Owners Manual – GECO - HORA MUSIC

Table of contents

1. What is Geco ?

Geco is a visual VCV plugin maker. It allows to create patches and panel positionning and use these to generate all the needed files/code to compile the plugin/module just like any other coded manually.

It offers premade objects for :

- DSP (oscillators, env, filters, effects, previous sample,...)
- Mathematical operations
- Counter
- Variables, arrays, Strings
- Comparison, conversion
- Non volatile saving (json),...
- Widget (knob, button, switch, rotary switch, led, input and ouput)
- Screen
- WaveTables
- Buffer
- ...

Advanced users can create their own blocks by using the vult transcompiler and a template that allows to create new objects for Geco.

This software is made for :

- People that has never code a line of code.
- Developers who want to avoid annoying parts of the development.
- Real hardware module's developers that want to copy and distribute or just test a software version of their ideas for coming or existing modules.

Geco is currently in Beta (0.1 version), so by using it, you are an “early tester”. Some imporvements and a lo of new features will come until the 1.0 version.

2. Prequisities :

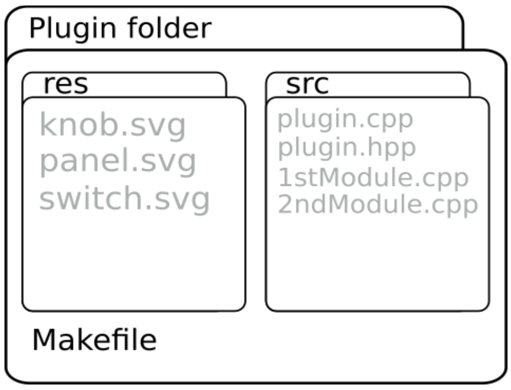
You have to know some basics about VCV to be comfortable with Geco :

2.1. VCV files organization

The « VCV rack » plugins are store in a folder called plugins that is in your rack directory :

- Windows: My Documents/Rack/
- MacOS: Documents/Rack/
- Linux: ~/.Rack/

The plugins in this directory are organised like this :



The « res » folder contains the graphisms (svg files) for the widgets (knob, switch, input,...). The « src » folder contains the code files (the « sources »), the plugins generally contains at least one cpp file and one hpp file for the plugin and one cpp file for each modules. The Makefile is used by the compiler, just name correclty your plugin slug and it will be write correctly (look at the VCV rack manual if needed) .

2.2. VCV files organization

The plugin folder automatically created by Geco are organised like explain in section 2.1. Then, you can compile the generated plugin just like any others. This manual won't explain how to compile plugins. This step is depending on

your OS and is explained on the VCV github page and manual :
https://vcvrack.com/manual/
https://github.com/VCVRack/Rack

2.3. Think like a developer

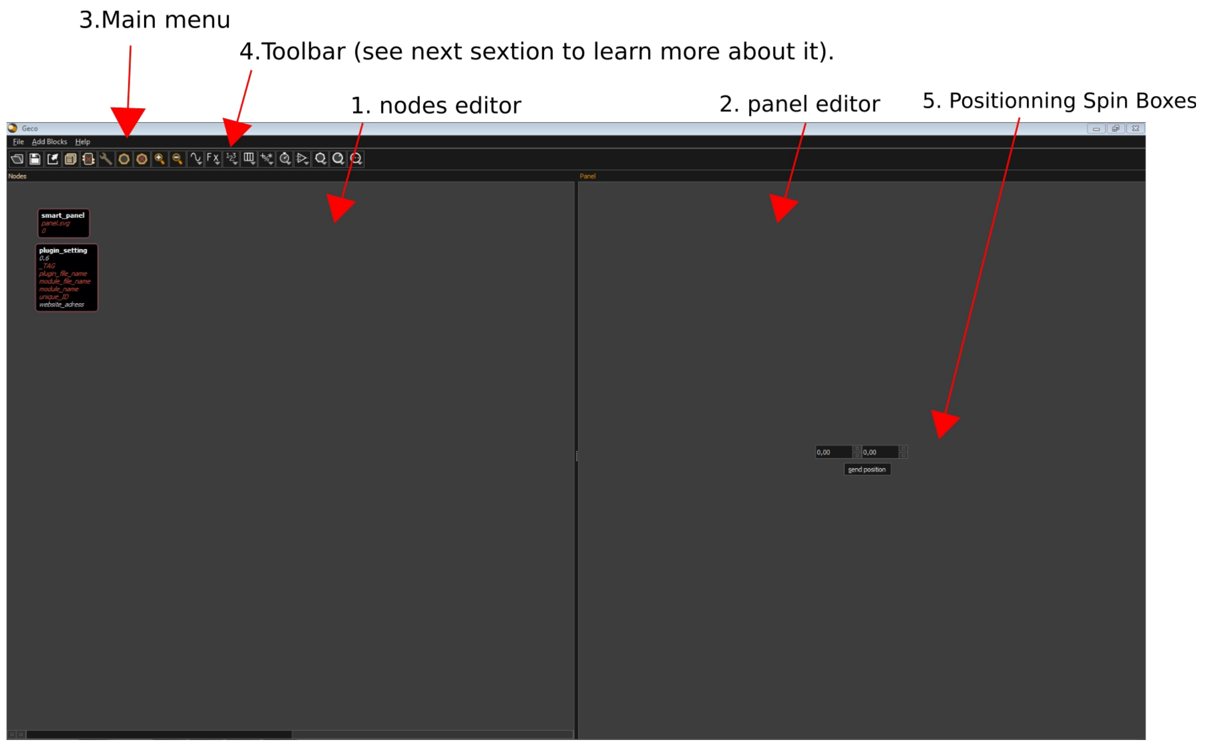
Even if it isn't needed to code when using Geco, users can deal with objects like arrays, integer, float, comparators,... These are used when coding but can be unknown by neophytes. If you don't know what these are, you have many ways to learn a bit about programming:

