VisUI

Read me

By SEAUVE Melvin

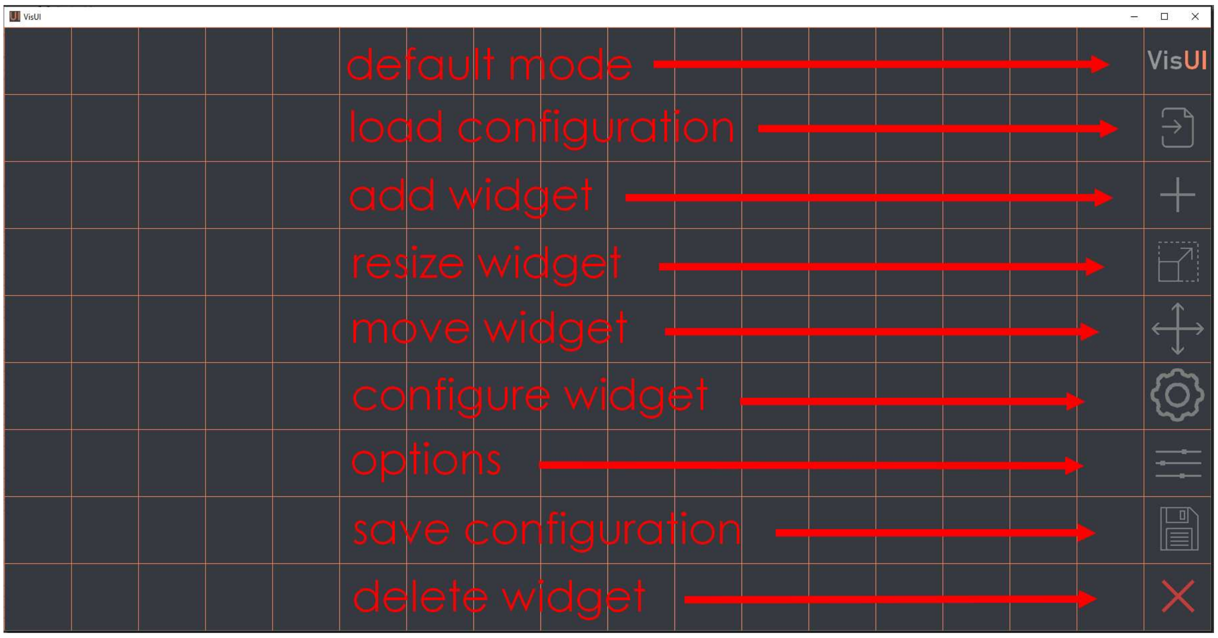
A software to visualize your data

SUMMARY :

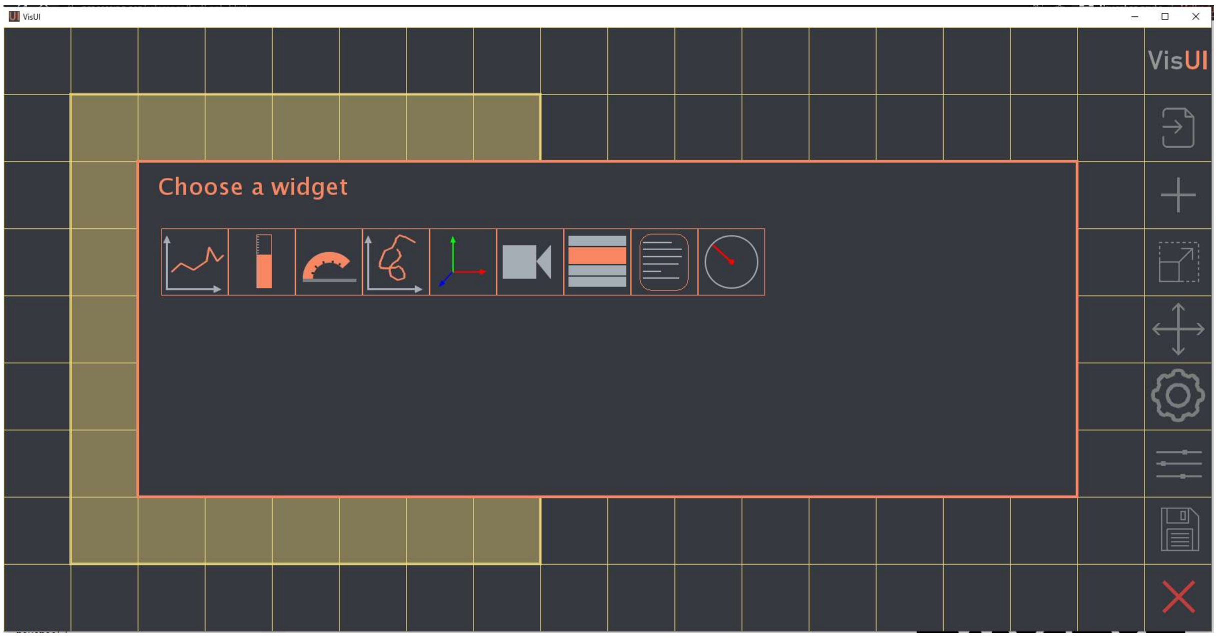
Introduction

VisUI in an application with the objective to show data from a serial communication. There are a lot of widget to allow you to visualize your data as you want, you can also create your own widget. This application was originally made for a student rocket project (Belisama), launch at the 2023 CSpace edition (At Tarbes in France).

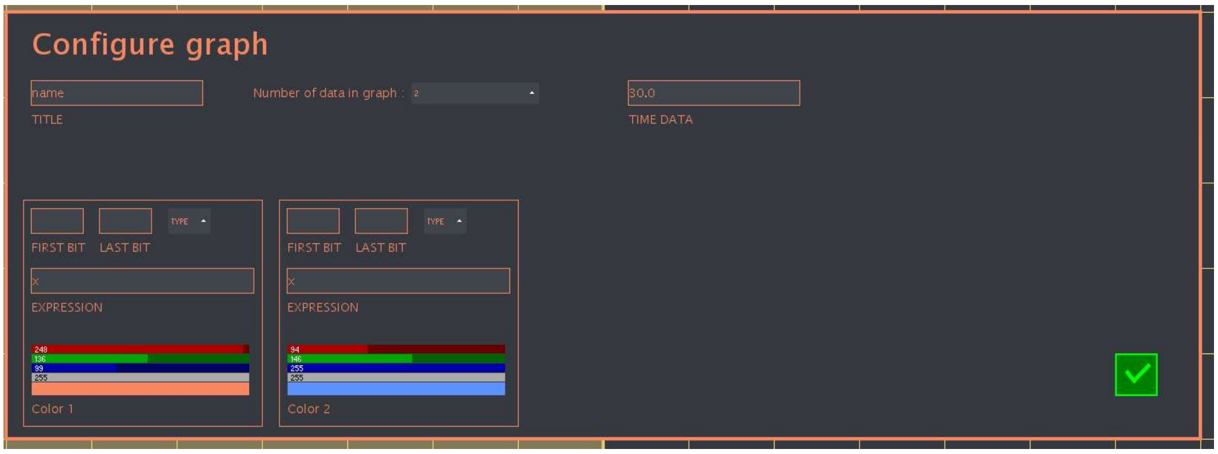
How to use



When you launch the application, you will be on this view. You can see some buttons on the right side of the app. Firstly you can choose to add a widget, select a large area, then choose a widget in the list.



If you choose the first widget in the list, the graph, you will be on his configuration menu. You can show data up to 4 different colors. The time data is the maximum display time of your data. For each data you to select the type (int or float) that the app will received on his COM port, the first and last bit are to define the position of the information in the message. You can also modify the received value with a mathematical expression (to optimize the length of your sending data).

