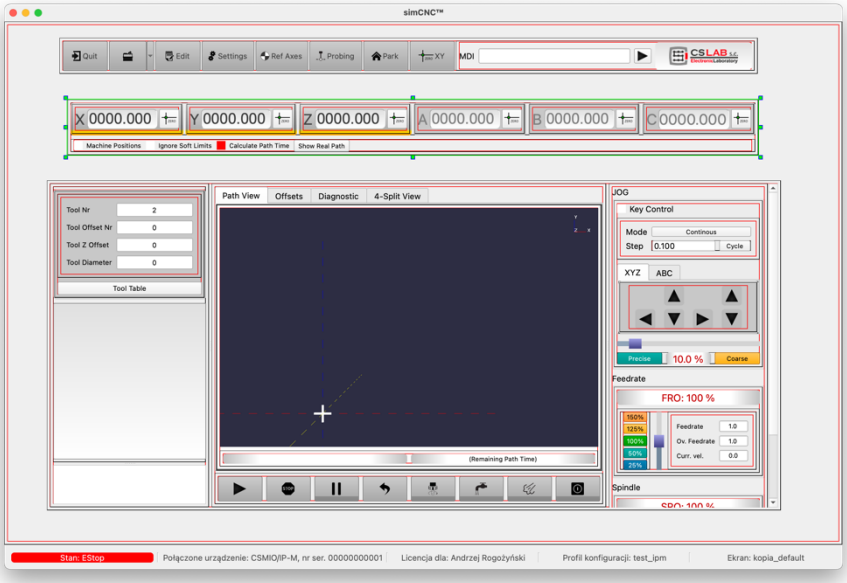
4.3. Main Window Layout

The main simCNC window is also a container for which you can select the type of elements arrangement. Referring to the hierarchical structure discussed in the previous subsection – it is the main container at the very top of the hierarchy.

The type of layout for the main window is selected in the editor window:



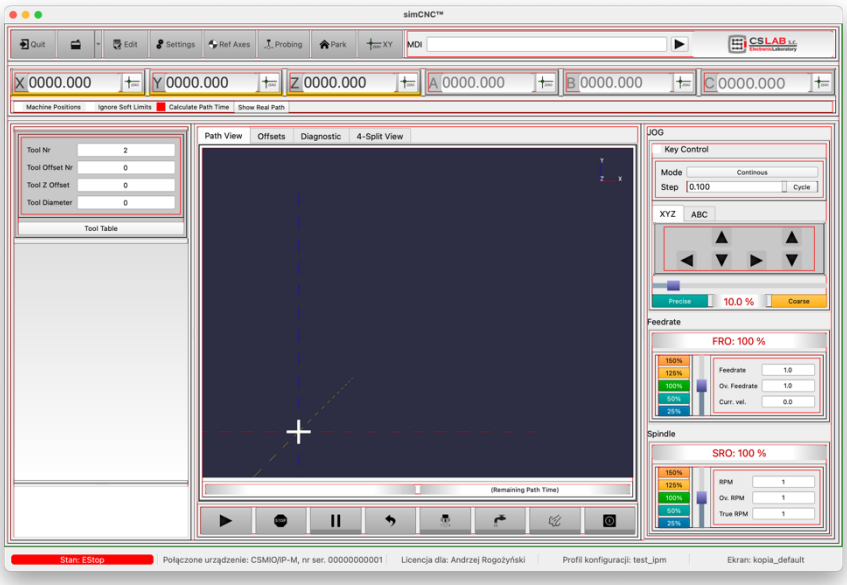
While **Horizontal, Vertical, Grid** and **Form Layout** are already known, "Free **Layout**" has not been mentioned yet. **Free Layout** disables auto-layout. Disabling auto-layout is convenient at the early stage of designing the screen elements or making significant modifications to the design. The choice of the type of layout for the window is made when we have designed the main groups of elements.



In the screenshot above, you can see the design of the default simCNC interface with the auto-placement of the main container disabled.
In this project, we have three main interface groups:

- A group with button bar subgroups, MDI, and logo
- A group with subgroups of axis position and option widgets
- Splitter with subgroups with the rest of the elements

The concept of the project assumes a vertical layout of the main groups. Below you can see the screenshot after setting the "**Vertical Layout**" for the main container. From now on, resizing the window will automatically adjust all content to the new size.



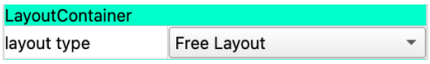
4.4. Widgets containing containers

The following widgets in the interface editor are used to group elements and contain containers with the ability to set the layout type:

- GroupBox
- Frame
- TabBox
- ScrollArea

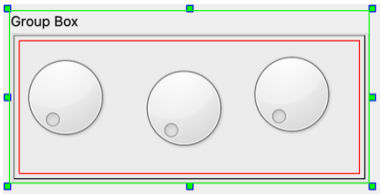
Properties of these widgets and their appearance are shown in the widgets chapter.

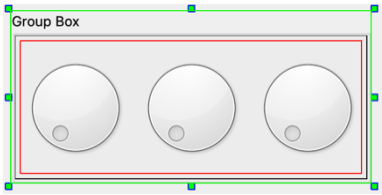
As with the main container, you can disable auto-layout for the widgets by selecting "**Free Layout**".



This can sometimes be useful when predesigning widget content.

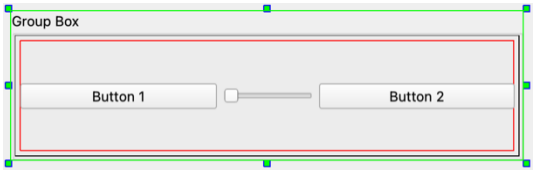
The following is an example of a GroupBox widget with three Dial widgets, before and after enabling **Horizontal Layout**:



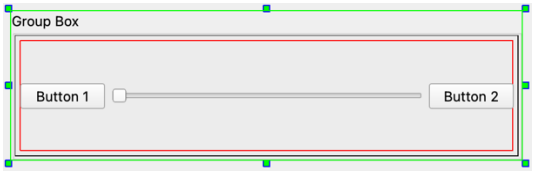


4.5. Space division in containers

There is often a need to define how the auto-layout system should allocate the available space in a container to individual elements. Take, for example, such a group:



Here we see two buttons and a slider in a horizontal layout. At first glance, everything looks correct, but operating the slider would be more precise if it were wider. The simCNC screen editor allows you to accurately control this aspect through the settings of the space division in the container **(stretch)** and the policy of scaling elements **(size policy).**



Above, we can see the same group, but we now have the horizontal size policy set to "Expanding"– i.e., maximum use of available space for the slider.

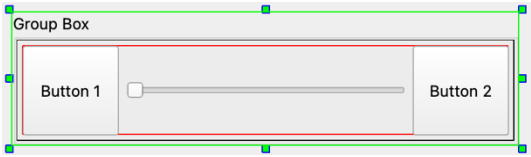
4.5.1. Elements scaling policy settings (size policy)

For each element, you can set the following **vertical size policy** and **horizontal size policy:**

TYPE	DESCRIPTION
Fixed	The size of an element is set rigidly using the **minimum width** and **minimum height** properties. Setting these properties to zero uses the default minimum size.
Minimum	The default size of a widget is its minimum size. The widget can be enlarged, but it is not forced.
Maximum	The default size of a widget is its maximum size. The widget size can be reduced but not below dimensions that make it impossible to use.
Preferred	The widget size can be enlarged or reduced but not below dimensions, making it impossible to use.
Expanding	Forcing an element to take up as much available space as possible. The widget size can also be reduced but not below dimensions that make it impossible to use.
Minimum Expanding	Forcing an element to take up as much available space as possible.
Ignored	If possible, an element will have need space assigned. If a container is too small, the element may be skipped and will not be displayed at all.

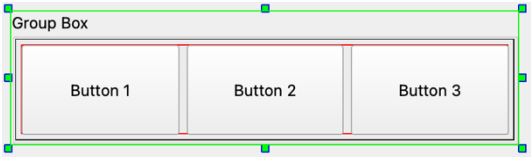
In practice, the most used settings are **Fixed, Preferred,** and **Expanding.**

Below, for example – the same group as before, but the vertical size policy for buttons has been changed from **Fixed** to **Preferred.** As expected, the height of the buttons has been adapted to use the available space. Horizontally, on the other hand, the slider takes up the most space because the buttons have the **horizontal size policy** parameter set to **Preferred** and the slider to **Expanding.**

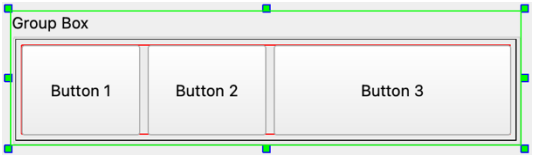


4.5.2. The ratio of elements size in a container (stretch)

This property of containers has already been mentioned when discussing the types of layouts, but as a reminder – the **stretch** parameter can quickly and conveniently control how space should be assigned for elements in a container. So that this parameter works properly, it is best to set **the size policy** for elements to **Preferred (Fixed,** e.g., always forces the default size of an element). Below you can see a group of three buttons in the horizontal layout **(Horizontal Layout).** The **size policy** parameters of the buttons are set to **Preferred,** and in the **stretch** field for the container, there is "1,1,1".

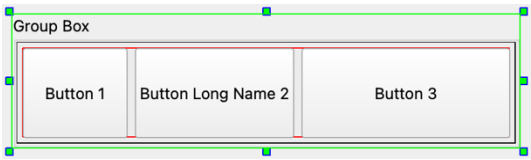


A value of "1,1,1" means that the recommended size of each of the three buttons should be the same. Notice what happens when you change the **container stretch** parameter to "1,1,2":

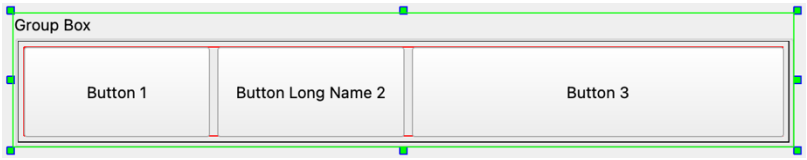


As you can see, the size of the third button is now twice as large. It is worth mentioning that in the **stretch** parameter, the ratio of values matters, not the absolute values - that is, if you set "10, 10, 20" the effect will be identical. Using larger values can be useful to define the division more precisely. Entering "10,10,15" will make the third button 1.5x larger than the others.

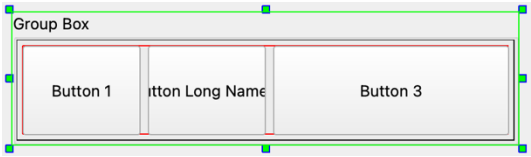
Another thing is that the given values are the recommended ones. Please look at what will happen if the **stretch** parameter is set to "1,1,2", but the description of the second button will be changed to a much longer one.



As you can see, despite the recommended division, the auto-layout system has allocated more space for the second button to fit the text displayed in it. The set division will be applied only when the size of the container is sufficient.



Another option, if we want to keep the desired division, is to set **the horizontal size policy** to **Ignored** for the second button.



As you can see, the change in the scaling policy for the second button caused that the division set by the **stretch** parameter was kept, but the button text was cut.

5. Connection of the graphical interface with simCNC software

Widgets in their properties can have **inputs** and **outputs**. With these properties, we connect the graphical interface simCNC functions and values.

Widget inputs allow you to update its status or content when a value in simCNC changes. As an example, we can use a display of a current axis position: we create a **Label** widget, and as "**Input: text**" select "**Axis X display position**" from the list. When the position of the axis (in this case, the X-axis) changes, the widget text is updated.

Widget outputs allow the graphical interface to activate actions or modify parameters in simCNC. An example here is a machining start button: we create the **ToolButton** widget, and as "**Output: Clicked**" in its properties, select "**Start trajectory**".

5.1. Widget input signals

NAME	DESCRIPTION
Anti-dive delay	Anti-dive delay value for automatic plasma torch height control (THC)
Anti-dive velocity	Anti-dive speed value for automatic plasma torch height control (THC)
Axis (...) abs position	The current absolute position of an axis (machine coordinates)
Axis (...) current work offset	The current working offset of an axis.
Axis (...) display position	Current axis position with the option to switch between machine and software coordinates
Axis (...) prog position	Current programming position of an axis
Axis (...) tool offset	Offset tool in a given axis
Axis (...) tool wear offset	Offset tool wear in a given axis
Axis (...) velocity	Velocity in a given axis
Calculated path time	Calculated estimated execution time of a gcode file
Current spindle speed	Current spindle speed
Current torch voltage	Current plasma torch arc voltage
Current velocity	Current velocity
Feedrate	Set feed rate
Feedrate override	Set feed rate including regulator (FRO)
FRO	Value of a feed rate regulator
GCode file path	Path of a currently loaded gcode file
GCode line number	Number of a gcode file executed line
IO pin value	Hardware I/O pin value
JOG mode	Current JOG mode
JOG speed	Set JOG speed
JOG step	Set step for JOG step mode
Machine param	Value of a machine parameter with a set number
Override spindle speed	Set spindle speed including regulator (SRO)
Remain path time	Estimated time remaining until the gcode file is executed
Screen name	Name of a currently loaded screen
Selected tool nr	Number of a selected tool
Signal value	I/O signal value
Spindle CCW percent	Percentage of current spindle rotation (left rotation)
Spindle CW percent	Percentage of current spindle rotation (right rotation)
Spindle speed	Set spindle speed
Spindle tool nr	Number of a tool loaded in a spindle
SRO	Spindle rotational speed regulator value
THC init position	Saved "Z" coordinate, at which the automatic torch height control function was activated
THC max deviation positive	Maximum range of motion in **the positive** direction for automatic plasma torch height control
THC max deviation negative	Maximum range of negative **movement** for automatic plasma torch height control

THC mode	Selected mode of operation of automatic torch height control
THC position deviation	Current initial value of automatic torch height control – distance from THC init position
THC smart analog amplification	Amplification value for smart-analog mode of automatic torch height control function
THC velocity	Velocity for automatic torch height control
THC voltage deadband	Deadband voltage range for automatic torch height control.
Tool diameter	Diameter of a currently selected tool
Tool diameter wear	Tool wear offset (diameter)
Tool offset number	Number of a selected tool offset
Torch on mode	Plasma torch arc detection mode
Torch on voltage max	Upper voltage limit for plasma torch arc detection function
Torch on voltage min	Lower voltage limit for plasma torch arc detection function
Torch voltage division factor	Divider for measuring plasma torch arc voltage
Torch voltage potentiometer max	Arc voltage value for the maximum position of a cutting height adjustment potentiometer
Torch voltage potentiometer min	Arc voltage value for the minimum position of the cutting height adjustment potentiometer
Torch voltage threshold	Arc voltage threshold value for up/down motion for "analog" and "smart-analog" modes of automatic torch height control
Work offset number	Number of a currently selected working offset