- Read a bit of doc about programming basics.
- Use pure data or max msp that use also numbers, comparators, and allows to test these in real time.
- Use any other visual programming laguage that is simple enough for beginners.

You will need to be carefull with some objects naming, these are automatically named but if you name these manually you have to avoid same naming for many objects. If some objects are sharing the same name a warning will appear when exporting telling the user wich name is used by more than one object.

Finally, keep in mind that Geco offer freedom of creation that make it possible to create modules that don't do what you were expecting or cannot be compiled. Verify your patch, read the error message(s), it is generally easy to identify errors if you understand your patch and programming basics.

3. Basic operations



3.1. The panel editor :

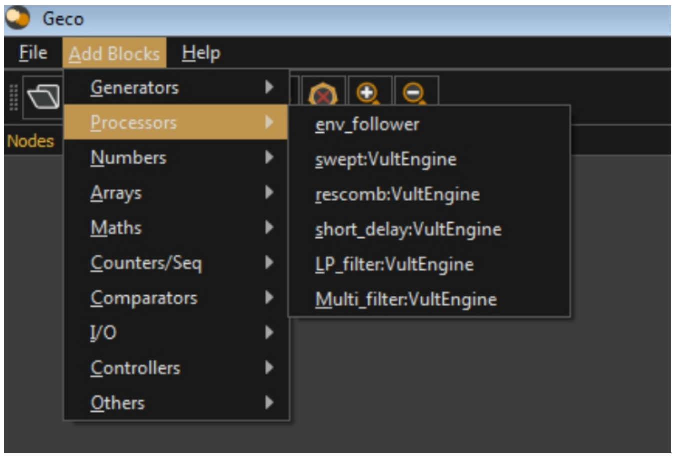
The nodes editor is where blocks are displayed and where the patch is made. It allows to use blocks relative to DSP or widgets, patch these and set their properties.

3.2. The panel editor :

The Panel editor display the widgets and the module's panel and allow to set their position by dragging it with the mouse or by using the two spin boxes.

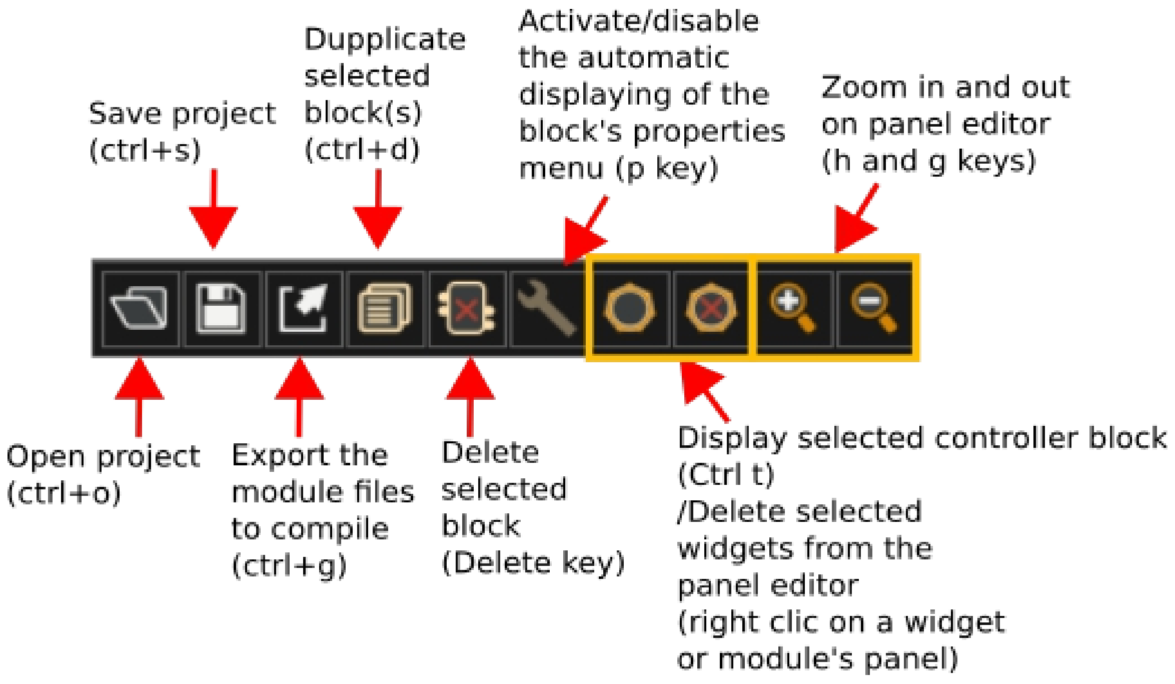
3.3. Main menu :