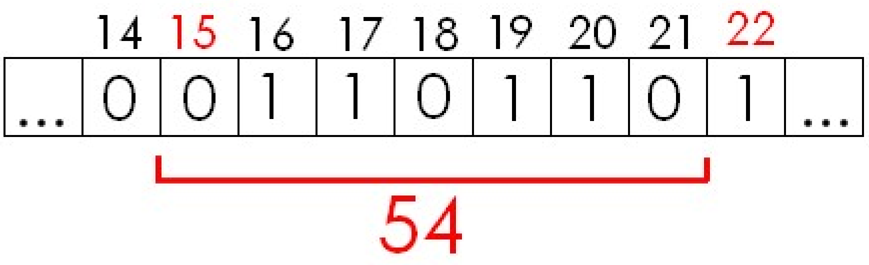


Configure graph

TITLE: name	Number of data in graph : 2	TIME DATA: 30.0

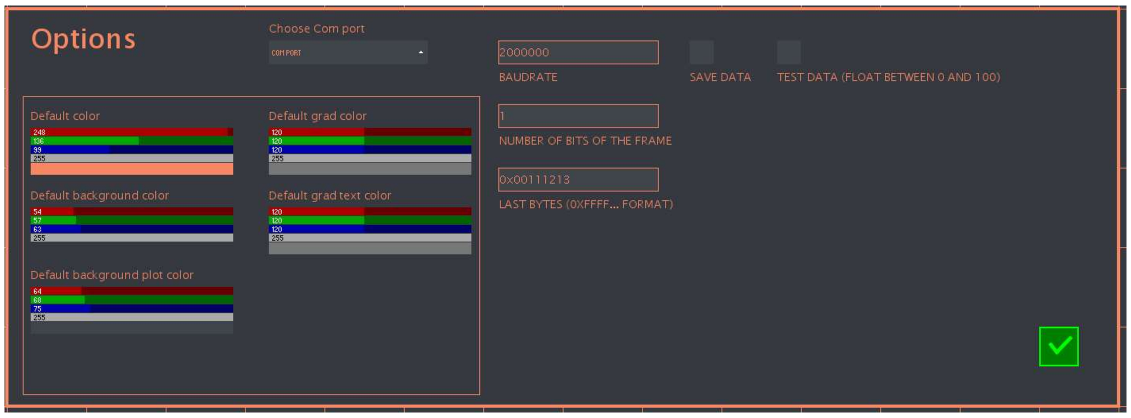
FIRST BIT	FIRST BIT
LAST BIT	LAST BIT
TYPE	TYPE
EXPRESSION: x	EXPRESSION: x
Color 1: 248, 136, 99, 255	Color 2: 94, 146, 255, 255

What first/last bit mean? This app is config to received binary values, you have to follow this kind of message.



The red index of the message (15 and 22) are the first and last bit to give to get the value of 54.

The options mode allow you to configure the color theme of the app, but also to choose the COM port or the baudrate. The save data button will save every data you received in text file. The test data button will generate random value (between 0 and 100) to allow you to verify the configuration of the app without data.



Options

Choose Com port: COM PORT

BAUDRATE	2000000
NUMBER OF BITS OF THE FRAME	1
LAST BYTES (0XFFFF... FORMAT)	0x00111213

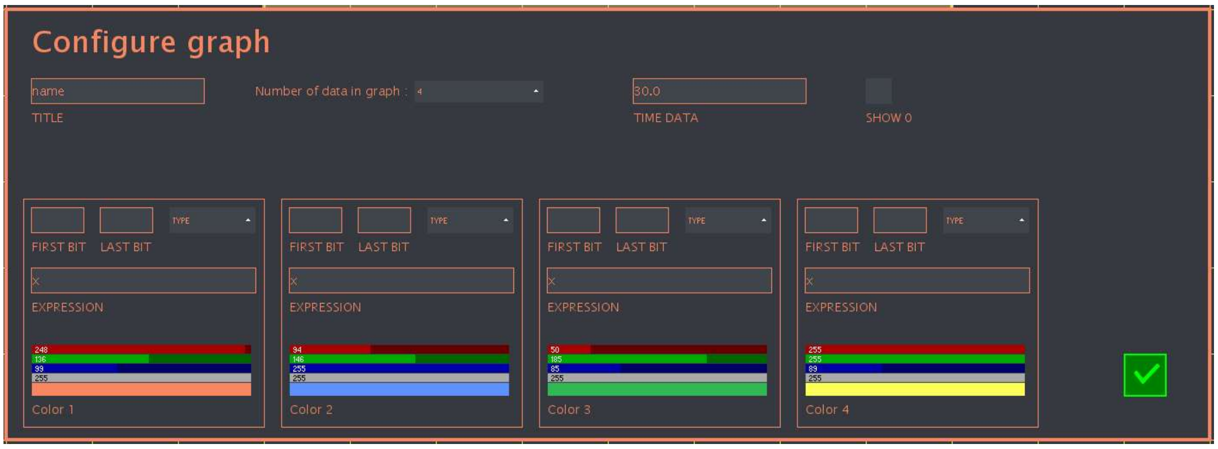
SAVE DATA

TEST DATA (FLOAT BETWEEN 0 AND 100)

Default color: 248, 136, 99, 255	Default grad color: 120, 120, 120, 255
Default background color: 54, 57, 63, 255	Default grad text color: 120, 120, 120, 255
Default background plot color: 64, 68, 75, 255	

Graph widget

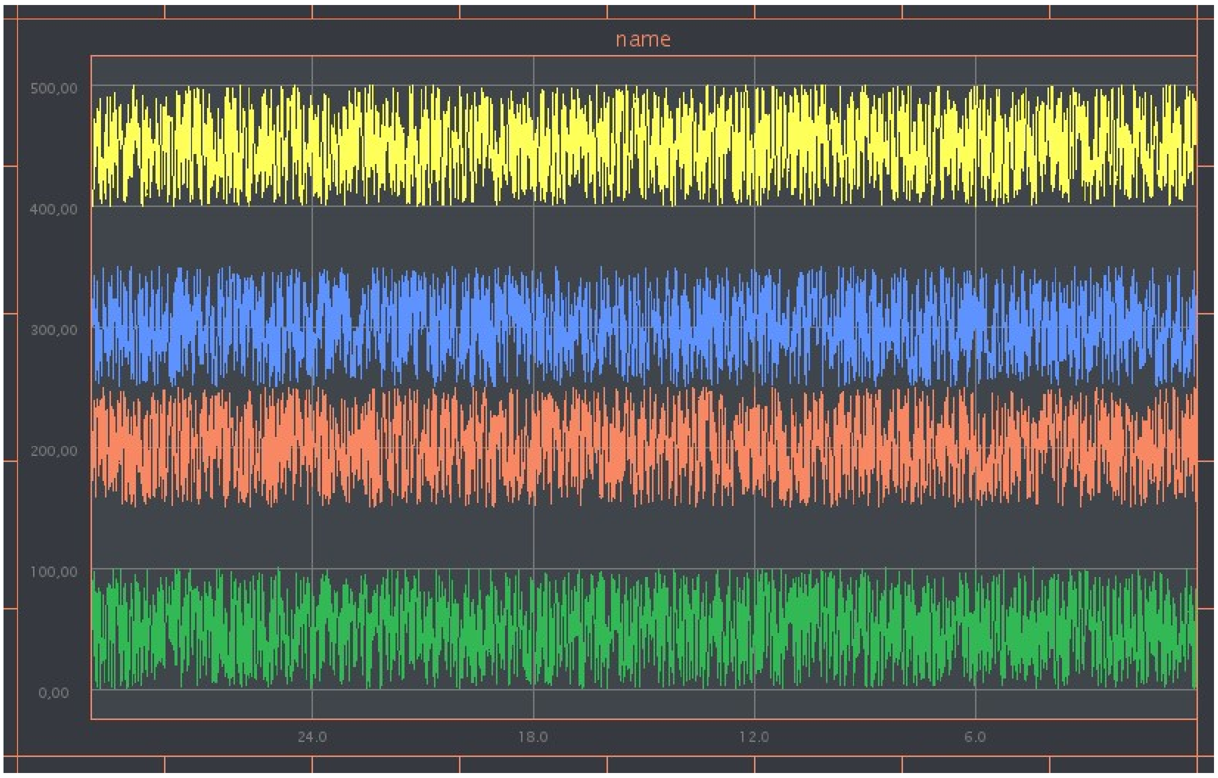
The graph widget allow you to show 1 to 4 different data for a given time.


Configure graph

TITLE: name
Number of data in graph : 4
TIME DATA: 30.0
SHOW 0

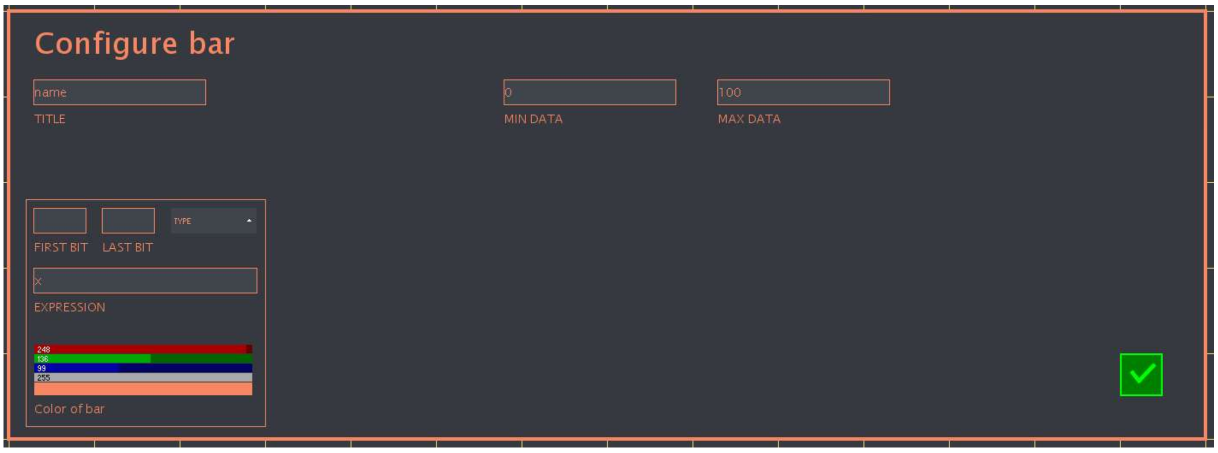
FIRST BIT	FIRST BIT	FIRST BIT	FIRST BIT
LAST BIT	LAST BIT	LAST BIT	LAST BIT
TYPE	TYPE	TYPE	TYPE
EXPRESSION: x	EXPRESSION: x	EXPRESSION: x	EXPRESSION: x
248	94	50	255
136	146	185	255
99	255	85	89
255	255	255	255
Color 1	Color 2	Color 3	Color 4

For each data you can choose a color, the first and last bit, the data type and the expression. The expression is the mathematical expression to apply to the data, exp, cos, sin and tan are allowed. You can choose if you want to always show 0.