5.2. Widget output signals

NAME	DESCRIPTION
Close simCNC	Close simCNC software
Edit G-Code	Open the gcode edit window
Execute probing script	Run a python macro for tool measuring
JOG (...)+ pressed	Start JOG move in a direction positive for an axis
JOG (...)+ released	Stop JOG move in a direction positive for an axis
JOG (...)- pressed	Start JOG move in a direction negative for an axis
JOG (...)- released	Stop JOG move in a direction negative for an axis
Open settings window	Open the simCNC settings window
Path simulation	Simulation of gcode file, speed analysis, acceleration, etc.
Ref (...) axis	Starting the axis homing procedure
Ref all axes	Starting the homing sequence of axes selected for automatic homing in simCNC settings
Rewind trajectory	Rewind to the beginning of a gcode file
Run script	Runs an indicated python macro
Run spindle clockwise	Switching on the right spindle rotation
Run spindle counter-clockwise	Switching on the left spindle rotation
Set anti-dive delay	Setting the anti-dive delay value of the automatic plasma torch height control function
Set anti-dive velocity	Anti-dive velocity setting of the automatic torch height function
Set axis (...) current work offset	Setting a working offset value in a given axis
Set axis (...) prog position	Modification of the working offset value in a given axis by providing a current value of a program position
Set current tool diameter	Setting a diameter value of a currently selected tool
Set current tool diameter wear	Setting a wear value for the diameter of a currently selected tool
Set feedrate	Setting the feed rate setpoint
Set flood on/off	Turns the flood coolant on/off
Set FRO	Sets a current value of a feed rate regulator
Set IO pin value	Setting a pin state (output)
Set JOG mode	Setting a JOG mode
Set JOG speed	Setting JOG speed (0 – 100%)

Set JOG step	Setting step for JOG step mode
Set machine param	Setting a value of a machine parameter
Set mist on/off	Turns the mist coolant on/off
Set pause on/off	Stops/resumes execution of a gcode file
Set selected tool number	Sets a number of a currently selected tool
Set spindle speed	Sets the set spindle rotation
Set spindle tool number	Sets a number of a tool loaded in a spindle
Set SRO	Sets a current value of the spindle rotational speed regulator
Set THC max deviation negative	Sets the maximum range of motion in the negative direction for the automatic torch height control function
Set THC max deviation positive	Sets the maximum range of motion in the positive direction for the automatic torch height control function
Set THC off	Disables automatic torch height control
Set THC on	Enables automatic torch height control
Set THC smart-analog amplification	Sets an amplification for the "smart -analog" mode of the automatic torch height control function
Set THC velocity	Sets velocity for automatic torch height control
Set THC voltage deadband	Sets a allowed range of fluctuations of a torch arc voltage for which no height correction is performed
Set tool number offset	Sets the working offset number
Set torch on voltage max	Sets the upper arc voltage limit for plasma torch arc detection function
Set torch on voltage min	Sets the lower arc voltage limit for plasma torch arc detection function
Set torch voltage division factor	Sets a value of a plasma torch arc voltage measurement divider
Set torch voltage potentiometer max	Sets a arc voltage setpoint for the maximum potentiometer position
Set torch voltage potentiometer min	Sets a arc voltage setpoint for the minimum potentiometer position
Set torch voltage threshold	Setting a threshold value for the up/down motion of a torch ("analog" and "smart-analog" mode of the automatic torch height control function)
Set work offset number	Sets a working offset number
Show real trajectory	Enables a view of an actual trajectory of movement, taking into account optimization and constant cutting speed.
Start trajectory	Start machining – start of a currently loaded gcode file
Stop trajectory	Stop machining
Switch to EStop/Idle state	Switching simCNC between stop and idle states
Tool table	Displays a tool list window

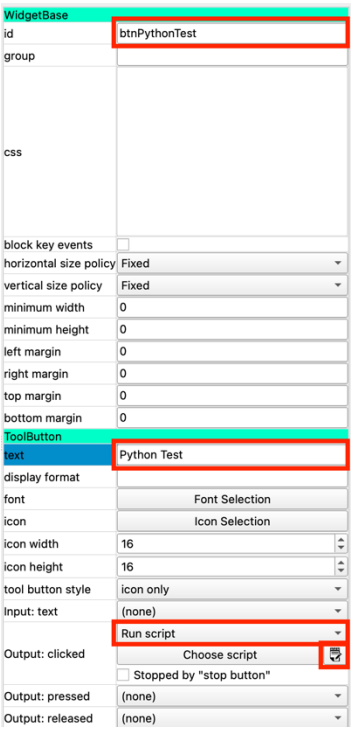
6. Connecting the graphic interface with Python scripts

The simCNC GUI allows you to:

- execute user scripts after clicking a button
- refer to widgets from a script
    - run widget actions
    - read and modify widget properties (e.g., text)

6.1. Activating a script with a screen button

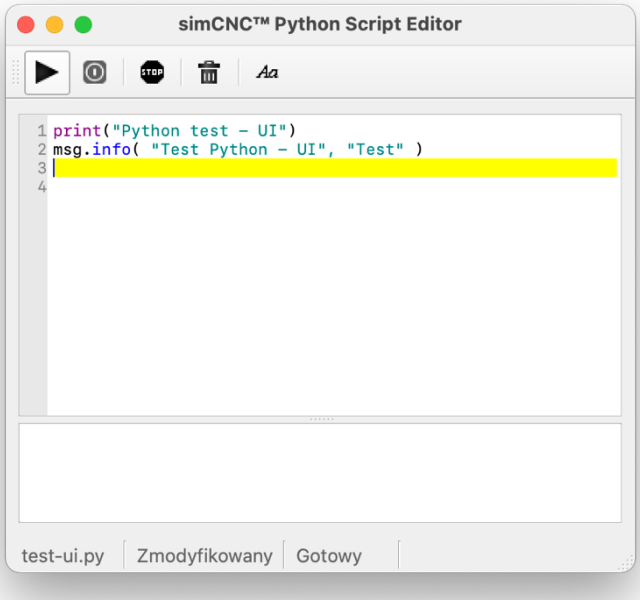
The previous chapter lists the possible output signals of widgets. However, there is often a need to implement a custom function activated by an operator from the graphical interface. Below is an example of how this can be implemented:



We create a button in the interface editor **(ToolButton).** We give it an identifier – e.g. **btnPythonTest. Y**ou can also set a label of the button, e.g., "Python Test".

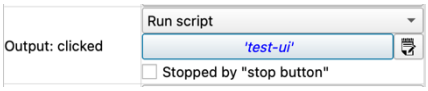
Then, for the **Output: clicked** button, set **Run script.** A script selection window opens, and we should close it because we will create a new file in a moment.

Open the script editor (icon next to the **Choose script** button).



Enter script code in the editor (as above). Our sample script displays a message in the console and an information window.

Save the file, preferably in a catalog with data of the designed screen. In our example, the screen is named "my_test_screen". So we go to the subdirectory "screens/my_test_screen/scripts" relative to the simCNC installation location, and we give the file a name – e.g., "test-ui".



Then we can close the Python editor window, click the Choose script button in the widget properties, and select the "test-ui" file. Keep changes in the project by clicking **Save**. Close the interface editor and done – we have created a button that runs our Python macro. You can check how the button works by clicking on it. In the python console (if we have this widget in the project), the text "Python test – UI" should appear and also a window with the message "Test Python – UI".

6.2. Referencing to interface elements with Python script

The simCNC graphic interface system allows you to refer to widgets from the level of Python scripts. When is it necessary? Imagine that we have a machine tool with automatic tool change, and we want to have a status or error code from the tool changer displayed on our screen. Tool change is most often carried out through the **M6** machine macro. Therefore, the M6 macro code must be able to update the widget text on a screen to display the desired information.

When creating a macro code in simCNC's built-in editor, a class named **gui** is automatically imported, and it has all the graphic elements of a loaded screen in it. We refer to a specific element through the widget identifier (**id** property).

gui.<id>.<nazwa metody>(<argumenty>)

Using the widget example from the previous subsection, if you want to change the text on the button, we do it as follows:

gui.btnPythonTest.setText("Some New Text")

6.2.1. Widget class methods

The widget class provides a programmer with the following methods:

METHOD NAME	DESCRIPTION
executeClickedOutput	Runs an action defined by the left mouse button Arguments: (none) Return value: (none)
executePressedOutput	Runs an action defined by pressing the left mouse button on a widget Arguments: (none) Return value: (none)
executeReleasedOutput	Runs an action defined to release the left mouse button on a widget Arguments: (none) Return value: (none)
executeOutput	Runs an action defined on a specified widget output. Arguments: • Widget output name • Return value: (none)
getAttribute	Get a value of a widget property. Arguments: • Property name Return value: • Property value
getAttributes	Get a list of widget properties Arguments: (none) Return value: • List of widget property names
getOutputs	Get a list of widget output signals Arguments: (none) Return value: • List of widget output names

getText	Get a **text** property of a widget. Arguments: (none) Return value: • Contents of the widget's **text** properties
setAttribute	Setting widget properties Arguments: • Property name • Property value Return value: (none)
setText	Set the **text** property of a widget Arguments: • Value for widget **text** property Return value: (none)

6.2.2. Widget styling change

Using the **setAttribute** method, we can dynamically set a style sheet (css) for the widget. The name of the property is **styleSheet.** The following examples use **btnPythonTest** from subsection 6.1 as the widget ID.6.1

Example 1 (background color change to red):

gui.btnPythonTest.setAttribute(„styleSheet", „background-color: red;")

Example 2 (background color change to red and font color to yellow):

gui.btnPythonTest.setAttribute(„styleSheet", „background-color: red; color: yellow;")

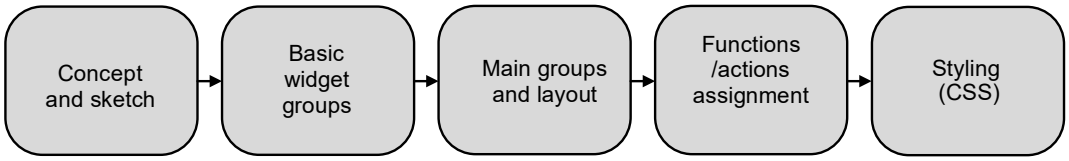
Example 3 (font size change):

gui.btnPythonTest.setAttribute(„styleSheet", „font-size: 24px;")

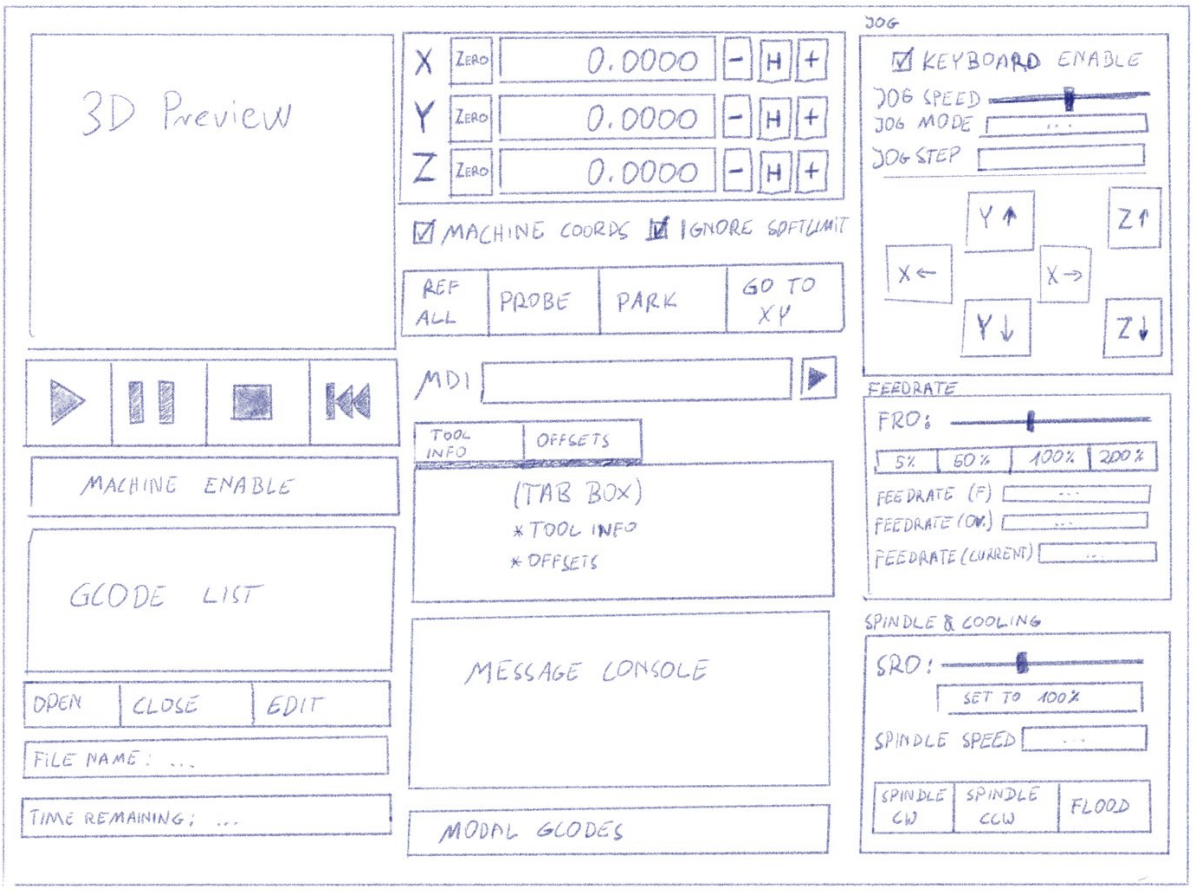
Appendix – Step by step interface project

This appendix shows you how to build a complete interface, step by step – using the information included in the previous chapters.

As a reminder – this will be the order of tasks during construction:



Concept and sketch



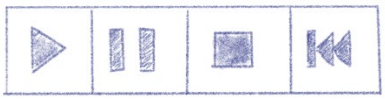
According to the above sketch, we assume the construction of a compact interface for a three-axis milling machine. As you can see, even a freehand drawing is enough. If the purpose of some elements is not clear – don't worry, everything will be explained later.

Creating a new interface project and editing start

- Select from the menu: **Configuration→Set screen**
- In the screen selection window, click the **Create new**
- We give a name, for example, "ui_example"
- Select the name of the newly created interface in the list and click the **Load**
- Select an item from the menu: **Configuration →Open GUI editor**

Basic widget groups

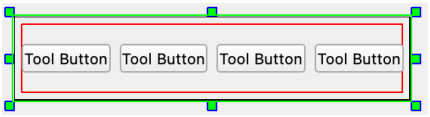
"Start", "Pause", "Stop" and "Rewind" group of buttons



Here we have four buttons in the horizontal position so we will use the **Horizontal Layout** container.