- The « file » sub-menu allow to save or load a new Geco's project.
- The « help » sub-menu display this owner manual.
- The « add blocks » sub-mneu allow to add blocks that are stored in categories :

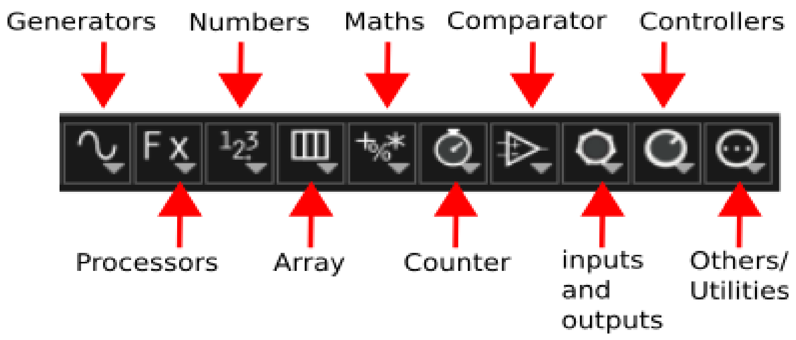


3.4. The toolbar :

The toolbar or the corresponding shortcuts can be used for main editing features :


Save project (ctrl+s)
Dupplicate selected block(s) (ctrl+d)
Activate/disable the automatic displaying of the block's properties menu (p key)
Zoom in and out on panel editor (h and g keys)
Open project (ctrl+o)
Export the module files to compile (ctrl+g)
Delete selected block (Delete key)
Display selected controller block (Ctrl t) /Delete selected widgets from the panel editor (right clic on a widget or module's panel)

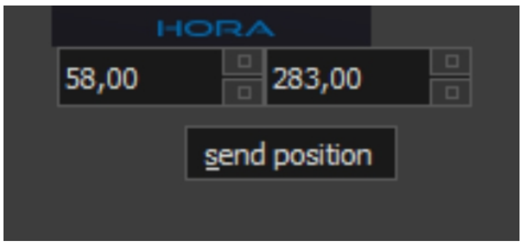
It also allows to add blocks :



The blocks are store exactly like in the « add blocks » submenu (explain at section 3.3). The categories and blocks description are avaiblable in section 5 of this manual.

3.5. The spinBoxes :

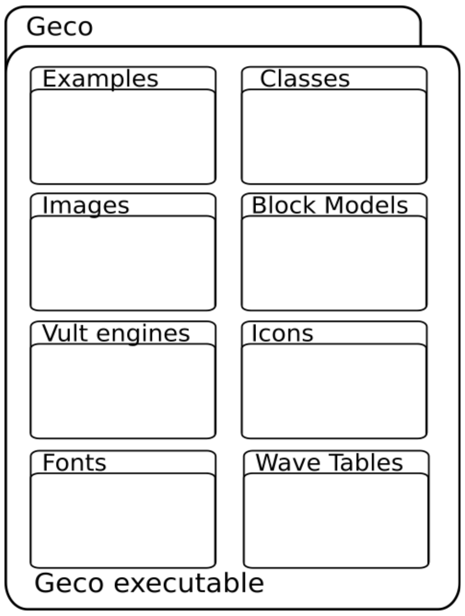
The spinboxes can be used to set the position of the widget, set the spin boxes then press « send position » to make the setting effective on the current selected widget.



3.6. Select many items :

You can select many blocks or widget by pressing and holding the « ctrl » key and then select these with the mouse (left clic). This can be usefull to transfer many controllers on the panel editor or move/dupplicate many blocks.

3.7. Geco files :



The Geco Folder comes with many folders/files that are organized like this :

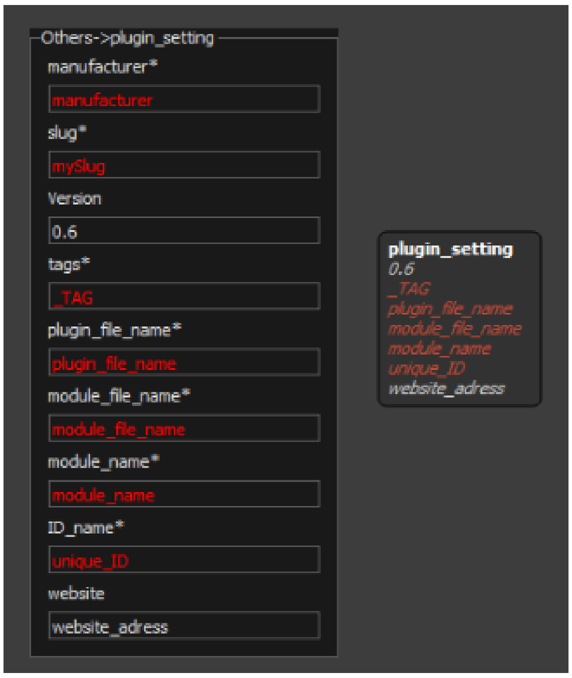
- The « Examples » folder is where Geco examples projects are stored.
- The « Classes » folder contain the C++ classes use by geco blocks.
- The « Images » folder contains the svg files of the wigets (knob, switch, ...) and panels. Users have to put their personnal svg files in this folder to be usable with Geco.
- The « Block Models » folder contains the blocks model files (usefull for advanced users that wat to create their own blocks and for users that want to add new blocks from other developers).
- The « Vult Engines » folder contains the vult Engines used by Geco (usefull for advanced users that wat to create their own blocks and for users that want to add new blocks from other developers).
- The « Icons » folder contains the icons of the Geco toolbar (useless for any users).
- The « Fonts » folder contains the fonts that can be used by the screen widget.
- The « WaveTables » folder contains the wave table that can be use by the wavetable block (see section 5. for more about the blocks)