Bar widget

The bar widget show data between a range defined by the user.


Configure bar

name
TITLE

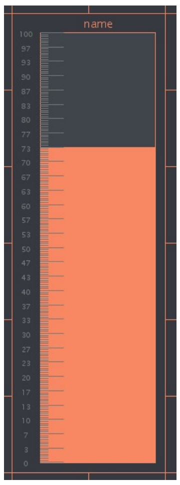
0
MIN DATA

100
MAX DATA

FIRST BIT
LAST BIT
TYPE

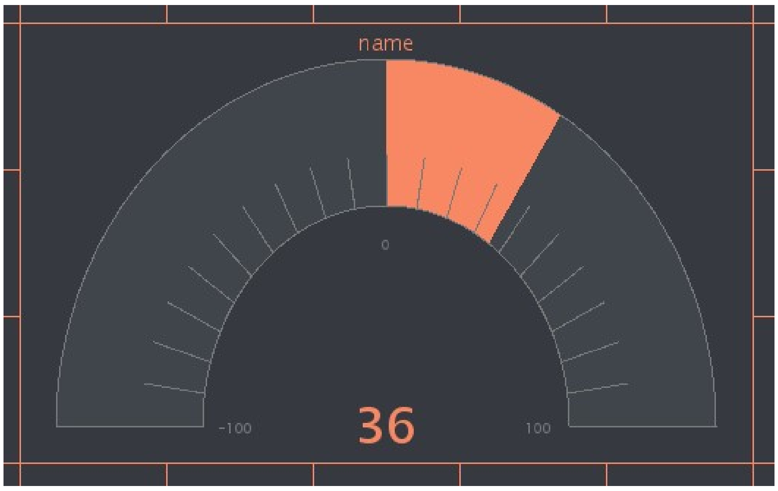
x
EXPRESSION

248
136
99
255
Color of bar



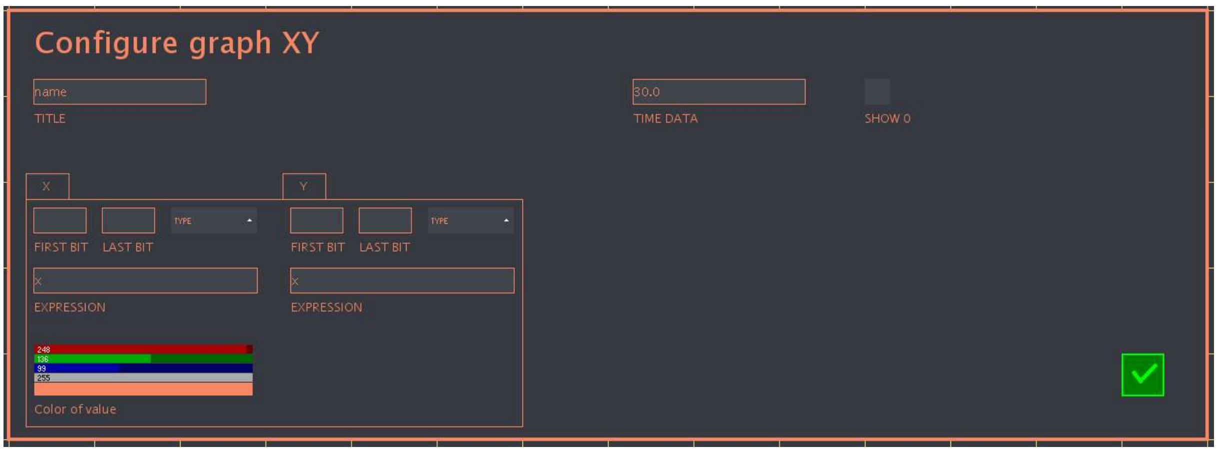
Gauge widget

The gauge widget has the same parameters, it shows data between a range.



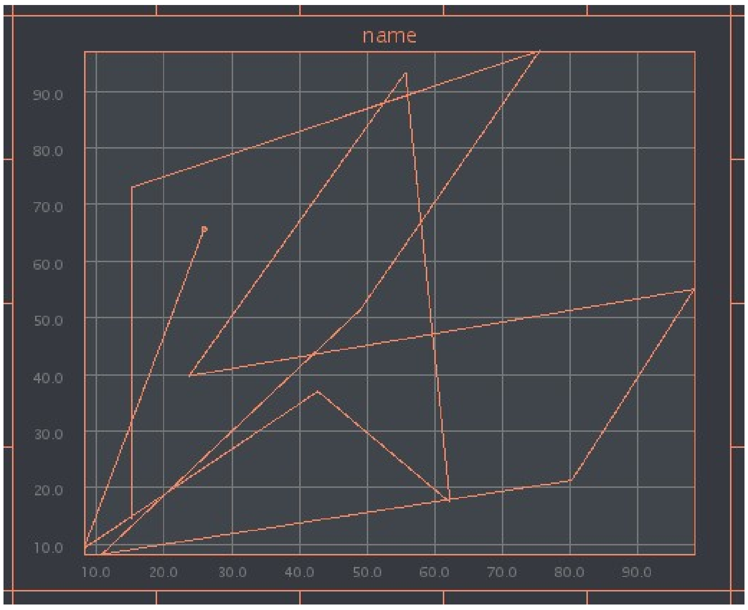
Graph XY widget

The graph XY widget show one data from 2 position (x and y) and its evolution during a given time.



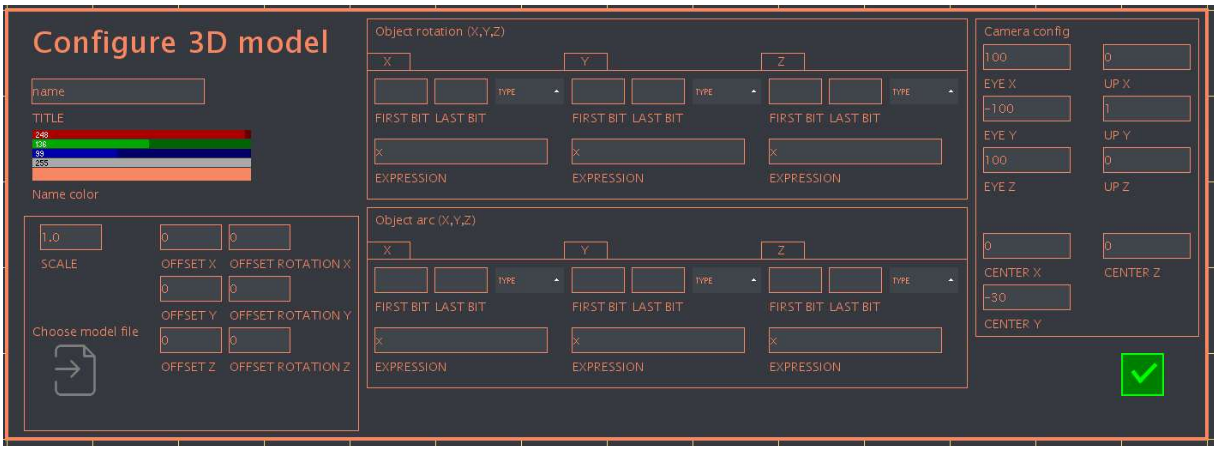
Configure graph XY

TITLE	name
TIME DATA	30.0
SHOW 0	



3D model widget

This widget show a 3d model selected by the user and some additional data.