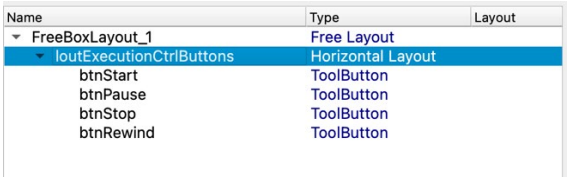
- From the list of widgets in the editor window, drag **Horizontal Layout** to the simCNC window
- From the same list, we drag four **Tool Button** widgets to the container

As a result, we should get something like this:

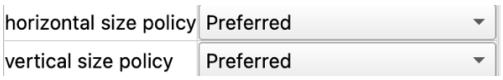


- We change the default names of elements to those that will allow you to find out quickly their purpose later. Here we assumed: **loutExecutionCtrlButtons** for the container and **btnStart, btnPause, btnStop** and **btnRewind** for the buttons. Specify the name in the **id** field of the widget properties.

The object tree in the editor window should look like this:

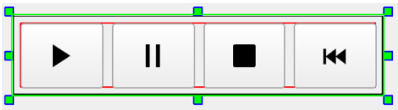


- Next, we change the policy of scaling the buttons so that their size adapts to the size of the container. Click on **btnStart** and while holding down the shift key – click on **btnRewind.** With all four buttons selected, change the **horizontal** and vertical **size policy** properties to **preferred.**

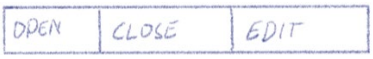


- We set the icons for the buttons by selecting the object and clicking the **Icon Selection** button next to the **widget icon** property (the icons used in this example can be downloaded from here https://soft.cs-lab.eu/ui_example_icons.zip)
    - **btnStart** → „icon_play.png”
    - **btnPause** → „icon_pause.png”
    - **btnStop** → „icon_stop.png”
    - **btnRewind** → „icon_rewind.png”

The look of the group now matches what we wanted to get:



"Open", "Close" and "Edit" group of buttons

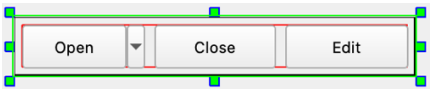


The situation is very similar to the previous one – three buttons in the horizontal layout, except, that for the **Open** button we will use a dedicated widget that will remember the list of recently opened files.

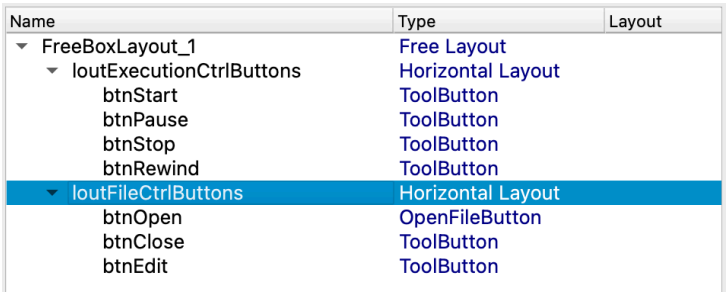
- Drag another **Horizontal Layout** object from the editor window to the simCNC window
- Drag the **Open File Button** widget and two **Tool Button** widgets to the container
- We give names (**id** property) for the container and widgets
    - **loutFileCtrlButtons** for the container
    - **btnOpen**, **btnClose** and **btnEdit** for buttons

- Select all three buttons and change **the horizontal** and **vertical size policy** to **Preferred** so that the button sizes adapt to the size of the container
- We set button labels by modifying the **text** property of widgets
    - "Open", "Close" and "Edit" respectively for **btnOpen, btnClose** and **btnEdit**

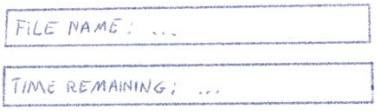
The group is ready and looks like this:



In turn, the tree of objects of our interface should now look like this:



File name and processing time group



The sketch would suggest that there are separate groups here, but remember that the sketch is only a general concept. It is not uncommon for some minor changes to be made during design, whether to get a better look or to simplify the design. This is not a problem if we do not completely change the concept. In this case, it would be good to make a new sketch.

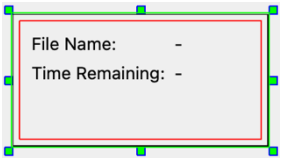
- Drag the **Form Layout** object from the editor window to the simCNC window
- We give the name of the added container (**id** property) on **loutFileInfo**
- We set the **container's vertical size policy** to **Maximum** – thanks to this, we will avoid unnecessary enlargement of the widget when scaling the window
- We drag four **Label** elements into the added container and give them names (**id**) according to the table below:

lbFileNameDesc	lbFileName
lbTimeRemainingDesc	lbTimeRemaining

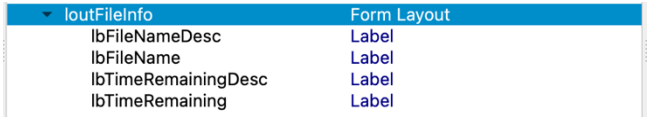
- We change the **text** property of the added labels according to the following table:

File Name	-
Time Remaining	-

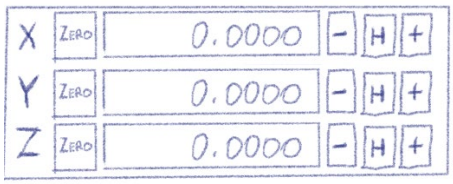
The group is ready and should look like the following:



And this is how it should look like in the object tree:



Axis position indicator group



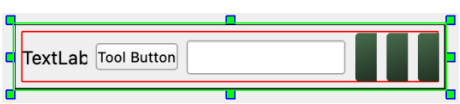
For each of the axes, we have here a group of six widgets in a horizontal layout:

- Axis name (**Label**)
- Program position reset button (**Tool Button**)
- A field for displaying and editing items (**Line Edit**)
- Three limit switch and homing indicator lights (**Digital IO indicator**)

We start by creating a group for one axis:

- Drag the **Horizontal Layout** container from the editor window to the simCNC window
- We drag widgets into the container one by one, placing them from the left to the right side of the container:
  - **Label**
  - **Tool Button**
  - **Line Edit**
  - 3x **Digital IO Indicator**

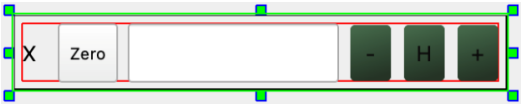
We should get something like this:



- Set **horizontal** and **vertical size policy** for the button (**Tool Button**) to **Preferred**
- Set **text property** for the button to „Zero”
- Set **vertical size policy** for edit field (**Line Edit**) to **Preferred**

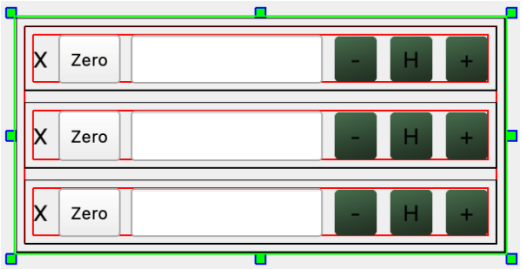
- Set **text** property of Label object to **X**
- Set **text** property of **Digital IO indicator** lights consecutively (from left) to „-”, „H” and „+”

We have a group of one axis almost ready:



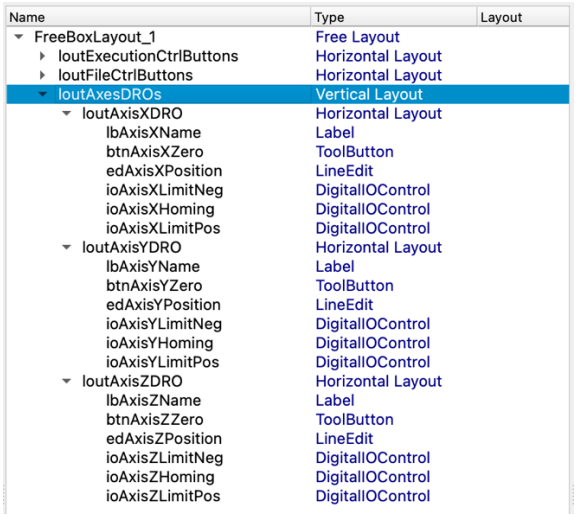
- Drag the **Vertical Layout** container from the editor window to the simCNC window
- Select the container in which we have just added widgets and press CTRL-X (cut)
- We select the empty **Vertical Layout** container that we added and press CTRL-V (paste) three times

The effect should look like the following:



- We set the widget names (**id** property)
  - X axis group – container name: **loutAxisXDRO** and widget as follows, from left:
    - **lbAxisXName**
    - **btnAxisXZero**
    - **edAxisXPosition**
    - **ioAxisXLimitNeg**
    - **ioAxisXHoming**
    - **ioAxisXLimitPos**
  - Y and Z axis group – same as X-axis, only in the names we change "AxisX" to "AxisY" and "AxisZ" respectively
  - Parent container name **(Vertical Layout): loutAxesDROs**

The object tree should look like this:



Remember to save the project from time to time by pressing CTRL-S or clicking the **Save** key in the editor window.

„Machine coords" and „Ignore Soft Limit" check box group

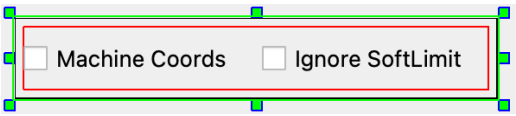
A group of two **Checkbox** widgets in a horizontal layout:

- switching between displaying machine and software coordinates **(Machine Coords)**
- disable software limits **(Ignore SoftLimit)**

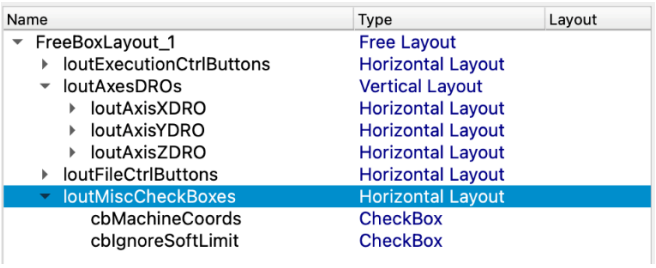
We start by adding a container **(Horizontal Layout)** as standard:

- Drag the **Horizontal Layout** container from the editor window to the simCNC window
- We drag two **Checkbox** widgets to the container
- We set the names **(id** property)
    - Container: **loutMiscCheckboxes**
    - **cbMachineCoords** and **cbIgnoreSoftLimit** for widgets
- Set the display text **(text** property) for **Checkbox** widgets
    - „Machine Coords" and „Ignore SoftLimit"

As a result, the group should look like the following:



And this is how the group should look in the tree of project objects:



„Ref All", „Probe", „Park" and „Go To XY" group of buttons

Here we have a group of four buttons **(Tool Button)** in a **Horizontal Layout.**

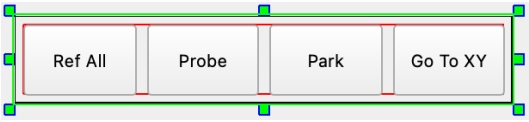
- „Ref All" → axis homing
- „Probe" → tool measurement
- „Park" → moving machine axis to park position
- „Go To XY" → moving X, Y axes to program zero

Again, we start by adding a container:

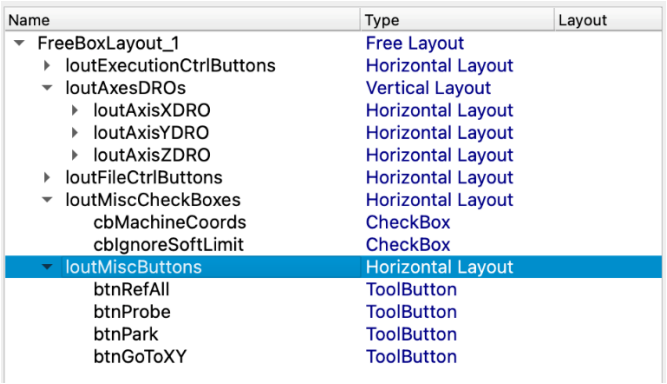
- Drag the **Horizontal Layout** container from the editor window to the simCNC window
- Drag four **Tool Buttons** to the container

- Select (shift-clicked) button widgets and change their horizontal and **vertical size policy** properties to **Preferred** so that their size automatically adjusts to the container
- Set the object names **(id** property):
  - Container name: **loutMiscButtons**
  - Widgets name: **btnRefAll**, **btnProbe**, **btnPark**, **btnGoToXY**
- We set the **text** property of the buttons:
  - From left: „Ref All", „Probe", „Park", „Go To XY"

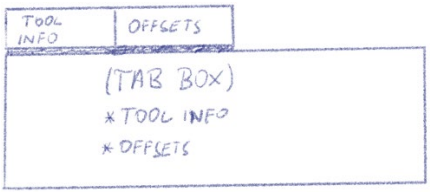
The effect should be as follows:



And this is what a group in the object tree should look like:

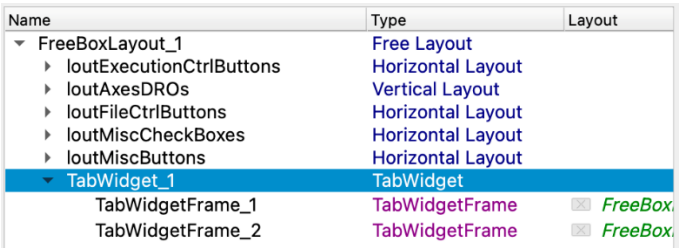


Widget with „Tool Info" and „Offsets" tabs

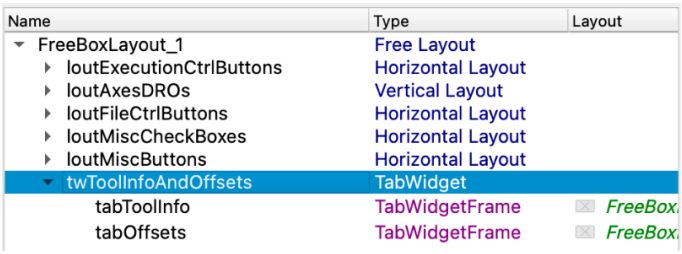


- Drag the **Tab Box** widget from the editor window to the simCNC window
- Select the widget and set the number of bookmarks **(tabs quantity** property) to "2".

In the object tree, we see three new objects: **TabWidget**, which controls the display and switching of tabs, and two **TabWidgetFrame** objects, which are containers of individual tabs – in them that we will place widgets.