4. Create a module

4.1 Set Your plugin :

On Geco launch, Geco open an empty projects that contains the two always needed blocks : « plugin_setting » and « smart_panel ».
The plugin_setting block let you set the needed plugin's informations :

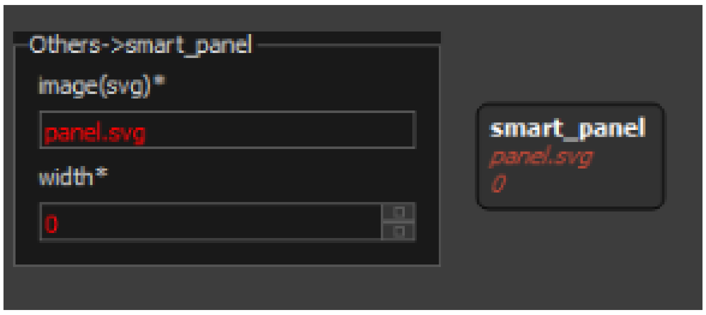
- Manufacturer : The name of your company.
- Slug : The slug of your plugin (see VCV rack manual if needed)
- Version : The verison of the plugin (let this on the default value in most of the cases).
- Tags : Enter the tags for your plugin (see VCV rack manual if needed).
- Plugin File name : The name of the main folder of the plugin and the plugin's CPP and HPP files.
- Module File Name : The name of the module's CPP file.
- Module name : The name of the module.
- ID_name : The unique ID Name of the plugin.
- Website : The website adress of the company (www.yoursite)


Others->plugin_setting
manufacturer*
manufacturer
slug*
mySlug
Version
0.6
tags*
_TAG
plugin_file_name*
plugin_file_name
module_file_name*
module_file_name
module_name*
module_name
ID_name*
unique_ID
website
website_adress

plugin_setting
0.6
_TAG
plugin_file_name
module_file_name
module_name
unique_ID
website_adress

4.2. Set Your panel :

The automatically added smart_panel block can be set but you need to have a svg file of your panel in the images folder of Geco as explained in section 3.7. (a good way to create panel for noobs is to open an existing panel svg file from the « res » folder of an open source plugin and modify it with inkscape or any other software that can do it).



Others->smart_panel

image(svg)*

panel.svg

width*

0

smart_panel

panel.svg

0

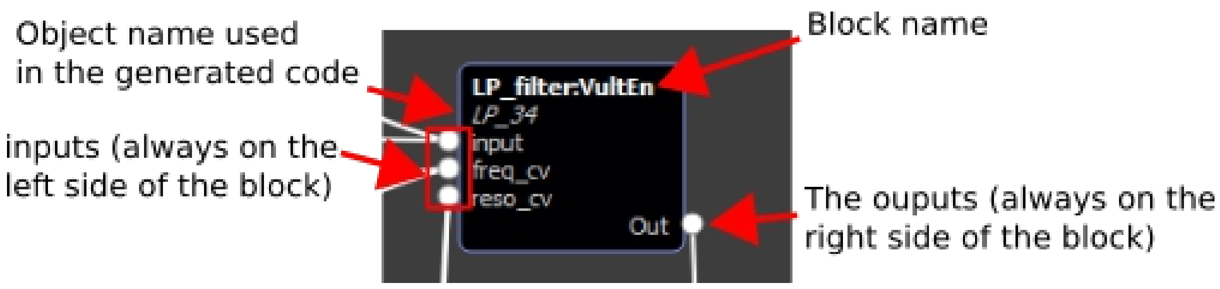
- Image (svg) : Enter the name of the panel of the module that you have to add to the images folder as explained in section 3.7.
- width : This is calculate automatically from the width of the panel's graphics (.svg) but you can change it manually if needed.
  Note : You can activate or disable the automatic display of the properties menu of the selected block by pressing the p key or using the toolBar (explained in section 3.4)

4.3. Add blocks and create a patch:

To add a block use the toolbar or the « add blocks »menu.