Configure 3D model

name
TITLE
248
136
99
255
Name color

1.0	0	0
SCALE	OFFSET X	OFFSET ROTATION X
	0	0
	OFFSET Y	OFFSET ROTATION Y
Choose model file	0	0
	OFFSET Z	OFFSET ROTATION Z

Object rotation (X,Y,Z)

X	Y	Z
FIRST BIT LAST BIT TYPE	FIRST BIT LAST BIT TYPE	FIRST BIT LAST BIT TYPE
x	x	x
EXPRESSION	EXPRESSION	EXPRESSION

Object arc (X,Y,Z)

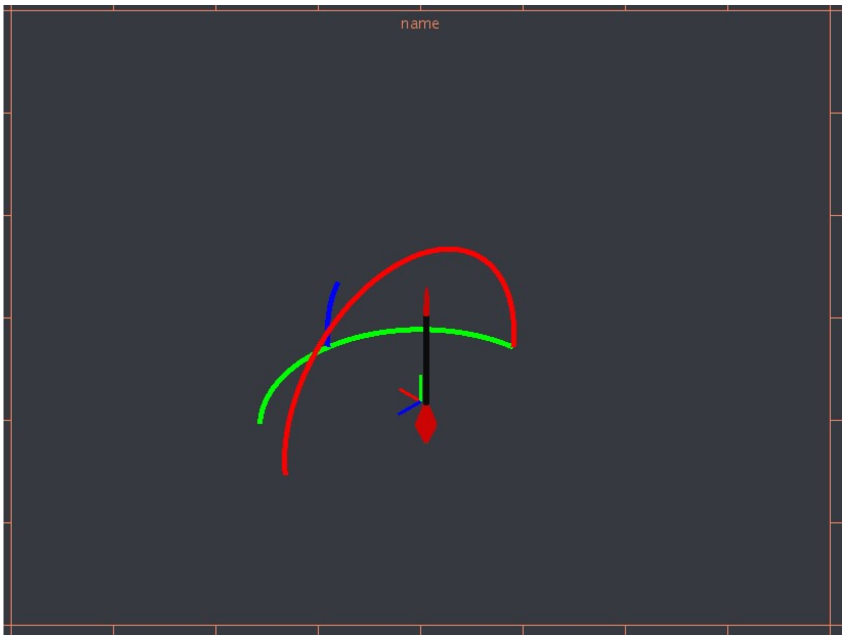
X	Y	Z
FIRST BIT LAST BIT TYPE	FIRST BIT LAST BIT TYPE	FIRST BIT LAST BIT TYPE
x	x	x
EXPRESSION	EXPRESSION	EXPRESSION

Camera config

100	0
EYE X	UP X
-100	1
EYE Y	UP Y
100	0
EYE Z	UP Z
0	0
CENTER X	CENTER Z
-30	
CENTER Y	

It have a huge configuration panel, in the first config group beneath the color selector, there is the configuration of the model. By click on the image the user can select the model he want to show (only .obj with .mtl with the same name), the scale, the position and the rotation offset. To define the rotation offset more easily, allow, in the options mode, the test data.

In the object rotation and object arc group you can define the 3 component of the rotation of the model, and the arc created around the model. If you don't want show these data, write "x-x" in the expression field. The rotation you be given in rad, if you get your angle in degree, use the expression field.



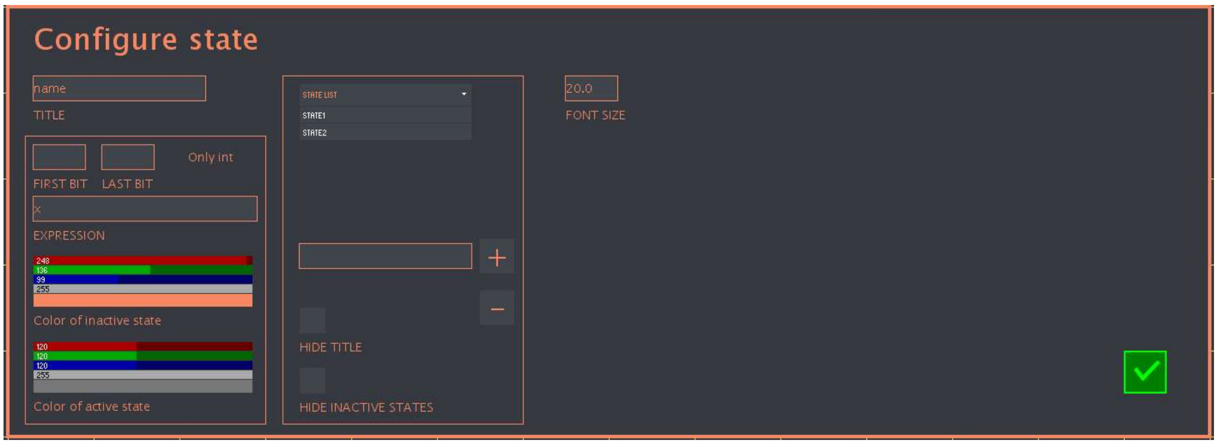
The camera config group define the camera position and view direction by 9 variables, to see more information about these variables, look at Processing documentation and Opengl documentation.

Camera widget

This widget show a camera from the available camera, click on the widget to go to the next camera.

State widget

The state widget show the current state from a list, you can show inactive state.



Configure state

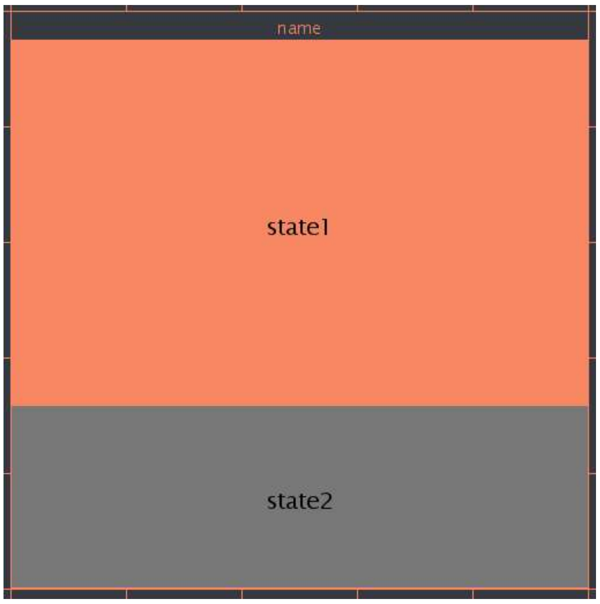
name
TITLE

Only int
FIRST BIT LAST BIT
x
EXPRESSION
248
136
99
255
Color of inactive state
120
120
120
255
Color of active state

STATE LIST
STATE1
STATE2

HIDE TITLE
HIDE INACTIVE STATES

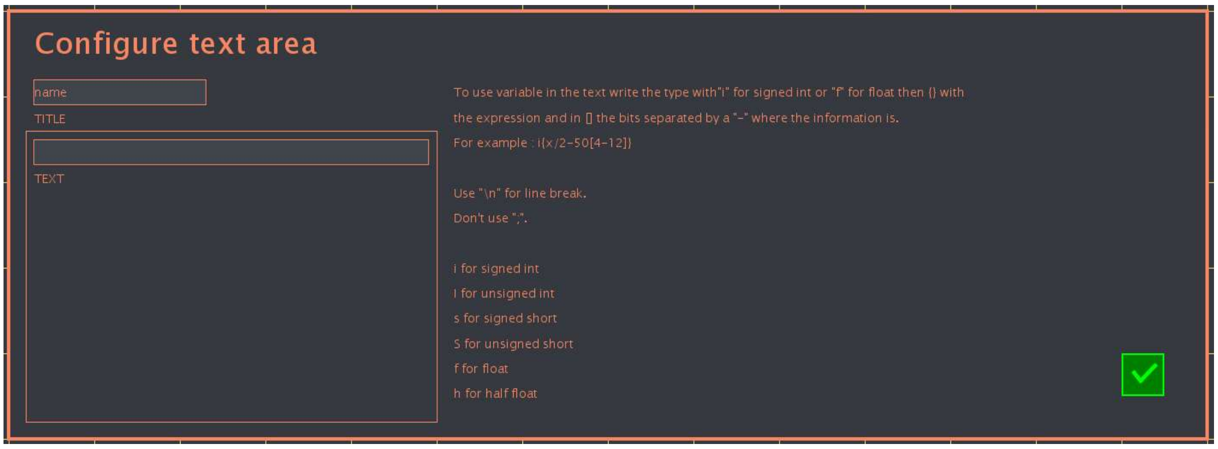
20.0
FONT SIZE



You can add as much state as you want.

Text area widget

This widget show a text defined by the user.