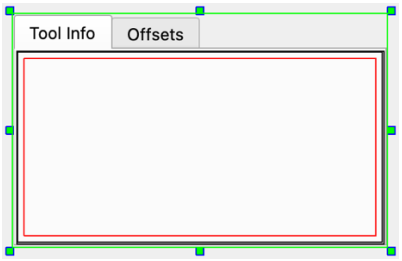


- We give names (property **id):**
  - Object **TabWidget**: **twToolInfoAndOffsets**
  - Tabs: **tabToolInfo** oraz **tabOffsets**

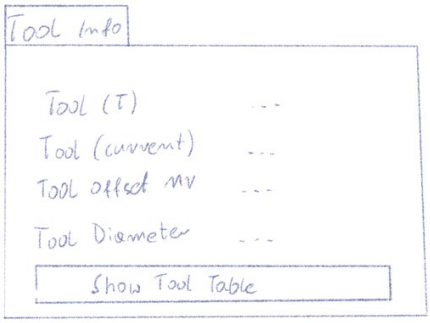


- Set labels for tabs (**title** property) - You must first select the appropriate tab in the object tree, e.g., by clicking on **tabToolInfo:**
    - „Tool Info" for **tabToolInfo** and „Offsets" for **tabOffsets**

Our widget should look like this at this point:



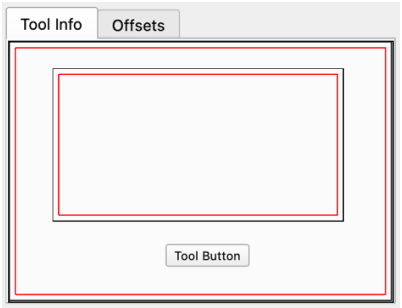
Now we can proceed to create the content of the tabs. The following is a concept of the **Tool Info** tab:



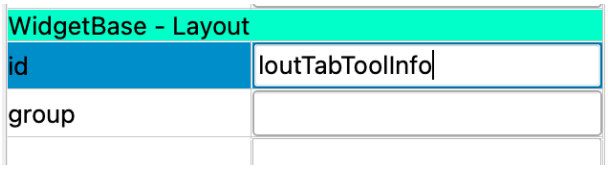
Here we can see several **Label** objects – descriptions and displayed values (three-dot on the sketch) and one **Show Tool Table** button to open the tool table window. **Label** objects will be placed in a form (**Form Layout),** while the tab will have a vertical arrangement (**Vertical Layout)** containing the form and a button (**Tool Button).**

- Drag **Form Layout** from the editor window to the **Tool Info** tab
- Drag the **Tool Button** from the editor window to the **Tool Info** tab, placing it below the Form **Layout** container

This should look like this:



- Click **tabToolInfo** in the object tree and set the **layout type** property to **Vertical.** This defines the placement of objects in the tab.
- Set the name**(id)**of the tab container to **loutTabToolInfo**

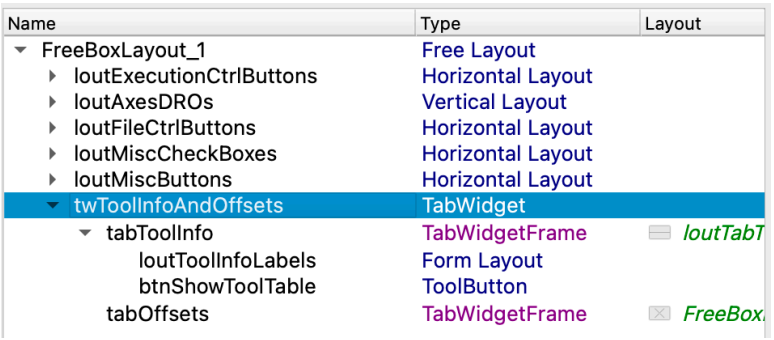


- We give names (**id**) for the Form **Layout** container and for the button
  - Container: **loutToolInfoLabels**
  - Button: **btnShowToolTable**

- Set the **horizontal size policy** property for the button to **Preferred**
- We set the **text** property for the button to "Show Tool Table"

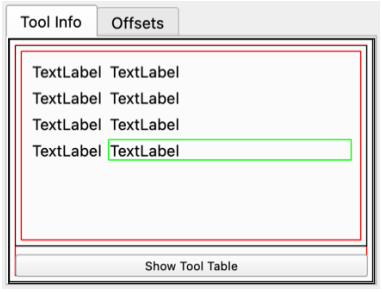
At this time, the widget should look like this:

And this is how it should look in the object tree:



We can proceed to place **Label** objects in the **loutToolInfoLabels** container.

- We drag eight **Label** objects to the **loutToolInfoLabels** container. We place them in a way corresponding to the sketch, i.e., four rows of two widgets.



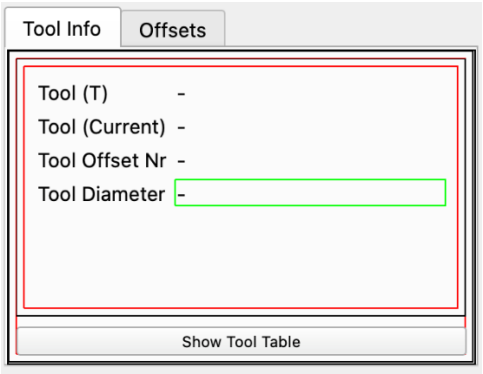
- We give the **Label** widgets a name**(id),** according to the table below

lbSelectedToolDesc	lbSelectedTool
lbCurrentToolDesc	lbCurrentTool
lbToolOffsetNrDesc	lbToolOffsetNr
lbToolDiameterDesc	lbToolDiameter

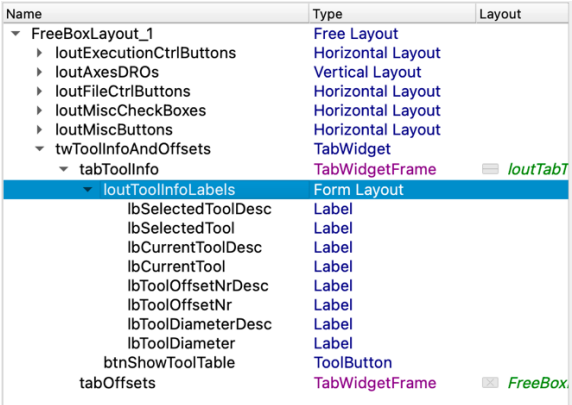
- We give **Label** widgets a **text** property according to the following table

Tool (T)	-
Tool (Current)	-
Tool Offset Nr	-
Tool Diameter	-

The widget should now look like this:



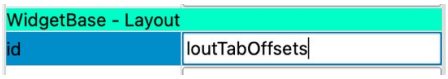
And so the tree of objects:



The "Offsets" tab remains. Here the matter is simpler because the tab contains only one widget of the **Offsets Table** type.

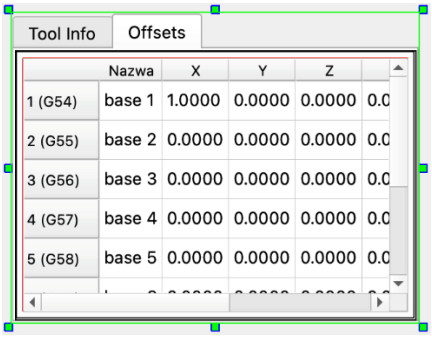
- Click on **tabOffsets** in the object tree
- We set the layout type to **Vertical** (Pionowe). The type of layout does not matter much here because only one widget will be placed in the container, but you must choose one so that the size of the widget automatically adapts to the size of the container).

- Set the name **(id)** for the tab container to **loutTabOffsets**

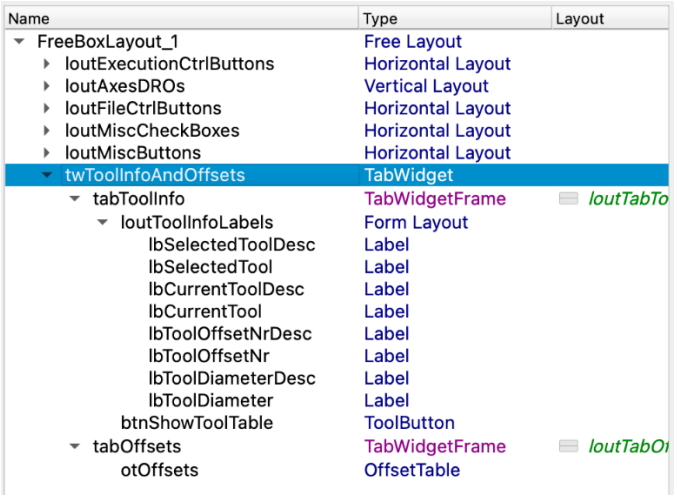


- In the simCNC window, click on the "Offsets" tab to activate it
- Drag the **Offset Table** widget to the "Offsets" tab from the editor window
- We give the name (**id**) to the **Offsets Table** widget on **otOffsets**

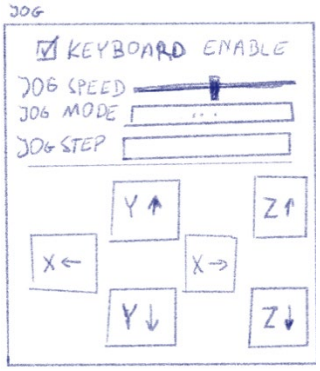
The widget is visually ready and should now look like the following:



In the object tree, the complete widget should look like this:



„JOG" group



From the top, we can see in the sketch:

- **Check Box** for allowing axis control from the keyboard
- **Label** and **Horizontal Slider** to set the jog speed
- **Label** and **Tool Button** to change jog mode
- **Label** and **Tool Button** to select the JOG step
- Separation line (**Frame**)
- A group of buttons **(Tool Button)** arranged in a grid (**Grid Layout)** to control individual axes

We will use the **Group Box** container widget here. The following are the design steps in turn:

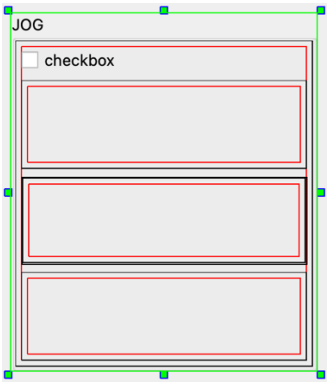
- Drag the **Group Box** widget from the editor window to the simCNC window
- Set the name**(id)** of the widget to **gbJog**
- Set the vertical arrangement – **layout type** property to **Vertical (Pionowe)**

- Set the name**(id)**of the container to **loutJog**

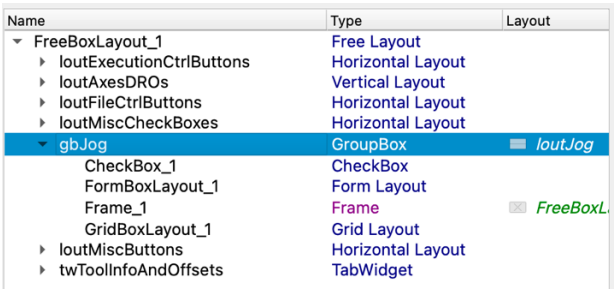
- Set the group label **(title** property) to "JOG"
- We drag the following widgets from the editor to the created group, placing them successively from the top:
    - **Check Box**
    - **Form Layout**
    - **Frame**
    - **Grid Layout**

  When placing widgets in a container with Auto-Layout enabled, pay attention to the tags that appear. They show where the new object will be located.

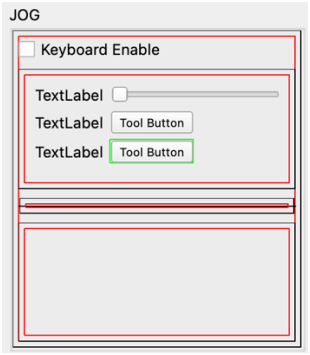
This is how the widget should look like now:



And so, the tree of objects:



- Set names (id) of the added objects, one by one, from the top: **cbKeyboardJogEnable, loutJogConfig, frJogLine** and **loutJogButtons**
- We set the **text** property of the **cbKeyboardJogEnable** widget to "Keyboard Enable"
- Set the **shape** property of the **frJogLine** widget to **Horizontal Line**
- Set the **vertical size policy** property of the **frJogLine** widget to **Maximum**
- To the **loutJogConfig** container we drag from the editor three widgets **Label** and **Horizontal Slider** and two buttons (**Tool Button),** placing them according to the sketch



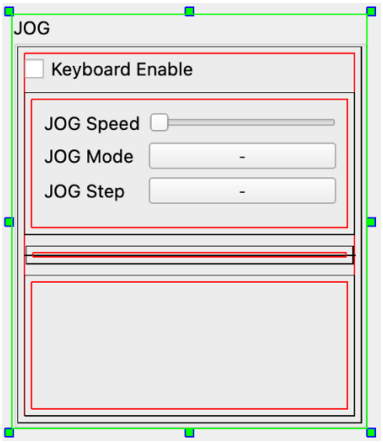
- We give the new widgets a name (**id**) according to the table below:

lbJogSpeedDesc	slJogSpeed
lbJogModeDesc	btnJogMode
lbJogStepDesc	btnJogStep

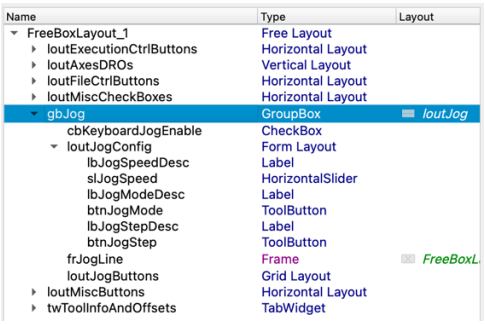
- We set the widgets text property according to the table below:

JOG Speed	(not applicable)
JOG Mode	-
JOG Step	-

- For **the btnJogMode** and **btnJogStep** widgets, we change the **horizontal size policy** property to **Preferred** to match the size of the container horizontally

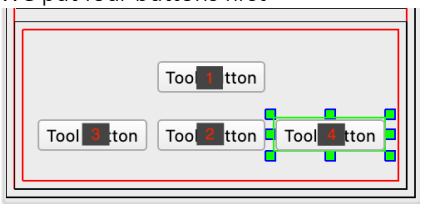


And the objects tree like this:

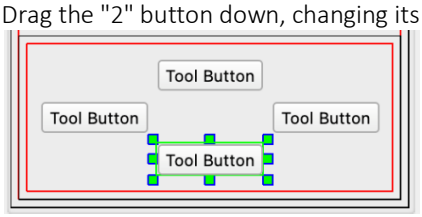


- We add six **Tool Buttons** to the **loutJogButtons** group. We place them according to the sketch. Again, pay attention to the displayed tags that show where the added widget will be "dropped". For this group, the easiest way to proceed is in the following order:

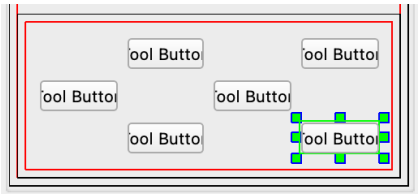
  - We put four buttons first



  - Drag the "2" button down, changing its position in the grid



  - Add the last two buttons



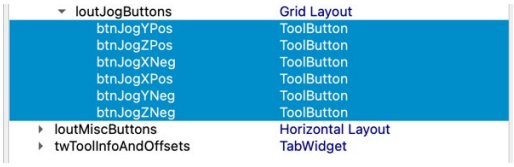
- We give names (**id**) to the buttons according to the table below

	btnJogYPos		**btnJogZPos**
btnJogXNeg		**btnJogXPos**	
	btnJogYNeg		**btnJogZNeg**

- We set the **text** property for the buttons according to the following table: (arrows are *unicode* characters, for ease of use, they can be copied and pasted)