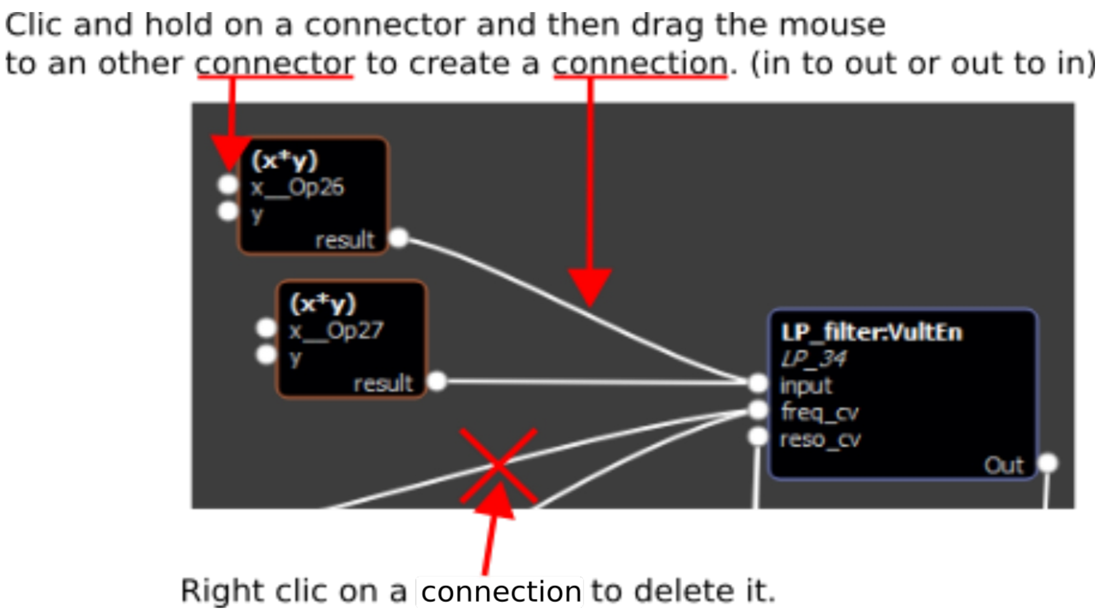
The new blocks will appear near the current selected block or at point 0, 0 if no blocks are selected.

The blocks descritption is available in section 5. This said, here is a generic block description.


Object name used in the generated code
inputs (always on the left side of the block)
Block name
The ouputs (always on the right side of the block)
LP_filter:VultEn
LP_34
input
freq_cv
reso_cv
Out

- LP_filter:VultEngine : This is the main block name.
- LP_34 : Name of the object that will be used in the code to write part relative to this block. Geco automatically set this parameter. You can manualy change it from the properties panel of the block but as you may assume this name has to be unique at least for all blocks of the same type, if there's « doublons » Geco will tell you it and give the list of the objects names used by many blocks on plugin exporting.
- Input, freq_cv and reso : These port are the inputs of the blocks, these cannot be renamed and can receive any other block outputs.
- Output : This is the output of the block, it can be link to one or many other blocks inputs.

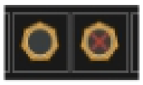
Once you had add some blocks you will have to create a patch by connecting these :


Clic and hold on a connector and then drag the mouse to an other connector to create a connection. (in to out or out to in)
(x*y)
x__Op26
y
result
(x*y)
x__Op27
y
result
LP_filter:VultEn
LP_34
input
freq_cv
reso_cv
Out
Right clic on a connection to delete it.

Note : If many numbers (float or integer) are send to the input they are automatically summed. Sending of many strings to an input will give unexpected results, to concatain strings you should use the « stringpack » block (in maths category) and then send the concatained strings to the input.

4.4. The panel editor:

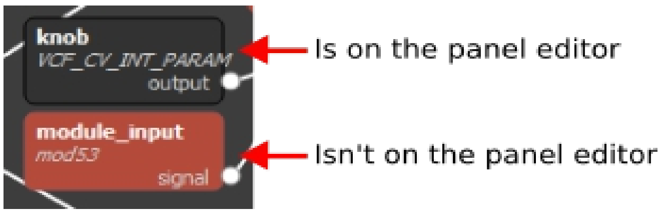
You can add controllers to the panel editor by selecting controllers, inputs or ouputs block(s) and then press ctrl + t or use these buttons :



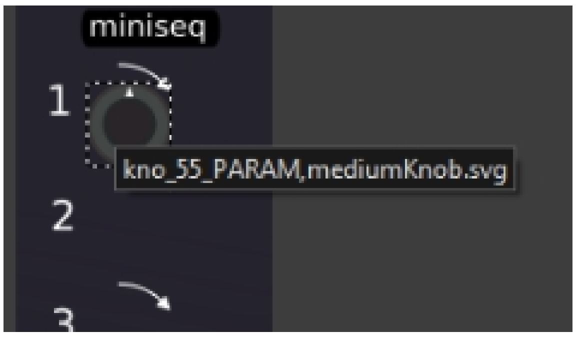
Display selected controller block (Ctrl t)
/Delete selected widgets from the panel editor
(right clic on a widget or module's panel)

Note : To delete a controller from the panel editor you can use the tool bar or right clic on it)

Controllers blocks that aren't on the panel yet, colored in red, are easy to spot:



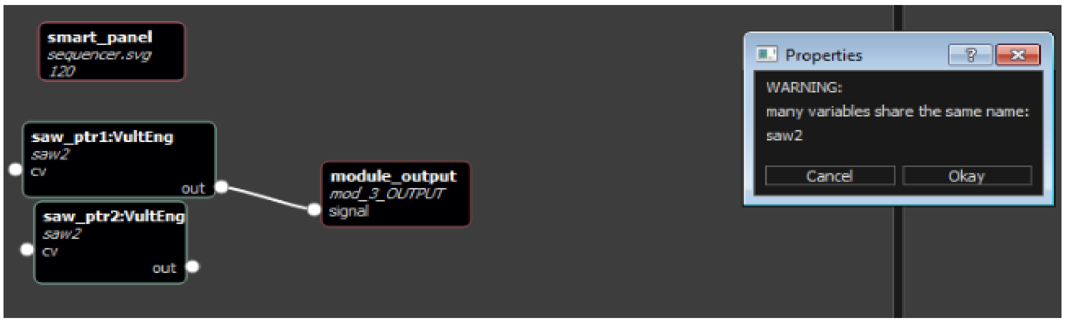
To easily remind wich controllers is link to wich block, a tooltip appear when the mouse is over the controller :