Configure text area

name

TITLE

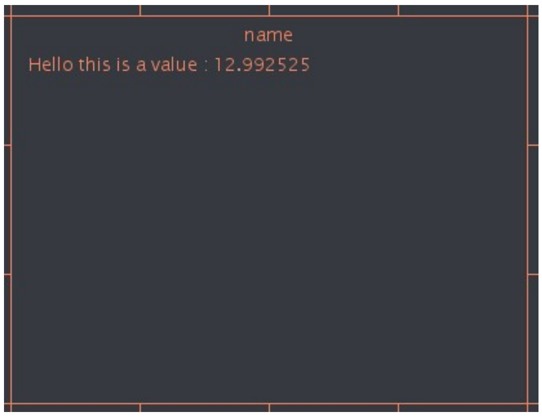
TEXT

To use variable in the text write the type with"i" for signed int or "f" for float then {} with the expression and in [] the bits separated by a "-" where the information is.
For example : i{x/2-50[4-12]}

Use "\n" for line break.
Don't use ";".

i for signed int
I for unsigned int
s for signed short
S for unsigned short
f for float
h for half float

You can show value from serial communication, for that use a letter for the data type followed by "{" then the expression. After the expression get the first and last bit between "[ ]" like "[4-8]" and close with "}".

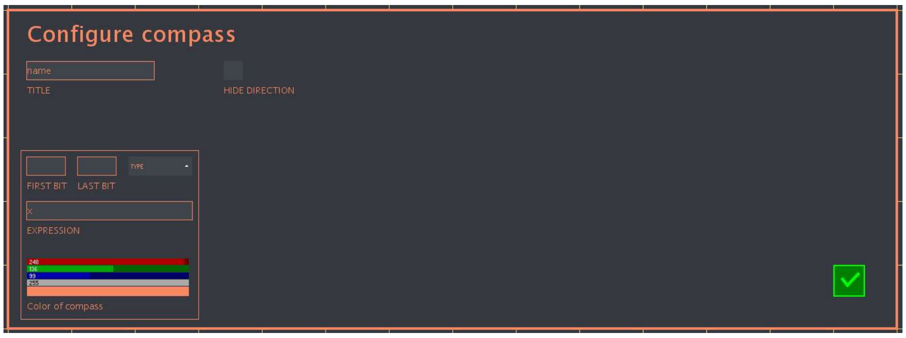


name

Hello this is a value : 12.992525

Compass widget

This widget show a direction with a value in rad.



Configure compass

name

TITLE

HIDE DIRECTION

FIRST BIT LAST BIT TYPE

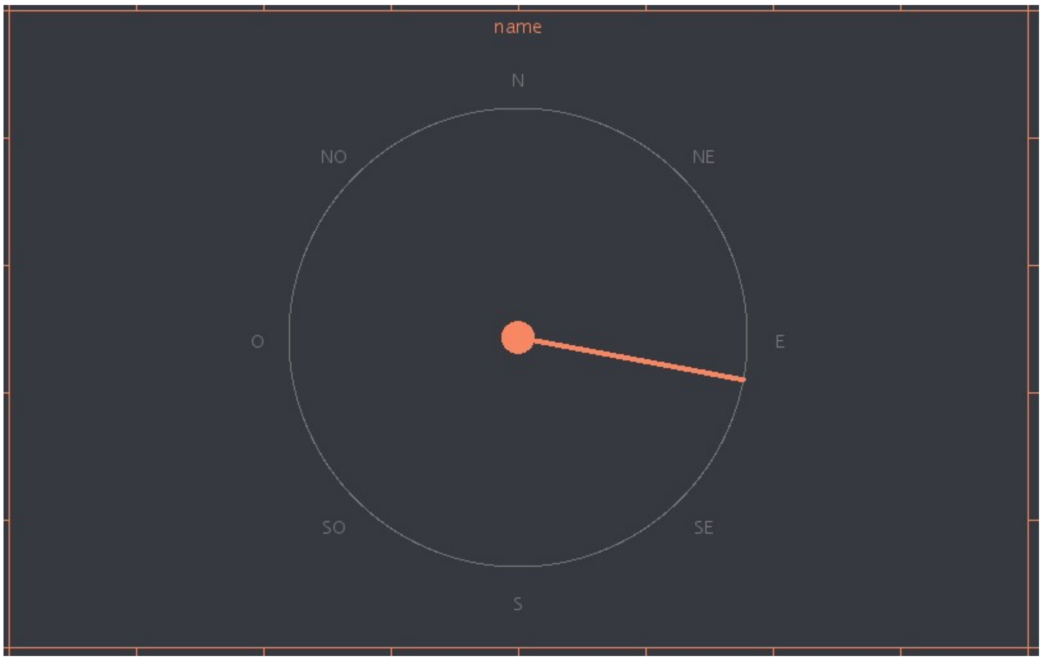
x

EXPRESSION

248
136
99
255

Color of compass

You can also hide the direction.



Data Loading

This program uses a serial connection in order to load and display the data. For that there is a thread that check the serial buffer. The first thing you need to do is to setup correctly the Serial Connection. Select the same baudrate as the device that will send data and choose the COM Port that refers to your device.

It's essential to send data within a well-defined frame structure, ensuring a predetermined order. The thread identifies data frames based on specific bytes that serve as markers for the start and end of each frame. As of now, the designated bytes for these markers are as follows:

- Start of frame: 0xBB
- End of frame: 0x0A, 0xAA, 0xFA

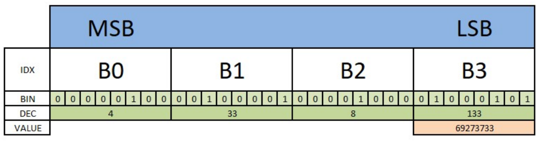
You need to be careful that these are to be taken into consideration when you read from the first bit to the last of your data. Should you need to modify these markers, you can conveniently do so at the outset of the "LoadData" file.

An example of a frame to be sent would be:

	Type	First bit	Last bit
Start of Frame	byte	0	8
Time	Unsigned Int	8	40
Pressure	Float	40	72
State	byte	72	80
End of Frame 1	byte	80	88
End of Frame 2	byte	88	96
End of Frame 3	byte	96	104

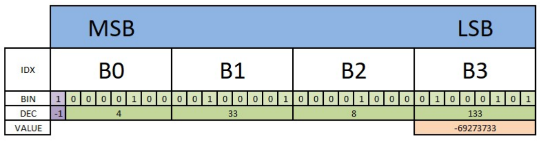
The data type are sent and processed in Big-Endian, it can be one of the following:

- Signed/Unsigned Int



	MSB			LSB
IDX	B0	B1	B2	B3
BIN	0 0 0 0 0 1 0 0	0 0 1 0 0 0 0 1	0 0 0 0 1 0 0 0	0 1 0 0 0 1 0 1
DEC	4	33	8	133
VALUE				69273733

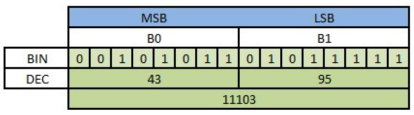
Unsigned Int



	MSB				LSB
IDX	B0		B1	B2	B3
BIN	1	0 0 0 0 1 0 0	0 0 1 0 0 0 0 1	0 0 0 0 1 0 0 0	0 1 0 0 0 1 0 1
DEC	-1	4	33	8	133
VALUE					-69273733

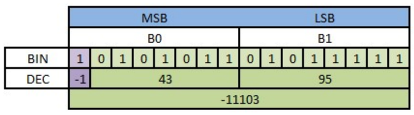
Signed Int

- Signed/Unsigned Short



	MSB	LSB
	B0	B1
BIN	0 0 1 0 1 0 1 1	0 1 0 1 1 1 1 1
DEC	43	95
	11103	

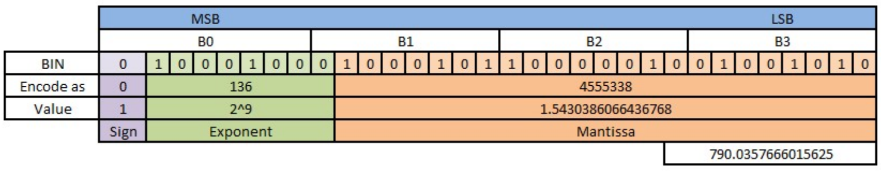
Unsigned Short (16 bits)



	MSB		LSB
	B0		B1
BIN	1	0 1 0 1 0 1 1	0 1 0 1 1 1 1 1
DEC	-1	43	95
	-11103		

Signed Short (16 bits)

- Float (32 bits)

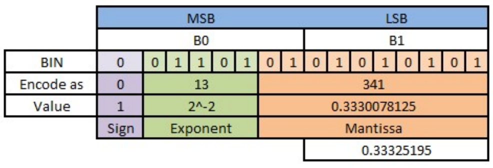


	MSB			LSB
	B0	B1	B2	B3
BIN	0 1 0 0 0 1 0 0	0 1 0 0 0 1 0 1	1 0 0 0 0 0 1 0	0 1 0 0 1 0 1 0

	Sign	Exponent	Mantissa
BIN	0	1 0 0 0 1 0 0 0	1 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
Encode as	0	136	4555338
Value	1	2^9	1.5430386066436768
			790.0357666015625