	Y↑		Z↑
←X		X→	
	Y↓		Z↓

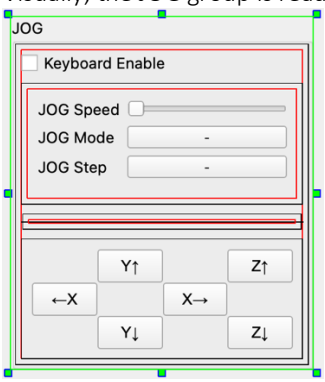
- In the object tree, select all **btnJog buttons...** and set the **horizontal** and **vertical size policy** properties to **Preferred**



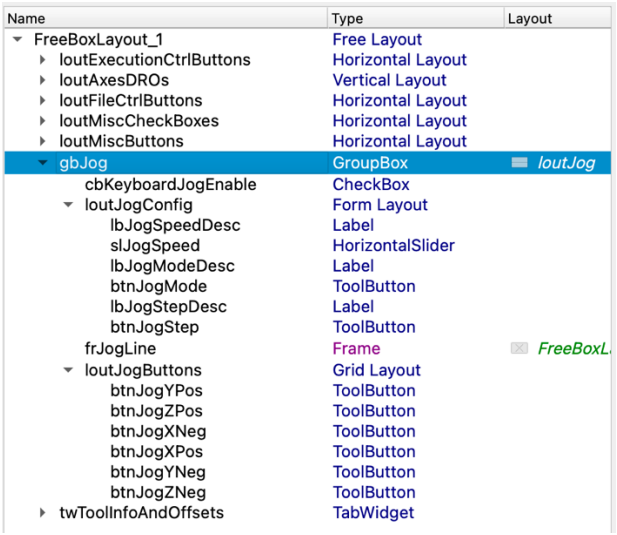
- Set the **buttons btnJog ...** to a slightly larger font (font property) – e.g. "Arial 14"
- Select the **loutJogButtons** object and set the following properties to "0":

  - **left, right, top** i **bottom margin**
  - **horizontal** i **vertical spacing**

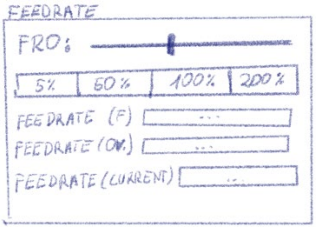
Visually, the JOG group is ready and should look like below:



And this is how the JOG group should look in the object tree:



„Feedrate" group



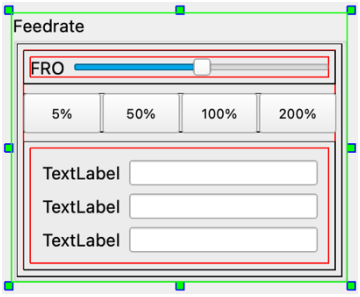
In the sketch above we have the following elements (from the top):

- **Label** and **Horizontal Slider** for machining speed correction (FRO)
- The group (**Horizontal Layout**) of buttons (**Tool Button**) allows you to set frequently used FRO values quickly
- **Label** and edition area (**Line Edit**) to display and change the set machining speed
- **Label** and edition area (**Line Edit**) to display the machining speed with FRO correction applied
- **Label** and edition area (**Line Edit**) to display the resultant current speed of a machine axis

As previously, we will use the **Group Box** container widget.

- Drag the **Group Box** widget from the editor window to the simCNC window
- Set the vertical type of layout by changing the **layout type** property to **Vertical**
- We give names (**id**) for the widget and the container contained in it: **gbFeedrate** and **loutFeedrate**

- We change the group label **(title** property**)** to "Feedrate"
- We add the following containers to the container, placing them from top to bottom:
  - 2x **Horizontal Layout**
  - **Form Layout**
- We give them names (**id**): **loutFroSlider**, **loutFroButtons**, **loutFeedrateInfo**
- To the container **loutFroSlider** we add widgets **Label** and **Horizontal Slider** and we give them names **lbFroDesc** and **slFro**
- We change **text properties** of **lbFroDesc** widget to „FRO"
- We add four buttons (**Tool Button)** to the **loutFroButtons** container and give them names **(id): btnFro5, btnFro50, btnFro100, btnFro200**
- We change the **text** property of the buttons respectively to "5%", "50%", "100%" and "200%"
- We change the **horizontal size policy** property of the buttons to **Preferred**
- We change the **vertical size policy** property for the **loutFroSlider** container to **Maximum** (this cause the container is not stretched vertically)
- We change the **vertical size policy** property for the **loutFroButtons** container to **Maximum**
- We add three **Label** widgets and three **Line Edit** widgets to the **loutFeedrateInfo** container, placing them as in the sketch: **Label** on the left, **Line Edit** on the right



- We give the widgets a name (**id**) according to the following table:

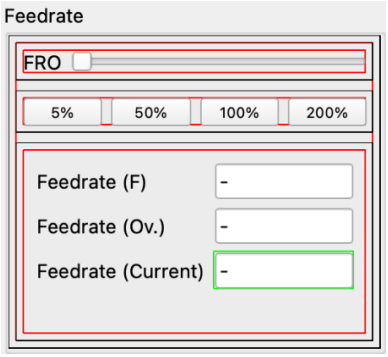
lbFeedrateDesc	edFeedrate
lbFeedrateOvDesc	edFeedrateOv
lbFeedrateCurrentDesc	edFeedrateCurrent

- We change the widgets' **text** property according to the table below

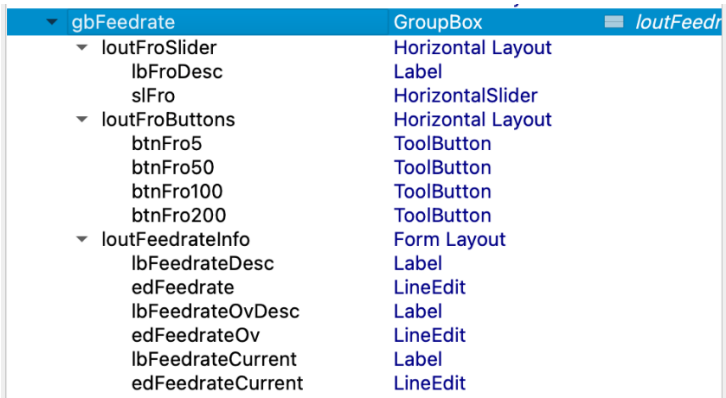
Feedrate (F)	-
Feedrate (Ov.)	-
Feedrate (Current)	-

- Set the **read only** property for the **edFeedrateOv** and **edFeedrateCurrent** widgets

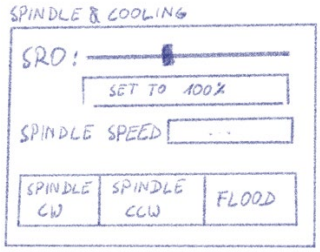
The "Feedrate" group is visually ready and should look like this:



And this is how it should look like in the object tree:



„Spindle & Cooling" group



In the sketch above, we have the following elements (from the top):

- **Label** and **Horizontal Slider** for spindle speed correction (SRO)
- Spindle speed correction reset button (**Tool Button**)
- **Label** and **Line Edit** to read and modify the set spindle rotation
- Three buttons (**Tool Button**) for switching on a spindle and coolant

As before – we will use the Group **Box** container widget.

- Drag the **Group Box** widget from the editor window to the simCNC window
- Set the vertical type of layout by changing the **layout type** property to **Vertical**
- We give names (**id**) to the widget and the container it contains: **gbSpindleAndCooling** and **loutSpindleAndCooling**

WidgetBase	
id	gbSpindleAndCooling

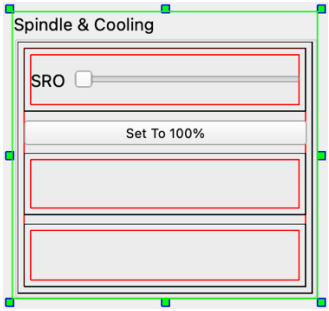
WidgetBase - Layout	
id	loutSpindleAndCooling

- Change the group label (**title** property) to "Spindle && Cooling" (The "&" character is a special character, so to have it displayed correctly, you must enter it twice)

GroupBox	
title	Spindle && Cooling

- We add the following elements to the container, placing them from top to bottom
  - **Horizontal Layout**
  - **Tool Button**
  - 2x **Horizontal Layout**

- We give names (**id**): **loutSroSlider**, **btnSroReset**, **loutSpindleInfo**, **loutSpindleAndCoolingCtrl**
- To the **loutSroSlider** container add **Label** and **Horizontal Slider**
- We give names (**id**): **lbSroDesc** and **slSro**
- Set the **text** property of the **lbSroDesc** widget to "SRO"
- Set the **text** property of the **btnSroReset** widget to "Set to 100%"

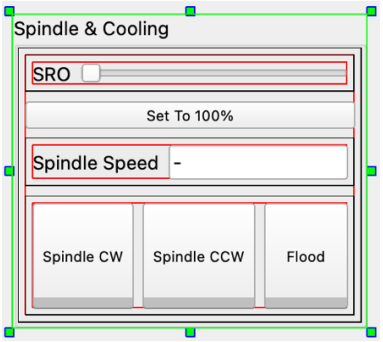


- Add **Label** and **Line Edit** to the **loutSpindleInfo** container

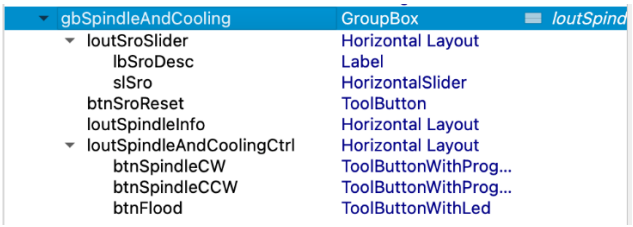
- We give them names (**id**): **lbSpindleSpeedDesc** and **edSpindleSpeed**
- We set the **text** property of the added widgets to "Spindle Speed" and "-".
- To the bottom **container loutSpindleAndCoolingCtrl** we add elements, from the left:
    - 2x **Tool Button with Progress Bar**
    - **Tool Button with LED**

- We give the added buttons a name**(id): btnSpindleCW**, **btnSpindleCCW** and **btnFlood**
- Set the **text** property for the buttons: "Spindle CW", "Spindle CCW" and "Flood"
- We change the **horizontal** and vertical **size policy** property buttons to **Preferred**
- Select the **LED visible** property for the **btnFlood** button
- We change the **vertical size policy** property of the **loutSroSlider** and **loutSpindleInfo** containers to **Maximum**

The "Spindle & Cooling" group visually is ready and should look like this:



And this is what it should look like in the object tree:

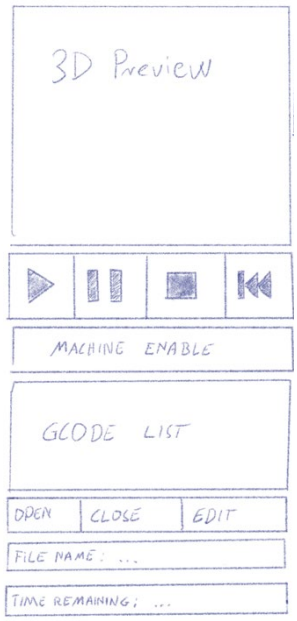


Main groups and layout

With all the basic widget groups created, we can start designing main groups. In the sketch at the beginning of the chapter, you can see that the design is made of three columns of widgets, and in each column, the elements are arranged vertically.

The left column

In the sketch, we see the left main group of widgets. It contains (from the top):

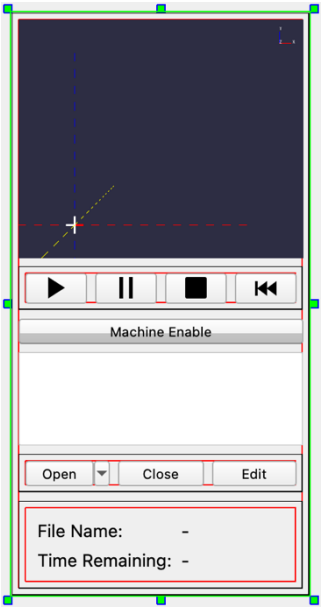


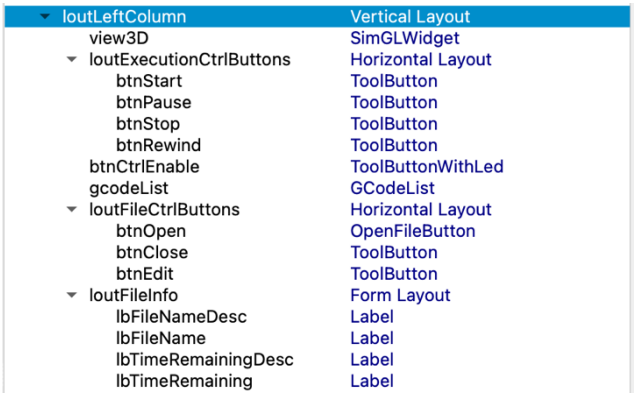
- 3D path preview (**Path View**)
- **loutExecutionCtrlButtons** buttons group
- A button to put a machine in standby mode (**Tool Button with LED**)
- Preview of a file in text form – list of G-Codes (**GCode List**)
- **loutFileCtrlButtons** buttons group
- File name and forecast processing time group - **loutFileInfo**

List of design operations:

- Drag the **Vertical Layout** container from the editor window to the simCNC window
- We give a name (**id**) to the container: **loutLeftColumn**
- We drag the following elements into the container, arranging them from top to bottom:
  - **PathView**
  - The pre-designed group **loutExecutionCtrlButtons**
  - **Tool Button with LED**
  - **GCode List**
  - The pre-designed group **loutFileCtrlButtons**
  - The pre-designed group **loutFileInfo**
- We give names (**id**) to **PathView**, **ToolButton with LED** and **GCode List** as follows: **view3D**, **btnCtrlEnable** and **gcodeList**
- We set the **LED visible** property for the **btnCtrlEnable** button
- We set **horizontal** and **vertical size policy property** for the **btnCtrlEnable** button to **Preferred**
- We set **text** property of the same button to „Machine Enable”

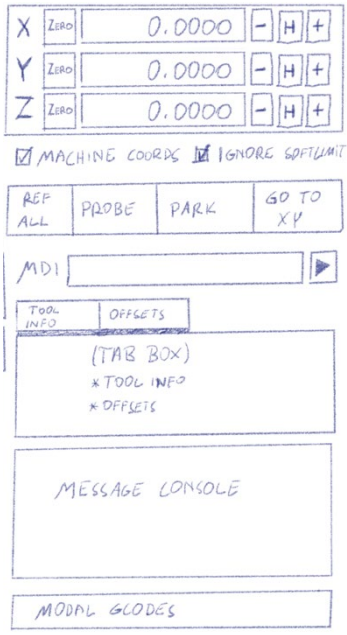
The left column of widgets is ready and should look like this:





The central column

In the sketch we see the central main group of widgets. It contains (from the top):