4.4. Export a Plugin:

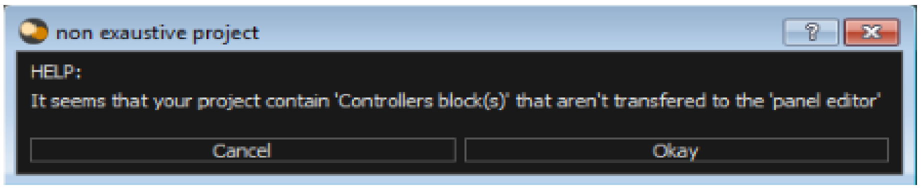
Use the « export » button of the toolbar or press ctrl+g to export the plugin code.

If your project contains object sharing the same name, a warning will appear showing a list of the concerned names.



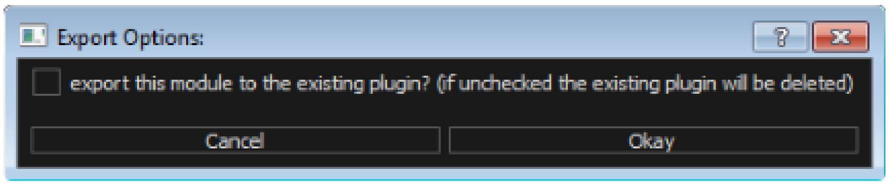
WARNING:
many variables share the same name:
saw2

If your project contains block(s) of controllers (knobs, switches, outputs, ...) that aren't transfered on the panel editor a warning will appear too. The concerned blocks are display in red color (as explained in section 4.4.)



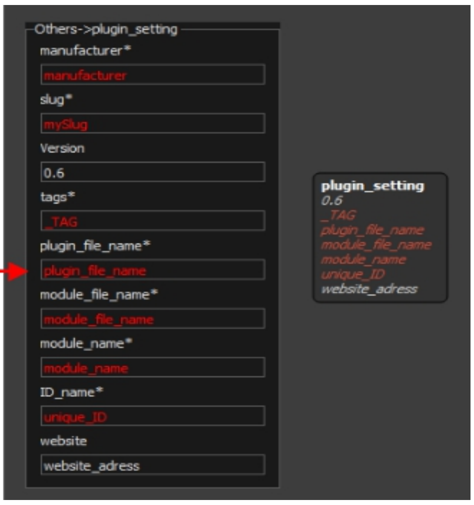
HELP:
It seems that your project contain 'Controllers block(s)' that aren't transfered to the 'panel editor'

If you want to create a plugin made of many modules, you have to give to this the same plugin file name and export these one after another. Geco will ask you if you want to add the module to the existing plugin. The plugin will be erase if you keep the box unchecked but it is safer to delete the plugin manually if you're exporting the first module of this.



export this module to the existing plugin? (if unchecked the existing plugin will be deleted)

By using the same plugin file name the modules will be exported in the same plugin folder to constitute plugin made of many modules .



Others->plugin_setting

manufacturer*: manufacturer

slug*: mySlug

Version: 0.6

tags*: _TAG

plugin_file_name*: plugin_file_name

module_file_name*: module_file_name

module_name*: module_name

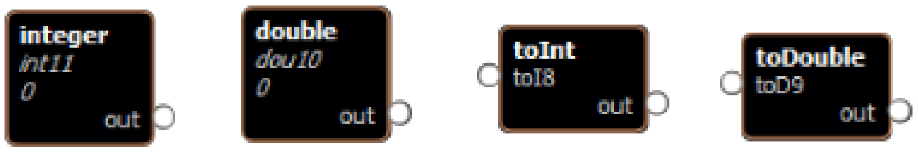
ID_name*: unique_ID

website: website_adress

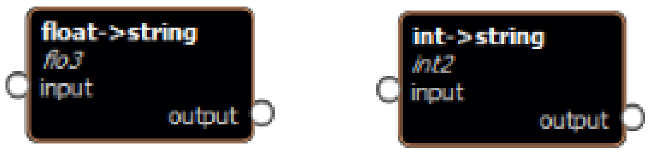
5. Blocks description

Numbers :

Geco allow to add floats and integers with an constant value and use these with any block input. It also allows to convert float to integer and integer to float for using of modules that need a specific type of variable. Integer and float can also be converted to string (see more about geco string in the arrays section).