Float (32 bits)

- Half Float (16 bits)



	MSB	LSB
	B0	B1
BIN	0 0 1 1 0 1 0 1	0 1 0 1 0 1 0 1

	Sign	Exponent	Mantissa
BIN	0	0 1 1 0 1	0 1 0 1 0 1 0 1 0 1
Encode as	0	13	341
Value	1	2^-2	0.3330078125
			0.33325195

Half Float (16 bits)

With the following function in order to make the half float type:

```
C++ Function
// Union for an easier conversion
union FloatConverter {
  float f;
  uint32_t i;
};
void convertToByteArray(float floatValue, byte *byteArray, bool Half_Float) {
  if (Half_Float)
  {
    FloatConverter converter;
    converter.f = floatValue;
    uint32_t floatbuff = converter.i;

    uint32_t sign = (floatbuff >> 16) & 0x8000;
    uint32_t exponent = ((floatbuff >> 23) & 0xFF) - 127;
    uint32_t fraction = floatbuff & 0x007FFFFF;


    // special values
    if (exponent == 128) {
      // Infinity or NaN
      exponent = 31;
      if (fraction != 0) {
        // NaN
        fraction >>= 13;
        fraction |= 0x2000;
      }
    } else if (exponent == -127) {
      // Zero ou denormalized
      exponent = 0;
      fraction >>= 13;
    } else {
      // normalized
      exponent += 15;
      fraction >>= 13;
    }

    uint16_t halfValue = static_cast<uint16_t>(sign | (exponent << 10) | fraction);


    *byteArray++ = static_cast<uint8_t>(halfValue >> 8);
    *byteArray++ = static_cast<uint8_t>(halfValue & 0xFF);

  }
  else
  {
    uint8_t *p = (uint8_t *)&floatValue;
    *byteArray++ = *(p + 3);
    *byteArray++ = *(p + 2);
    *byteArray++ = *(p + 1);
    *byteArray++ = *p;
  }
}
```

An example of a program sending Data via Serial can be the following:

```
void loop() {
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize>0) {
    // received a packet
    // read packet
    String Belissama_frame = "";
    Belissama_frame += (char)start_of_frame;
    while (LoRa.available()) {
      Belissama_frame += (char)LoRa.read();
    }
    // print RSSI of packet
    int16_t LoraRSSI = LoRa.packetRssi();

    byte RSSIByte[2];
    convertToByteArray(LoraRSSI, &RSSIByte[0]);
    Belissama_frame += (char)RSSIByte[0];
    Belissama_frame += (char)RSSIByte[1];


    Belissama_frame += (char)end_of_frame1;
    Belissama_frame += (char)end_of_frame2;
    Belissama_frame += (char)end_of_frame3;


    Serial.print(Belissama_frame);
  }
}
```

How to modify

During this part, you will see some tag between brackets like *[filename-1]*, this tag is made to you to find easily where in the code is the area to modify. In this example file name mean the file where the area is, and the number (here 1) is to use a Ctrl-F to find the area in the file to modify. The tag are not in croissant order for the number. In addition the file name tag is given in each top file.

Add a widget

Firstly, at [VisUI-1], you have to add some variables, the Pimage of your widget logo, this image will be show in the menu to select the widget you want setup. Then lower in the variable creation, you have a lot of ControlP5 variable, in this area, add the controlP5 controllers do you need to setup your widget if they are not already here from the others widgets.

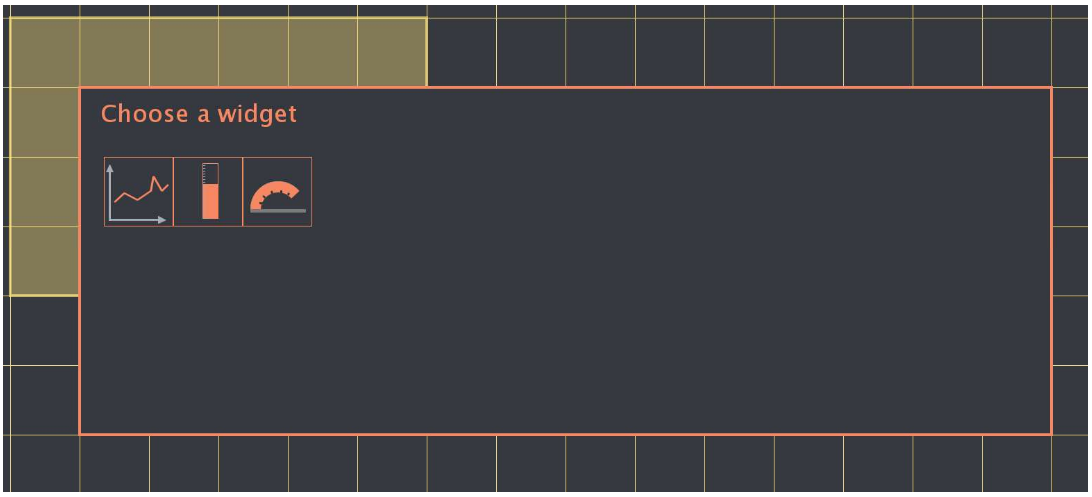
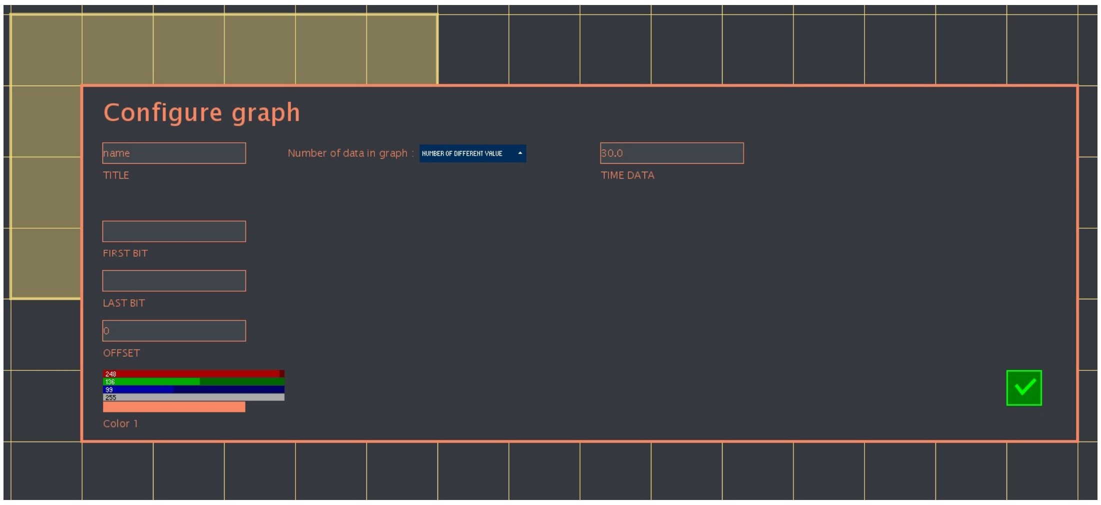


Figure 1, logo of all widgets



name
TITLE

Number of data in graph :

30.0
TIME DATA

FIRST BIT

LAST BIT

0
OFFSET

248
136
99
255

Figure 2, example of ControlP5 controllers

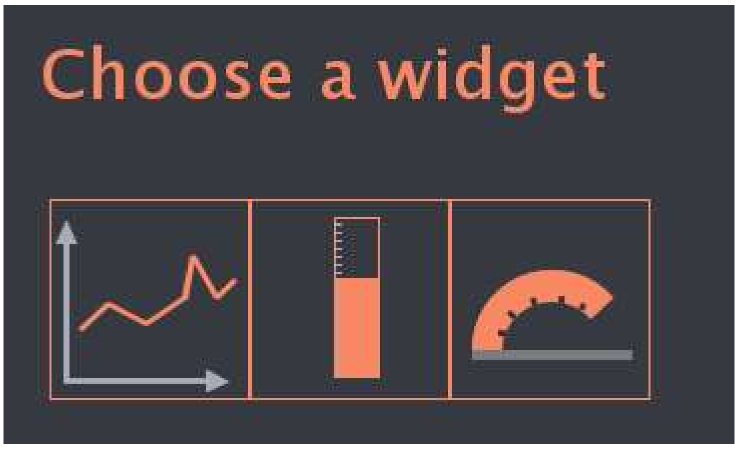
At [VisUI-1], you have to initialise the precedent variable, for example.

```
graph_logo = loadImage("graph_logo.png");
```

```
Time_data_Textfield = cp5.addTextfield("Time data").setPosition(-1000, -1000);
```

The position in the negative is because we don't want show it at the beginning.