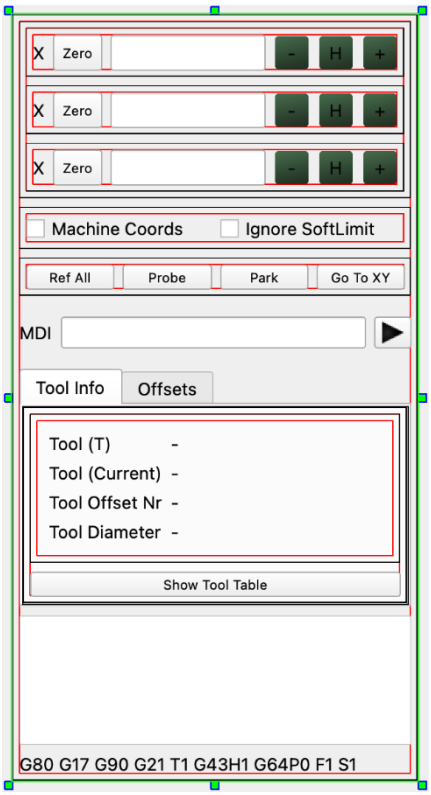


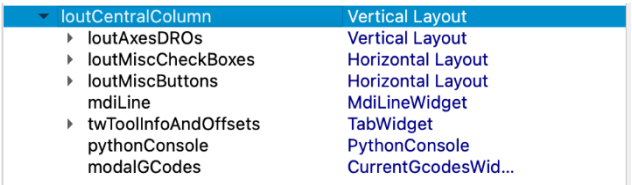
- A group of axis indicators - **loutAxesDROs**
- A group of check boxes **loutMiscCheckBoxes**
- A group of buttons **loutMiscButtons**
- A widget for manual input of **MDI line** machine commands
- A tabbed widget - **twToolInfoAndOffsets**
- Message Console - **Python Console**
- A modal G-codes list widget - **CurrentGCodesWidget**

List of design operations:

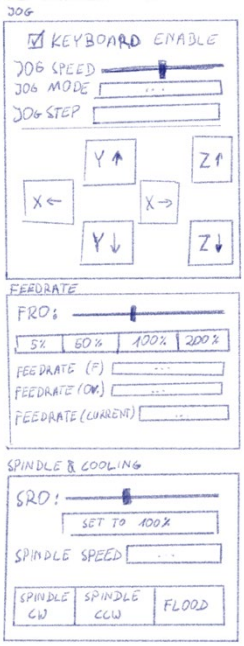
- Drag the **Vertical Layout** container from the editor window to the simCNC window
- We give a name (**id**) to the container: **loutCentralColumn**
- We drag the following elements into the container, arranging them from top to bottom:
    - The pre-designed group **loutAxesDROs**
    - The pre-designed group **loutMiscCheckBoxes**
    - The pre-designed group **loutMiscButtons**
    - **MDI line**
    - The pre-designed widget **twToolInfoAndOffsets**
    - **Python Console**
    - **Current g-codes**
- We name **(id)** the **MDI line**, **Python Console,** and **Current g-codes** widgets as follows: **mdiLine**, **pythonConsole**, **modalGCodes**

The main central group of widgets is ready and should look like this:





The right column



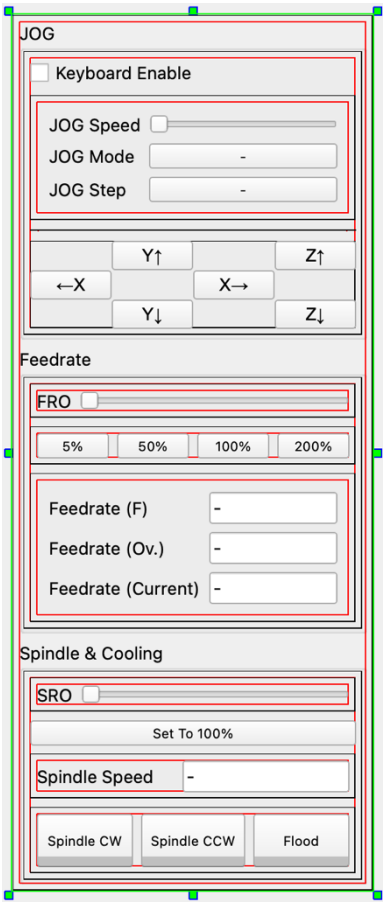
In the sketch we see the right main group of widgets. It contains (from the top):

- A group of manual control - **gbJog**
- A group of federate control - **gbFeedrate**
- A group of spindle and coolant control - **gbSpindleAndCooling**

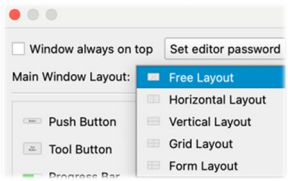
List of design operations:

- Drag the **Vertical Layout** container from the editor window to the simCNC window
- We name (**id)** the container: **loutRightColumn**
- We add the following elements to the container, arranging them from top to bottom:
  - The pre-designed group **gbJog**
  - The pre-designed group **gbFeedrate**
  - The pre-designed group **gbSpindleAndCooling**

The right main group is ready and should look like this:



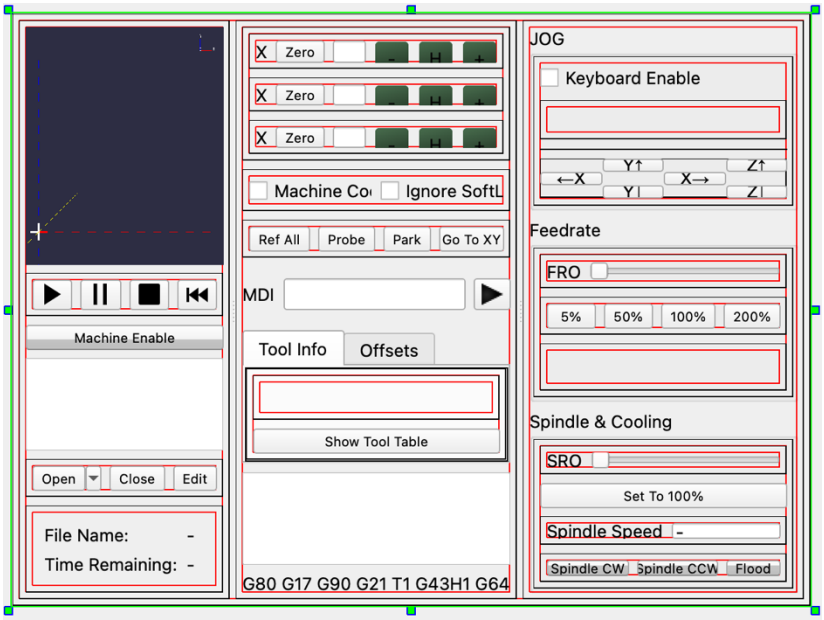
The main container layout



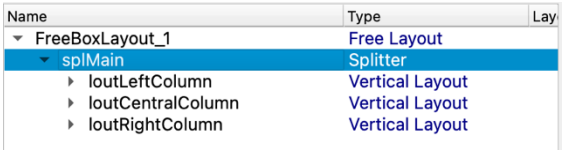
With all the three main groups of our interface ready, it's time to choose the type of layout in the main window. As you can see in the picture on the right, for the main container we can choose between the four types of layouts: horizontal, vertical, in the grid, and in the form. In this case, however, we will use the **splitter** so that an operator can easily change the division of space between the three columns of the interface. There is no **Splitter** in the available options, but you can get around it in a very simple way.

Follow the steps:

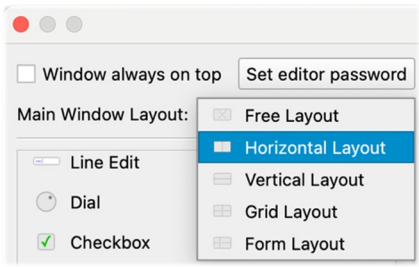
- Drag the **Splitter** object from the editor window to the simCNC window
- Give it a name (**id**): **splMain**
- Drag the following groups **loutLeftColumn**, **loutCentralColumn** and **loutRightColumn** to the **splMain** container. We drop them at the right edge to keep the desired column order.



- Pay attention to the object tree, whether the correct hierarchy is preserved.



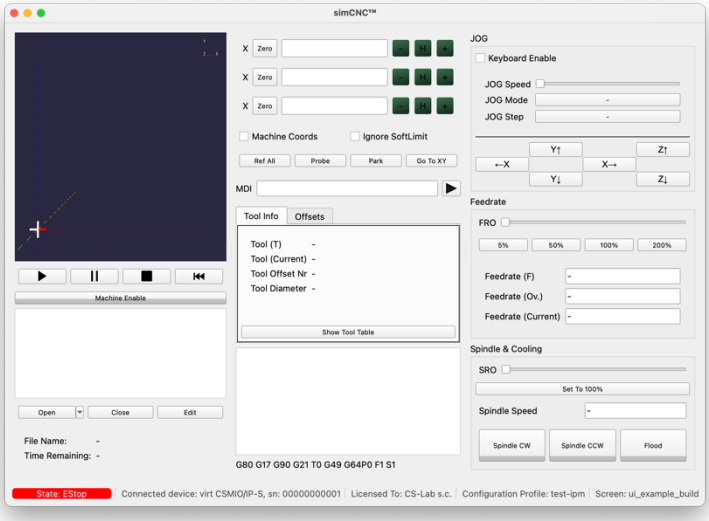
- Set the **Horizontal Layout** for the main window:



Since after adding the **splitter**, in the main window we really have only one group **(splMain),** it does not matter whether we choose horizontal or vertical layout. The **splMain** group will simply be adjusted to the size of the window anyway.

- In the object tree, select the **Horizontal Layout** object at the very top and give it a name (**id**): **loutMain**

We can now close the editor window for a moment (remember to keep the changes) and look at the effects of your work done so far. The program window should look like this:



As you can see, the appearance of the design is in line with our concept. It could be made more visually attractive, but we will take care of this later. Now it is time for the practical side, i.e., making the interface fulfill its primary task – presenting information and accepting operator commands.

Assigning functions/actions

Most interface objects are inactive at this stage. To make the widgets complete assumed tasks we have to define their input and output functions. Open the interface editor again (menu **Configuration→Open GUI editor**). Below is a table of project widgets along with the input and output properties that need to be set.
For example, for the G-Code execution start button (button with "play" icon in the left column below the 3D view) – **btnStart** set the property **Output: clicked** to **Start trajectory**).

WIDGET„ID"	PROPERTY TYPE	VALUE	DESCRIPTION
btnStart	Output: clicked	Start trajectory	Start G-Code after clicking the button
btnPause	Output: clicked	Set Pause On/Off	Enable/disable the pause of G-Code execution after clicking the button
btnStop	Output: clicked	Stop trajectory	Stop G-code execution after clicking the button
btnRewind	Output: clicked	Rewind trajectory	Rewind G-code to the beginning after clicking the button
btnCtrlEnable	Output: clicked	Switch to EStop/idle state	Switch between EStop and Standby status after clicking this button
btnCtrlEnable	Input: LED state	CSMIO enable	A state of the diode (LED) on the button will change along with the standby of a CSMIO controller
btnOpen	-	-	Widget used to open G-Code files, which does not require configuration
btnClose	Output: clicked	Run script	There is no file closing in the action list, but this can be done from a Python macro. For a list of macros for widgets, see the section below.
btnEdit	Output: clicked	Edit G-Code	Open the G-code file editor
lbFileName	Input: text	GCode file path	Displaying a path of the loaded G-Code file
lbTimeRemaining	Input: text	Remain path time	Display of remaining machining time
btnAxisXZero	Output: clicked	Run Script	Reset a program coordinate for an X-axis from a Python macro. For a list of macros for widgets, see the section below.
btnAxisYZero	Output: clicked	Run Script	As above, for a Y axis
btnAxisZZero	Output: clicked	Run Script	As above, for a Z axis
edAxisXPosition	Input: text	Axis X display position	Display of a coordinate (program/machine) for an X axis
edAxisXPosition	Output: returnPressed	Set axis X prog position	Modification of a program coordinate for an X axis after pressing the "return" key
edAxisYPosition	Input: text	Axis Y display position	Display of a coordinate (program/machine) for a Y axis
edAxisYPosition	Output: returnPressed	Set axis Y prog position	Modification of a program coordinate for a Y axis after pressing the "return" key
edAxisZPosition	Input: text	Axis Z display position	Display of a coordinate (program/machine) for a Z axis
edAxisZPosition	Output: returnPressed	Set axis Z prog position	Modification of a program coordinate for a Z axis after pressing the "return" key
ioAxisXLimitNeg	Input: state	Signal value (IP;0;mkit;0→limit--;0)	A state of an indicator light related to a state of signal „limit--" MotionKit'a no. 0 CSMIO/IP
ioAxisXHoming	Input: state	Signal value (IP;0;mkit;0→Home;0)	A state of an indicator light related to a state of signal „home" MotionKit'a no. 0 CSMIO/IP
ioAxisXLimitPos	Input: state	Signal value (IP;0;mkit;0→limit++;0)	A state of an indicator light related to a state of signal „limit++" MotionKit'a no. 0 CSMIO/IP
ioAxisYLimitNeg	Input: state	Signal value (IP;0;mkit;1→limit--;0)	A state of an indicator light related to a state of signal „limit--" MotionKit'a no. 1 CSMIO/IP
ioAxisYHoming	Input: state	Signal value (IP;0;mkit;1→Home;0)	A state of an indicator light related to a state of signal „home" MotionKit'a no. 1 CSMIO/IP
ioAxisYLimitPos	Input: state	Signal value (IP;0;mkit;1→limit++;0)	A state of an indicator light related to a state of signal „limit++" MotionKit'a no. 1 CSMIO/IP
ioAxisZLimitNeg	Input: state	Signal value (IP;0;mkit;2→limit--;0)	A state of an indicator light related to a state of signal „limit--" MotionKit'a no. 2 CSMIO/IP
ioAxisZHoming	Input: state	Signal value (IP;0;mkit;2→Home;0)	A state of an indicator light related to a state of signal „home" MotionKit'a no. 2 CSMIO/IP
ioAxisZLimitPos	Input: state	Signal value (IP;0;mkit;2→limit++;0)	A state of an indicator light related to a state of signal „limit++" MotionKit'a no. 2 CSMIO/IP