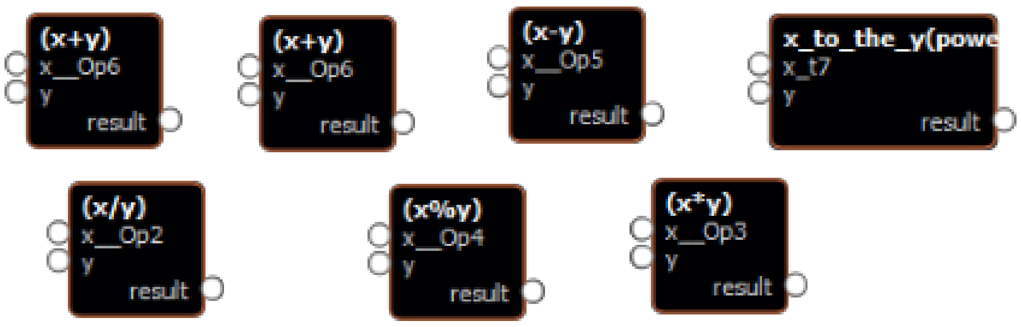


As geco offer objects working with string variables, the numbers section offer int to string and float to string converter too.



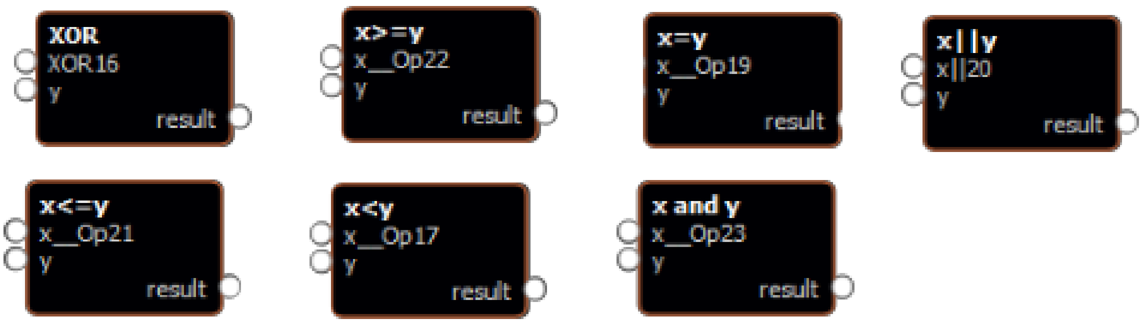
Maths :

The available mathematical operations are addition, difference, division, multiplication, modulo, power, square root and stringpack (that allow to concataine up to five strings).
The first ouput is x and the second is y, the result output is the result of the opreration on x and y as mentioned on the top of the block.



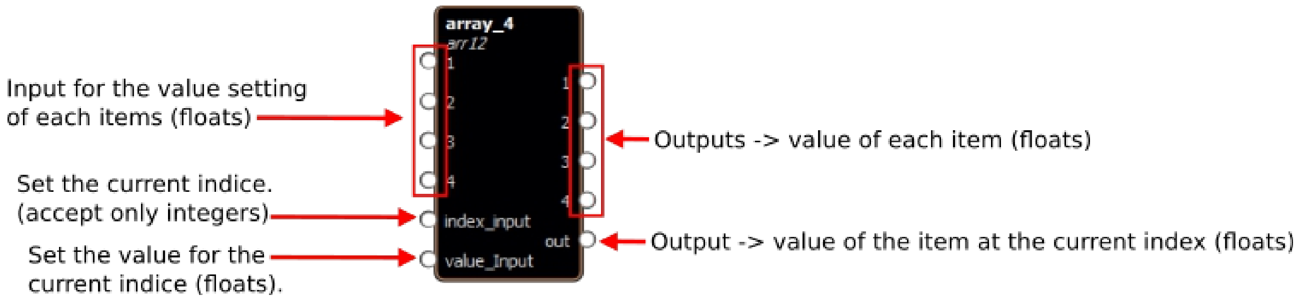
Comparators :

The available comparators are XOR, or, and, <, >, <=, >=, ==.
Concerning the inputs 0 mean false and any other value mean true. The output give 1 for true state and 0 for false state.

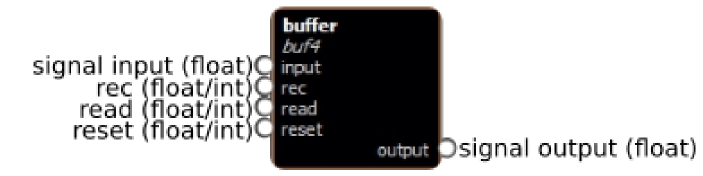


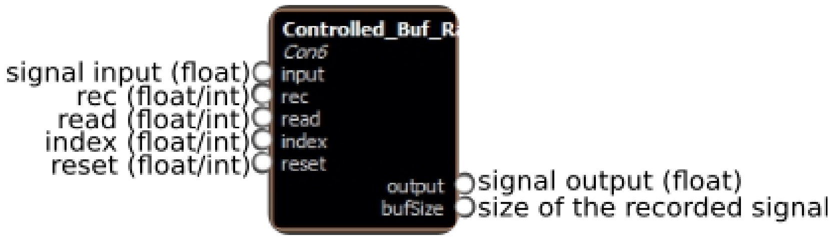
Arrays :