At [VisUI-2], you have to place your widget logo, it's the position in the widget selector.



```
image(graph_logo, 2*size_block + 35 , 2*size_block + 100 );
image(bar_logo, 3*size_block + 35 , 2*size_block + 100 );
image(gauge_logo, 4*size_block + 35 , 2*size_block + 100 );
```

Lower you have to add an if in the case where the mouse is over your logo.

```
if( mouseX < 3*size_block + 35 && mouseX > 2*size_block + 35 && mouseY < 3*size_block + 100 && mouseY > 2*size_block + 100 ){
  textFont(big_font);
  fill( default_line_color );
  text("Graph", 2*size_block + 30 , 6*size_block + 80 );
}
if( mouseX < 4*size_block + 35 && mouseX > 3*size_block + 35 && mouseY < 3*size_block + 100 && mouseY > 2*size_block + 100 ){
  textFont(big_font);
  fill( default_line_color );
  text("Bar", 2*size_block + 30 , 6*size_block + 80 );
}
if( mouseX < 5*size_block + 35 && mouseX > 4*size_block + 35 && mouseY < 3*size_block + 100 && mouseY > 2*size_block + 100 ){
  textFont(big_font);
  fill( default_line_color );
  text("Gauge", 2*size_block + 30 , 6*size_block + 80 );
}
```

At [Constructor-1] you have to place a new if like that.

```
if( widget == k ){
```

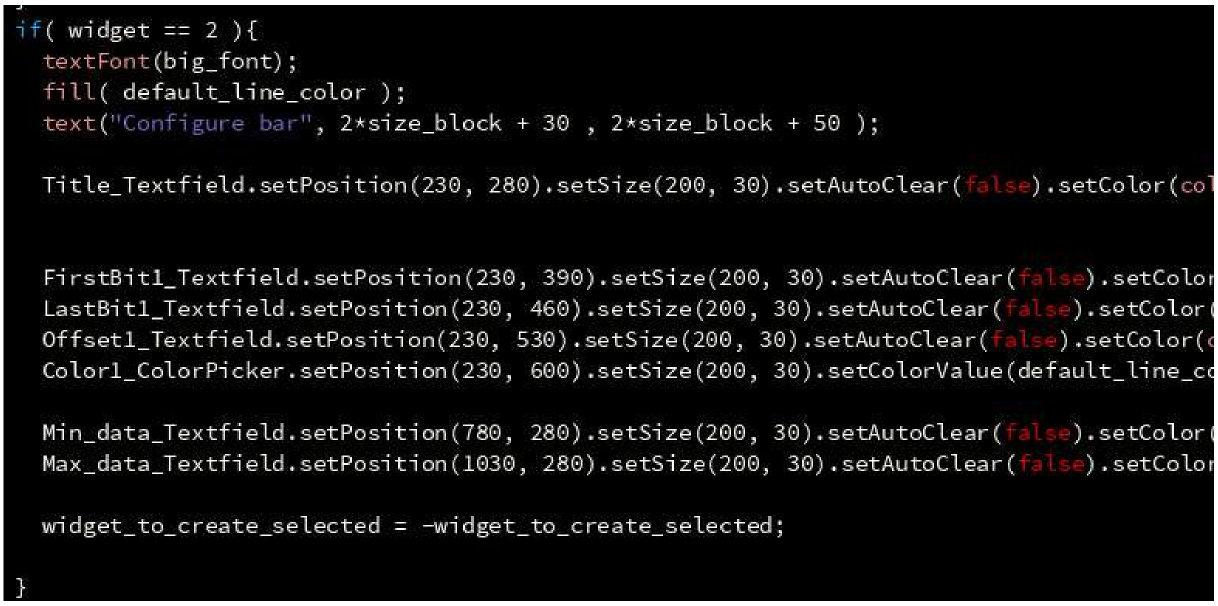
Where k is the id of your widget, to define the value of this idea, choose a value that is not use in this sequence of if.

You also have to make another if like that.

```
if( widget == -k ){
```

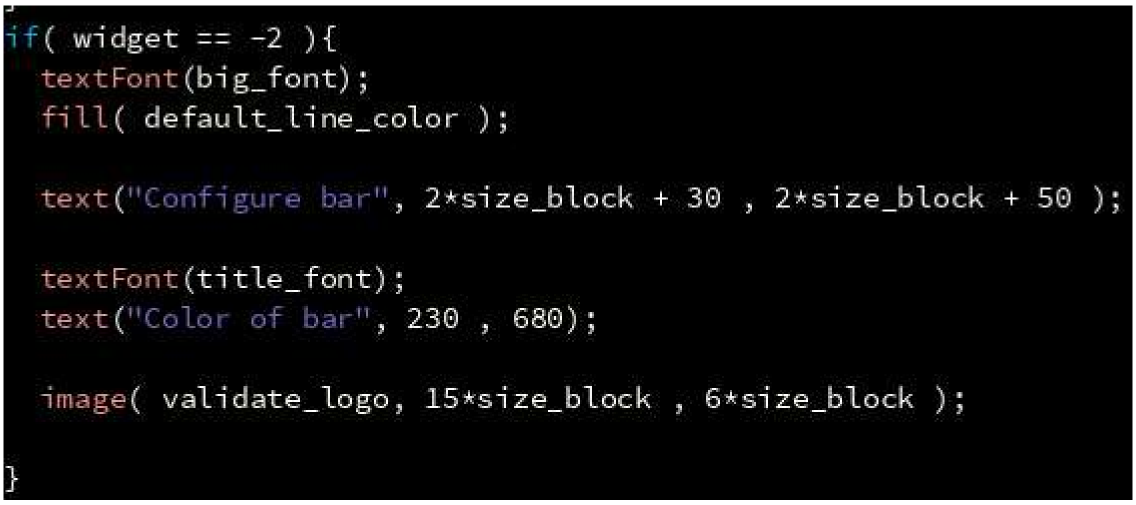
Now what do you will write in this two if statement? In the positive (if widget == k) you will write the instructions to apply only once. In the negative the instructions to apply every time during the configuration menu of your widget.

For example for the bar widget, the positive is.



```
if( widget == 2 ){
  textFont(big_font);
  fill( default_line_color );
  text("Configure bar", 2*size_block + 30 , 2*size_block + 50 );

  Title_Textfield.setPosition(230, 280).setSize(200, 30).setAutoClear(false).setColor(co


  FirstBit1_Textfield.setPosition(230, 390).setSize(200, 30).setAutoClear(false).setColor
  LastBit1_Textfield.setPosition(230, 460).setSize(200, 30).setAutoClear(false).setColor(
  Offset1_Textfield.setPosition(230, 530).setSize(200, 30).setAutoClear(false).setColor(c
  Color1_ColorPicker.setPosition(230, 600).setSize(200, 30).setColorValue(default_line_co

  Min_data_Textfield.setPosition(780, 280).setSize(200, 30).setAutoClear(false).setColor(
  Max_data_Textfield.setPosition(1030, 280).setSize(200, 30).setAutoClear(false).setColor

  widget_to_create_selected = -widget_to_create_selected;

}
```

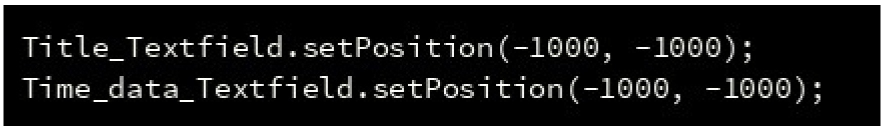
And the negative is.



```
if( widget == -2 ){
  textFont(big_font);
  fill( default_line_color );

  text("Configure bar", 2*size_block + 30 , 2*size_block + 50 );

  textFont(title_font);
  text("Color of bar", 230 , 680);

  image( validate_logo, 15*size_block , 6*size_block );

}
```

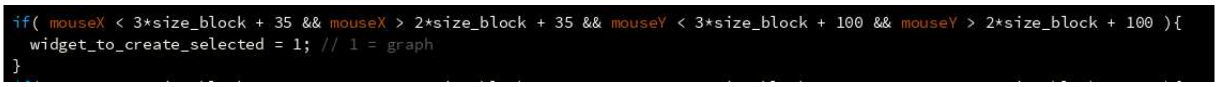
Do not hesitate to look at the code of the other widgets to better understand what is necessary.