WIDGET„ID"	PROPERTY TYPE	VALUE	DESCRIPTION
cbMachineCoords	Input: checkbox state	Ref position displayed	Check box status related to whether machine coordinates are currently displayed
cbMachineCoords	Output: stateChanged	Set position display to ref	Switch to the machine coordinates display when the check box is selected
cbIgnoreSoftLimit	Input: checkbox state	Global soft limit disabled	Check box status related to software limits disabled status
cbIgnoreSoftLimit	Output: stateChanged	Disable global soft limit	Disable software limits after selecting the check box
btnRefAll	Output: clicked	Ref all axes	Start all axes homing after clicking this button
btnProbe	Output: clicked	Execute probing script	Run the default Python macro for tool measuring after clicking this button
btnPark	Output: clicked	Run script	Run the Python macro ride to the park position. For a list of macros for widgets, see the section below.
btnGoToXY	Output: clicked	Run script	Run the Python macro ride to the working offset position. For a list of macros for widgets, see the section below.
lbSelectedTool	Input: text	Selected tool number	Display the number of a currently selected tool
lbCurrentTool	Input: text	Spindle tool number	Display the tool number that is currently in a spindle
lbToolOffsetNr	Input: text	Tool offset number	Display the selected tool offset number
lbToolDiameter	Input: text	Tool diameter	Display the diameter of a current tool
btnShowToolTable	Output: clicked	Tool Table	Display the list of tools after clicking
cbKeyboardJogEnable	Input: checkbox state	Key Control	Checkbox status related to keyboard machine control enable state
cbKeyboardJogEnable	Output: state changed	Key Control	Allow a machine to be controller by a keyboard if the check box is selected
slJogSpeed	Input: value	Jog speed	Slider position related to JOG speed
slJogSpeed	Output: valueChanged	Set Jog Speed	Changing position of the slider will change current JOG speed
btnJogMode	Input: text	Jog mode	The current JOG mode will set text on the button
btnJogMode	Output: clicked	Set Jog mode	Clicking the button will change the current JOG mode
btnJogStep	Input: text	Jog step	The current jog step will set text on the button
btnJogStep	Output: clicked	Run Script	Running Python macro that will cyclically switch between values. For a list of macros for widgets, see the section below.
btnJogXNeg	Output: pressed	Jog X- pressed	Pressed button triggers JOG X-axis ride in the negative direction
btnJogXNeg	Output: released	JOG X- released	Released button stops JOG X-axis ride in the negative direction.
btnJogXPos	Output: pressed	Jog X+ pressed	Pressed button triggers JOG X-axis ride in the positive direction
btnJogXPos	Output: released	JOG X+ released	Released button stops JOG X-axis ride in the positive direction.
btnJogYNeg	Output: pressed	Jog Y- pressed	Pressed button triggers JOG Y-axis ride in the negative direction
btnJogYNeg	Output: released	JOG Y- released	Released button stops JOG Y-axis ride in the negative direction.
btnJogYPos	Output: pressed	Jog Y+ pressed	Pressed button triggers JOG Y-axis ride in the positive direction
btnJogYPos	Output: released	JOG Y+ released	Released button stops JOG X-axis ride in the positive direction.
btnJogZNeg	Output: pressed	Jog Z- pressed	Pressed button triggers JOG Z-axis ride in the negative direction
btnJogZNeg	Output: released	JOG Z- released	Released button stops JOG Z-axis ride in the negative direction.
btnJogZPos	Output: pressed	Jog Z+ pressed	Pressed button triggers JOG Z-axis ride in the positive direction
btnJogZPos	Output: released	JOG Z+ released	Released button stops JOG Z-axis ride in the positive direction.
slFro	Input: value	Fro	The slider position will be set as the FRO value changes
slFro	Output: valueChanged	Set Fro	Changing the slider position will change the current FRO value
btnFro5	Output: clicked	Run Script	Running a Python macro that will set the FRO value to 5%. For a list of macros for widgets, see the section below.

WIDGET„ID"	PROPERTY TYPE	VALUE	DESCRIPTION
btnFro50	Output: clicked	Run Script	Running a Python macro that will set the FRO value to 50%. For a list of macros for widgets, see the section below.
btnFro100	Output: clicked	Run Script	Run a Python macro that will set the FRO value to 100%. For a list of macros for widgets, see the section below.
btnFro200	Output: clicked	Run Script	Run a Python macro that will set the FRO value to 200%. For a list of macros for widgets, see the section below.
edFeedrate	Input: text	Feedrate	Displaying a currently set feed rate.
edFeedrate	Output: returnPressed	Set Feedrate	Setting feed rate when pressing the return key.
edFeedrateOv	Input: text	Feedrate override	Display of the currently set feed rate, including FRO
edFeedrateCurrent	Input: text	Current velocity	Displaying the current resultant velocity - speed of a tool relative to a material.
slSro	Input: text	Sro	The slider position will be set as the SRO value changes
slSro	Output: valueChanged	Set Sro	Changing the slider position will change the current SRO value
btnSroReset	Output: clicked	Run Script	Running a Python macro that will set the SRO value to 100%. For a list of macros for widgets, see the section below.
edSpindleSpeed	Input: text	Spindle Speed	Displaying currently set spindle speed
edSpindleSpeed	Output: returnPressed	Set Spindle Speed	Setting spindle speed when pressing the return key
btnSpindleCW	Output: clicked	Run Spindle Clockwise	Enable right spindle revs after clicking the button
btnSpindleCW	Input: value	Spindle CW percent	The bar on the button will show progress in achieving set revs (right revs)
btnSpindleCCW	Output: clicked	Run Spindle Counter-Clockwise	Enable left spindle revs after clicking the button
btnSpindleCCW	Input: value	Spindle CCW percent	The bar on the button will show progress in achieving set revs (left revs)
btnFlood	Output: clicked	Set Flood On/Off	Enable/disable coolant
btnFlood	Input: LED state	Flood on	The LED on the button will be lit when the coolant is on

Python macros for widgets with „Run script" action

As you can see in the table above, some widgets do not have a specific function assigned, only **Run script**, which activates a Python macro. Below are the names of files with their source codes. You can use the built-in simCNC editor to create files (menu **Macros→Show script editor**), or any other editor, e.g., **VS Code.** It is best to save the files in the projected screen directory, in the scripts subdirectory. The files must have a ".py" extension.

For Windows, this will be the path: *„C:\Program Files\simCNC\screens\ui_example\scripts"*
For Linux: *"/opt/simCNC/screens/ui_example/scripts"*
For macOS: *"/Applications/CS-Lab/simCNC.app/Contents/MacOS/screens/ui_example/scripts/"*

*Macro for **btnClose** button - file„btnClose.py"*

```
d.closeGCodeFile( )
print(„GCode file closed.")
```

*Macro for **btnAxisXZero** button - file„btnAxisXZero.py"*

```
d.setAxisProgPosition( Axis.X, 0 )
print("Axis X prog position set to 0.000")
```

*Macro for **btnAxisYZero** button - file„btnAxisYZero.py"*

```
d.setAxisProgPosition( Axis.Y, 0 )
print("Axis Y prog position set to 0.000")
```

*Macro for **btnAxisZZero** button - file„btnAxisZZero.py"*

```
d.setAxisProgPosition( Axis.Z, 0 )
print("Axis Z prog position set to 0.000")
```

*Macro for **btnPark** button - file„btnPark.py"*

```
d.executeGCode( "G0G53 Z0" );
d.executeGCode( "G0G53 X0 Y0" );
print("Go to park position finished")
```

*Macro for **btnGoToXY** button - file„btnGoToXY.py"*

```
d.executeGCode( "G0G53 Z0" );
d.executeGCode( "G0 X0 Y0" );
print("Go to material zero XY position finished")
```

*Macro for **btnJogStep** button - file„btnJogStep.py"*

```
currentStep = d.getJogStep( )
newStep = currentStep * 10.0
if newStep > 1.0:
        newStep = 0.001
d.setJogStep( newStep )
print("JOG step set to: {:.3f}".format(newStep))
```

*Macro for **btnFro5** button - file„btnFro5.py"*

```
d.setFRO( 5 )
print("FRO set to: {:.1f}%".format(d.getFRO( )))
```

*Macro for **btnFro50** button - file„btnFro50.py"*

```
d.setFRO( 50 )
print("FRO set to: {:.1f}%".format(d.getFRO( )))
```

*Macro for **btnFro100** button - file„btnFro100.py"*

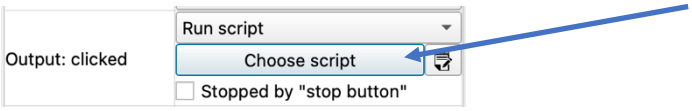
```
d.setFRO( 100 )
print("FRO set to: {:.1f}%".format(d.getFRO( )))
```

*Macro for **btnFro200** button - file„btnFro200.py"*

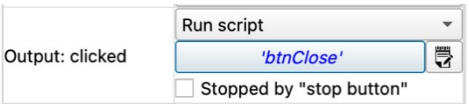
```
d.setFRO( 200 )
print("FRO set to: {:.1f}%".format(d.getFRO( )))
```

Assigning macros to widgets

To assign a macro to a widget, select the widget, and in the list of properties, click the macro selection button under the Run script action.



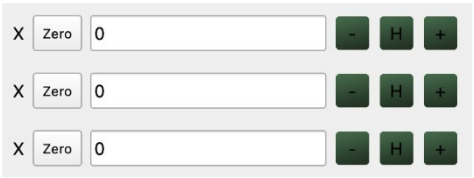
This is how it should look like for the **btnClose**button, after selecting the "btnClose.py" macro



In this way, we assign all of the above macros to the widgets.

Additions and minor corrections

At this stage, we already have an interface that works and can be used. Before we start working on visual improvements and cosmetics, we will make a few small corrections.



In edit mode, we change the **text** property of the **lbAxisYName** widget to "Y" and the **lbAxisZName** widget to "Z".

2. Setting the value range for **the slJogSpeed**, **slFro,** and **slSro** sliders
    a. Set **maximum** propriety of **slJogSpeed** widget to 100
    b. Set **maximum** propriety of **slFro** widget to 200
    c. Set **maximum** propriety of **slSro** widget to 200
3. Setting a value range for spindle revs visualization bars on **the btnSpindleCW** and **btnSpindleCCW** buttons
    a. Set **maximum** propriety of **btnSpindleCW** widget to 1
    b. Set **maximum** propriety of **btnSpindleCCW** widget to 1
4. Setting the coordinate display format to X.XXX
    a. In the **display format** property field of the **edAxisXPosition**widget, **edAxisYPosition,** and **edAxisZPosition,** type "%.3f" (without quotation marks)

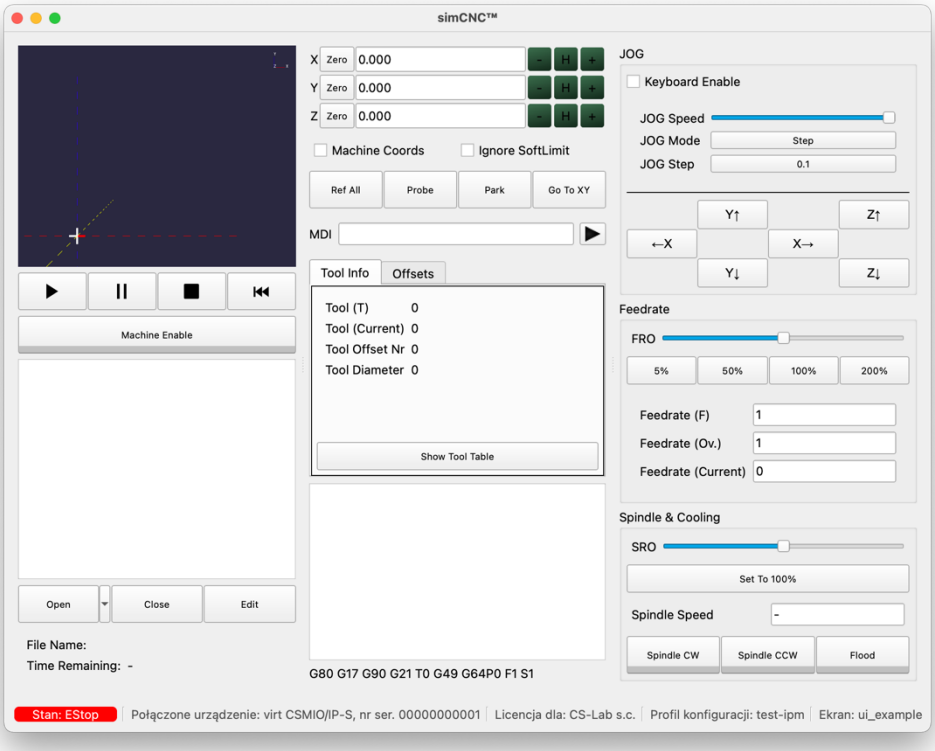
Stylization

As mentioned earlier, this step is optional. Our interface should function properly at this stage. However, it is worth spending a little more work to perfect its appearance.

At the beginning, we will "fine-tune" the scaling and the rules for dividing space of some widgets and containers. To do this, we set the following properties to objects:

OBJECT NAME („ID")	PROPERTY	VALUE
loutLeftColumn	stretch	1,0,0,1,0,0
loutExecutionCtrlButtons	minimum height	40
loutExecutionCtrlButtons	left, right, top, bottom margin	0
loutExecutionCtrlButtons	spacing	1
btnCtrlEnable	minimum height	40
loutFileCtrlButtons	minimum height	40
loutFileCtrlButtons	left, right, top, bottom margin	0
loutFileCtrlButtons	spacing	1
loutFileInfo	left, right, top, bottom margin	0
loutAxesDROs	left, right, top, bottom margin	0
loutAxesDROs	spacing	1
loutAxisXDRO	left, right, top, bottom margin	1
loutAxisXDRO	spacing	1
loutAxisYDRO	left, right, top, bottom margin	1
loutAxisYDRO	spacing	1
loutAxisZDRO	left, right, top, bottom margin	1
loutAxisZDRO	spacing	1
loutMiscButtons	minimum height	40
loutMiscButtons	left, right, top, bottom margin	0
loutMiscButtons	spacing	1
loutToolInfoLabels	left, right, top, bottom margin	0
btnShowToolTable	minimum height	30
loutRightColumn	stretch	1,0,0
loutJogConfig	vertical size policy	Maximum
frJogLine	vertical size policy	Maximum
loutFroButtons	minimum height	30
loutFroButtons	left, right, top, bottom margin	0
loutFroButtons	spacing	1
btnFro5	vertical size policy	Preferred
btnFro50	vertical size policy	Preferred
btnFro100	vertical size policy	Preferred
btnFro200	vertical size policy	Preferred
btnSroReset	minimum height	30
loutSpindleAndCoolingCtrl	minimum height	40
loutSpindleAndCoolingCtrl	left, right, top, bottom margin	0
loutSpindleAndCoolingCtrl	spacing	1

The interface should now look a bit neater and compact:



Let's also set a red color for the lights indicating states of limit switches:

OBJECT NAME („ID”)	PROPERTY	VALUE
ioAxisXLimitNeg	color	(red)
ioAxisXLimitPos	color	(red)
ioAxisYLimitNeg	color	(red)
ioAxisYLimitPos	color	(red)
ioAxisZLimitNeg	color	(red)
ioAxisZLimitPos	color	(red)

(...) and right alignment and slightly larger font for axis name widgets and coordinate display widgets:

OBJECT NAME („ID”)	PROPERTY	VALUE
edAxisXPosition	horizontal alignment	Right
edAxisXPosition	font	Arial 20
lbAxisXName	font	Arial 20
edAxisYPosition	horizontal alignment	Right
edAxisYPosition	font	Arial 20
lbAxisYName	font	Arial 20
edAxisZPosition	horizontal alignment	Right
edAxisZPosition	font	Arial 20
lbAxisZName	font	Arial 20