At [Constructor-2], you will reinitialize all it's needed when you left the widget creator menu. For most of the case, you just will move your ControlP5 controller in the negative coordinates.



```
Title_Textfield.setPosition(-1000, -1000);
Time_data_Textfield.setPosition(-1000, -1000);
```

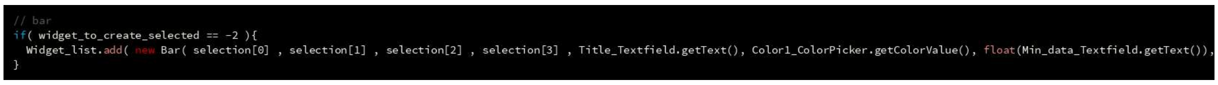
At [KeyInput-1] you will simply add an if statement for your widget. This code is execute when you choose a widget by clicking in the widget creator menu.



```
if( mouseX < 3*size_block + 35 && mouseX > 2*size_block + 35 && mouseY < 3*size_block + 100 && mouseY > 2*size_block + 100 ){
  widget_to_create_selected = 1; // 1 = graph
}
```

Don't forget to set widget_to_create_selected to your widget id.

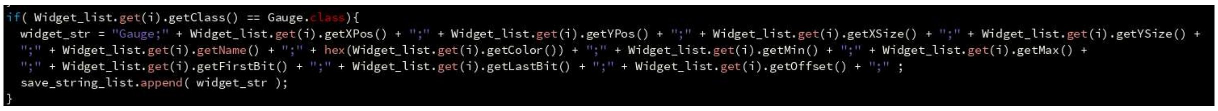
At [KeyInput-2] it's the lines who create your widget after user configure it.



```
// bar
if( widget_to_create_selected == -2 ){
  Widget_list.add( new Bar( selection[0] , selection[1] , selection[2] , selection[3] , Title_Textfield.getText(), Color1_ColorPicker.getColorValue(), float(Min_data_Textfield.getText()),
}
```

At this time you don't have write a file for your widget, so write a temporary code, and after you write it, comeback here.

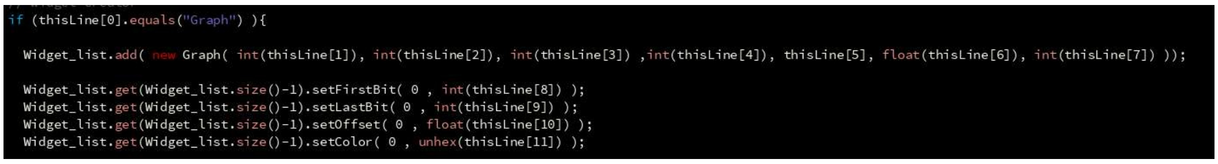
At [KeyInput-3] we will define the data to store in a txt file when the user click on the save button. The variable to store are different between the widgets, so don't hesitate to compare with other widget to understand how this part work.



```
if( Widget_list.get(i).getClass() == Gauge.class){
  widget_str = "Gauge;" + Widget_list.get(i).getXPos() + ";" + Widget_list.get(i).getYPos() + ";" + Widget_list.get(i).getXSize() + ";" + Widget_list.get(i).getYSize() +
  ";" + Widget_list.get(i).getName() + ";" + hex(Widget_list.get(i).getColor()) + ";" + Widget_list.get(i).getMin() + ";" + Widget_list.get(i).getMax() +
  ";" + Widget_list.get(i).getFirstBit() + ";" + Widget_list.get(i).getLastBit() + ";" + Widget_list.get(i).getOffset() + ";" ;
  save_string_list.append( widget_str );
}
```

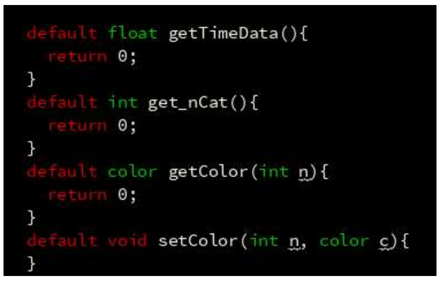
The variable are separated with comma and the string must end with a comma. To get this data you need some method (function of object) of your widget, we will see after where and how coded it. The order is not important BUT you have to use the same order in the save and in the load section.

At [KeyInput-4] you have the code to create the widget from the loaded file. You have to use the precedent order of your data and if it's needed, some method.



```
if (thisLine[0].equals("Graph") ){

  Widget_list.add( new Graph( int(thisLine[1]), int(thisLine[2]), int(thisLine[3]) ,int(thisLine[4]), thisLine[5], float(thisLine[6]), int(thisLine[7]) ));

  Widget_list.get(Widget_list.size()-1).setFirstBit( 0 , int(thisLine[8]) );
  Widget_list.get(Widget_list.size()-1).setLastBit( 0 , int(thisLine[9]) );
  Widget_list.get(Widget_list.size()-1).setOffset( 0 , float(thisLine[10]) );
  Widget_list.get(Widget_list.size()-1).setColor( 0 , unhex(thisLine[11]) );
```

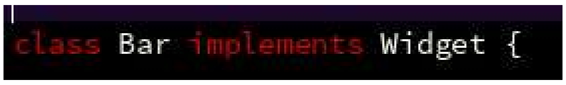
At [Widget-1] you have the default method, your widget method have to be write in another version in this file, you don't need to write something in, if you have to return something you can just return 0.



```
default float getTimeData(){
  return 0;
}
default int get_nCat(){
  return 0;
}
default color getColor(int n){
  return 0;
}
default void setColor(int n, color c){
}
```

Now you can create your widget file and code it, see an example.

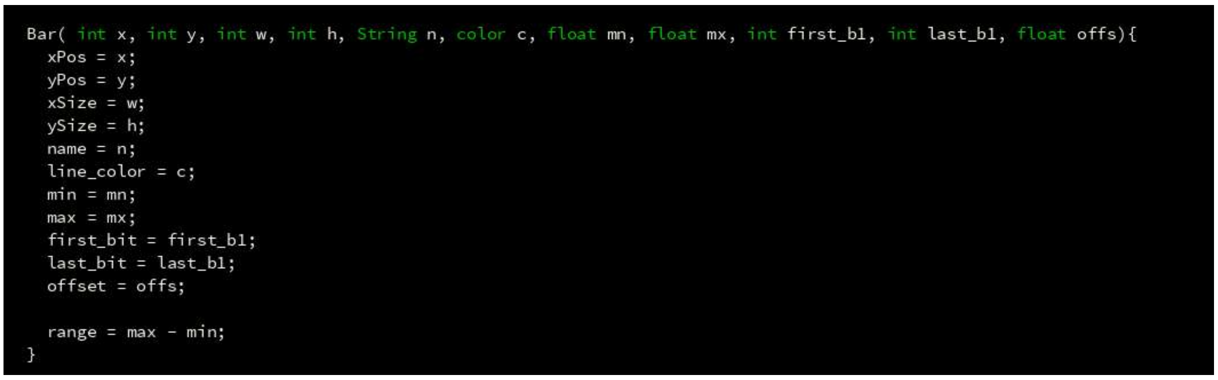
Firstly you have to define your widget on that way.



```
class Bar implements Widget {
```

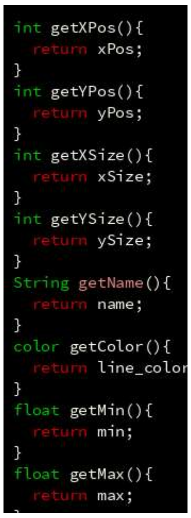
Follow that with your variables, for that and also all the following about the widget code, see a simple example like bar or gauge.

Then write the widget creator method.



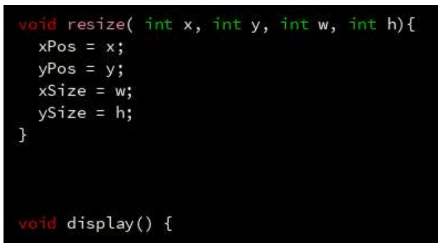
```
Bar( int x, int y, int w, int h, String n, color c, float mn, float mx, int first_bl, int last_bl, float offs){
  xPos = x;
  yPos = y;
  xSize = w;
  ySize = h;
  name = n;
  line_color = c;
  min = mn;
  max = mx;
  first_bit = first_bl;
  last_bit = last_bl;
  offset = offs;

  range = max - min;
}
```

And all the methods do you need.



```
int getXPos(){
  return xPos;
}
int getYPos(){
  return yPos;
}
int getXSize(){
  return xSize;
}
int getYSize(){
  return ySize;
}
String getName(){
  return name;
}
color getColor(){
  return line_color
}
float getMin(){
  return min;
}
float getMax(){
  return max;
```

Finally don't forget this 2 very important method, resize (to move or resize your widget) and display.



```
void resize( int x, int y, int w, int h){
  xPos = x;
  yPos = y;
  xSize = w;
  ySize = h;
}



void display() {
```

The display function is the most important, because it's here you will write all the visual code of your widget. It's also the most different part between the widget, so be creative!