(...) let's disable the additional frame in the widget with the "Tool Info" and "Offsets" tabs:

Nazwa („id”) obiektu	PROPERTY	VALUE
tabToolInfo	shape	No frame
tabOffsets	shape	No frame

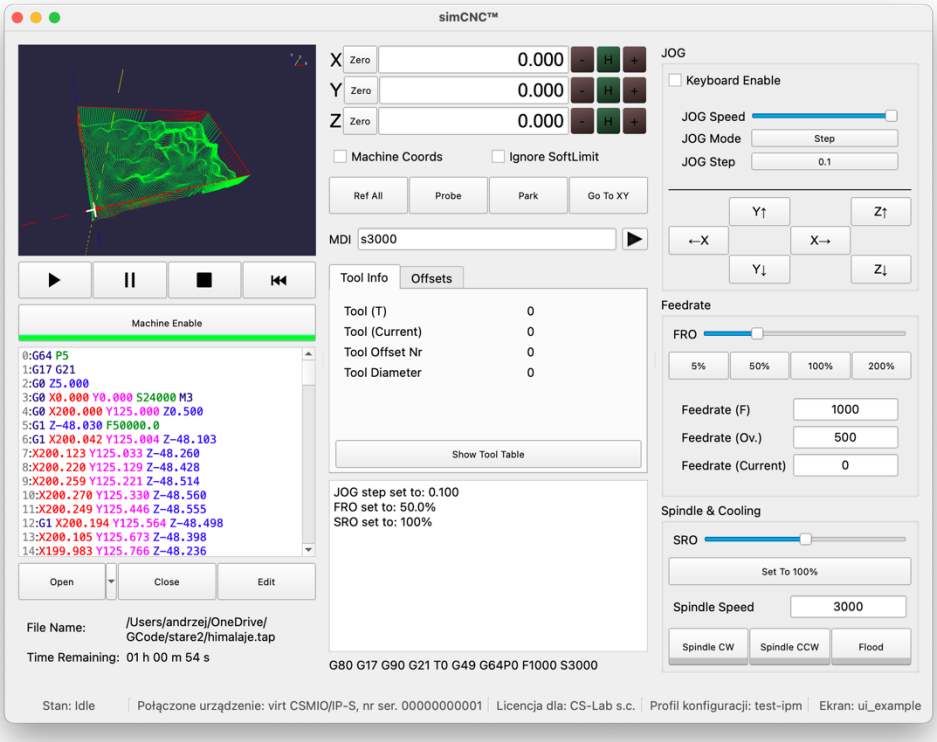
(...) let's center the display of some parameters

OBJECT NAME („ID")	PROPERTY	VALUE
lbSelectedTool	Horizontal alignment	Centered
lbCurrentTool	Horizontal alignment	Centered
lbToolOffsetNr	Horizontal alignment	Centered
lbToolDiameter	Horizontal alignment	Centered
edFeedrate	Horizontal alignment	Centered
edFeedrateOv	Horizontal alignment	Centered
edFeedrateCurrent	Horizontal alignment	Centered
edSpindleSpeed	Horizontal alignment	Centered

and set the properties of the lbFileName widget to fit longer names:

OBJECT NAME („ID")	PROPERTY	VALUE
lbFileName	Word wrap	(selected)
lbFileName	Vertical size policy	Maximum
lbFileName	Minimum height	32

The interface now looks like this:



Final cosmetic using css style sheets

Styling with style sheets is a powerful tool, but rather intended for more advanced users. A comprehensive overview of all features is beyond the scope of this manual. A simple example is provided later in this chapter, and for those interested, the links below will provide more detailed information on this subject.

https://www.w3schools.com/css/css_intro.asp
https://doc.qt.io/qt-5/stylesheet-syntax.html

I also encourage you to read and experiment with the css files in "default" screen designs. Please remember to make a copy and work on it, because any changes to the default screens may be overwritten after updating the simCNC software.

Properties passed by simCNC

SimCNC passes several properties to css sheets that allow you to change the appearance of elements depending on, for example, a state of the program or the dark theme mode.

PROPERTY	VALUE	DESCRIPTION
darkTheme	true false	Returns **true** if a dark theme is active
state	estop idle homing trajectory jog mdi mpg	Returns a current state of a machine
csmioState	init disabled prepare run fault	Returns a current status of a CSMIO/IP device
pause	true false	Returns **true** if machining is paused (pause)
Axis<X,Y,Z, ...>Referenced	true false	Returns **true** if an axis is correctly referenced
Axis<X,Y,Z, ...>Enabled	true false	Returns **true** if a given axis is enabled in the configuration

Assumptions of the example

In the following example, we will perform the following stylizations:

- We will change a background color and Y-axis line in the 3D preview
- We will change colors of X, Y, Z axes names and make the colors will depend on reference status of a given axis.
- We will darken the background under some labels.
- We will make the JOG buttons to backlight when we hover a mouse over them.
- We will make the FRO and SRO reset buttons to backlight when we hover a mouse over them.

Grouping

It often happens that we want to set similar style properties for many widgets. You can simplify your work by setting the **group** property in the interface editor. Thanks to this, in the **css** sheet we can refer to a group, not to individual elements. Now we set **group** property for the widgets according to the table below:

WIDGET NAME („ID”)	GROUP NAME (GROUP PROPERTY)
btnJogXPos	JogButtons
btnJogXNeg	JogButtons
btnJogYPos	JogButtons
btnJogYNeg	JogButtons
btnJogZPos	JogButtons
btnJogZNeg	JogButtons
btnFro100	Set100PercentButtons
btnSroReset	Set100PercentButtons
lbAxisXName	AxesNameLabels
lbAxisYName	AxesNameLabels
lbAxisZName	AxesNameLabels
lbSelectedTool	DarkerLabels
lbCurrentTool	DarkerLabels
lbToolOffsetNr	DarkerLabels
lbToolDiameter	DarkerLabels
lbFileName	DarkerLabels
lbTimeRemaining	DarkerLabels

Colors changing in the 3D preview

From the directory of our interface, we open the file "colors.css". You can use any text editor, such as **Notepad** or **Visual Studio Code**. The advantage of the latter is that it colors and partially analyzes the correctness of syntax.

In the file, we edit the background color (**path_view_background**) and the color of the Y axis (**axisY**) to make it more visible on the new background:

```
[type="path_view_background"] {
  color: #202020;
}
[type="axis_Y"] {
  color: #4646ff;
}
```

Creating a new style sheet file

In the directory of our interface, we create a new text file called "widgets.css" – so in our example, in Windows OS, the path will be like this:

C:\Program Files\simCNC\screens\ui_example\widgets.css

simCNC automatically searches the interface directory for **.css** files when the screen loads.

Changing colors of axis labels and visualization of reference state

In the "widgets.css" file, we add the following instructions:

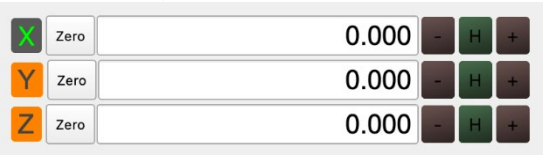
```
[group="AxesNameLabels"] {
  background-color: #e80;
  border-radius: 3px;
  margin: 2px;
  color: #444;
}

[id="lbAxisXName"][axisXReferenced="true"] {
  background-color: #a0000000;
  color: #0f0;
}
[id="lbAxisYName"][axisYReferenced="true"] {
  background-color: #a0000000;
  color: #0c0;
}
[id="lbAxisZName"][axisZReferenced="true"] {
  background-color: #a0000000;
```

As you can see, we refer to widgets here in two ways. First, we set the default appearance for the entire group (**AxesNameLabels**), while below for each axis, conditional styling is performed separately if the axis homing flag is set to **true**.

The modified properties are:

- background-color
- border-radius – rounded corners
- margin
- color – a label text color

Below we see the effect when the X-axis is referenced and the Y and Z axes are not:
(To make changes to css visible, reload the screen: menu Configuration→Reload screen)



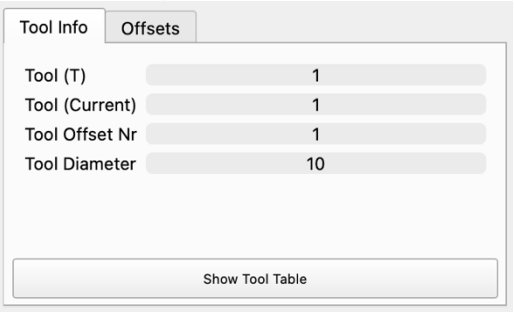
*Darker background under **DarkerLabels** group labels*

```
[group="DarkerLabels"] {
  background-color: #10000000;
  border-radius: 5px;
}
```

In this case, we set only two parameters for the group named **DarkerLabels**:

- background-color
- border-radius

Below you can see the effect – a darker background and rounded corners under the labels:
(To make changes to css visible, reload the screen: menu Configuration→Reload screen)

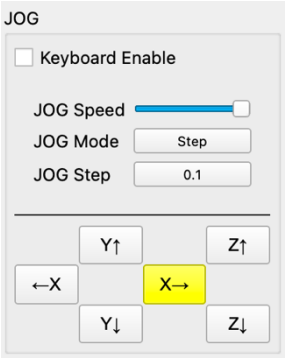


Changing the color (backlighting) of the JOG buttons

```
[group="JogButtons"]:hover {
  background-color: yellow;
}
```

As you can see above, we only modify the **background color**, however we do it conditionally. The **hover** keyword is responsible for that the style will be applied only when a mouse hover over the widget. This creates the effect of "backlighting" the button.

Below you can see the effect:
*(To make changes to css visible, reload the screen: menu **Configuration→Reload screen**)*

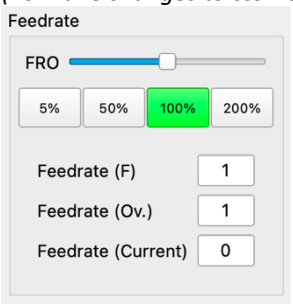


Changing the color (backlighting) of the FRO and SRO reset buttons

```
[group="Set100PercentButtons"]:hover {
  background-color: #0f0;
}
```

The same as for JOG buttons – we conditionally modify the **background color (background-color**). The **hover** keyword is responsible for that the style will be applied only when a mouse hover over the widget. This creates the effect of "backlighting" the button.

Below you can see the effect:
*(To make changes to css visible, reload the screen: menu **Configuration→Reload screen**)*

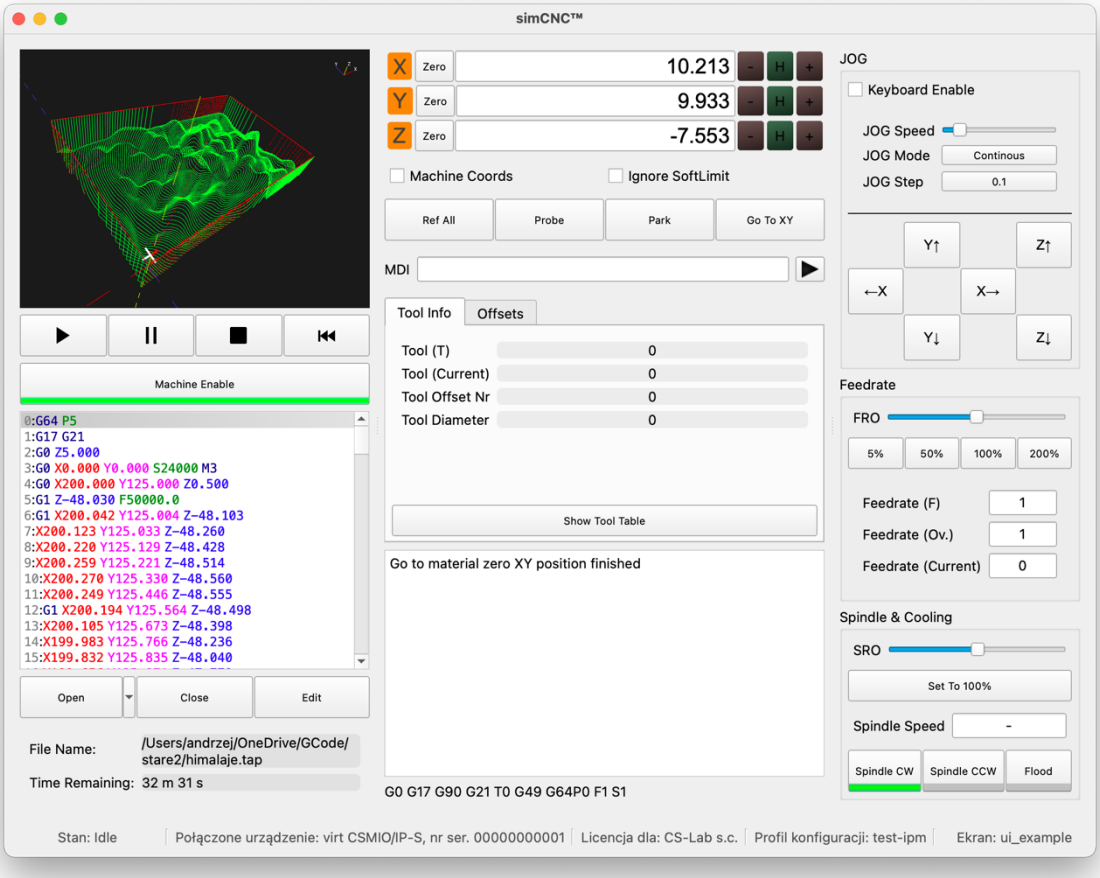


All contents of the "widgets.css" file

```
[group="AxesNameLabels"] {
  background-color: #e80;
  border-radius: 3px;
  margin: 2px;
  color: #444;
}

[id="lbAxisXName"][axisXReferenced="true"] {
  background-color: #a0000000;
  color: #0f0;
}
[id="lbAxisYName"][axisYReferenced="true"] {
  background-color: #a0000000;
  color: #0c0;
}
[id="lbAxisZName"][axisZReferenced="true"] {
  background-color: #a0000000;
  color: #0c0;
}

[group="DarkerLabels"] {
  background-color: #10000000;
  border-radius: 5px;
}

[group="JogButtons"]:hover {
  background-color: yellow;
}
```

Above is the entire "widgets.css" file. As you can see, a dozen or so lines are enough to make the interface design more visually attractive, and often more convenient to use and more readable for an operator.

The final result and summary



This is how the finished project looks like. As you could see by reading this chapter, making the interface "from scratch" requires a bit of work, but with this feature, simCNC software can be adapted relatively quickly to convenient operation of many types of machines and their accessories.

The project described in this chapter is included in the standard simCNC installation (3.410 version and later).

If you have made your own project that you would like to share with other users, please let us know at office@cslab.eu.

The CS-Lab Team gives you best regards, wishes you fruitful work and great results ☺