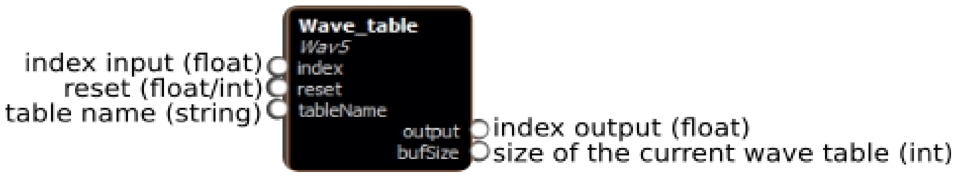
The arrays are usefull for sequencers, routing, line selection,...
Geco offers arrays of 4, 8, 16, 32 and 64 items and his property panel allows to ser the his exact size. These work like this :


array_4
arr12
1
2
3
4
index_input
value_Input
1
2
3
4
out
Input for the value setting of each items (floats)
Set the current indice. (accept only integers)
Set the value for the current indice (floats).
Outputs -> value of each item (floats)
Output -> value of the item at the current index (floats)

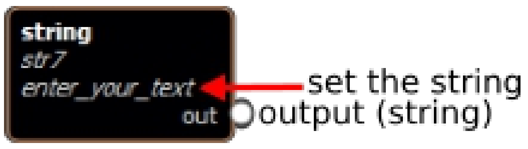
The arrays setion offers wave table and buffers too. Rec read and reset input are active if the input is different to 0. The controlled buffer « raw » simply read the value of the current index but The Wave table and the regular controlled buffer offer linear interpolation. The wave tables are stored in the « WaveTables » folder. They has to be at the same format (GWT). To create your own you need to create a file with the .gwt extension. This file as to contain the value of the table coma seperated (look at the existing wavetables to get an example).





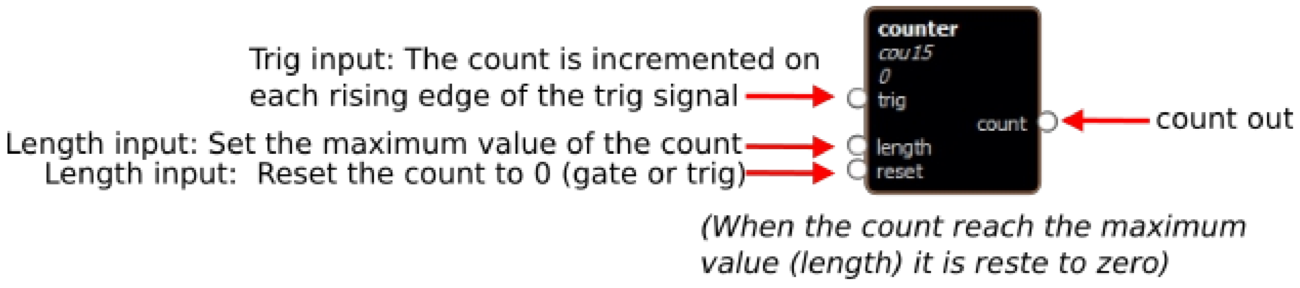


The arrays section offers string too. In Geco, strings are characters arrays limited to 100 characters.

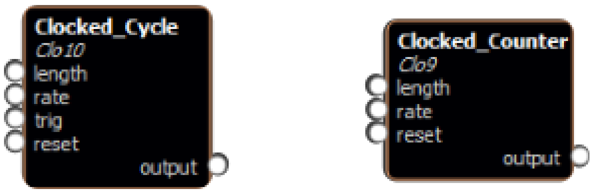


Counters :

The counter is usefull for frequency division of clock signals, array's indice incrementation, event counting,... These work like this :

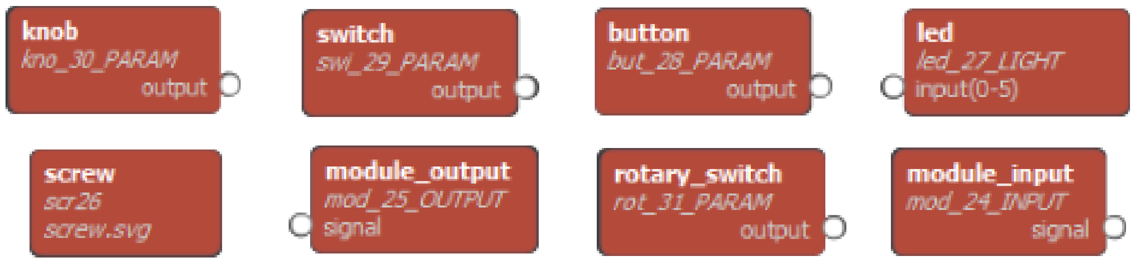

counter
cou15
0
Trig input: The count is incremented on each rising edge of the trig signal
trig
count
count out
Length input: Set the maximum value of the count
length
Length input: Reset the count to 0 (gate or trig)
reset
(When the count reach the maximum value (length) it is reste to zero)

The clocked Counter and clocked cycle are usefull for wavetable index, buffer index, integrator, phasor ,... (look at the wavetable example). The length is the maximum value of the counter, the rate is the value that is incremented on each step and the reset value reset the counter to zero. The trig input on the clocked cycle launch the counter for one cycle, the clocked counter is continually running.



Inputs, outputs and controllers:

The input, outputs and controllers have to be display on the panel editor by pressing the button in the toolbar or use the ctrl + t shortcut. You can use your own graphics (svg files) by adding these in the images folder of Geco (explained in section 3.7).

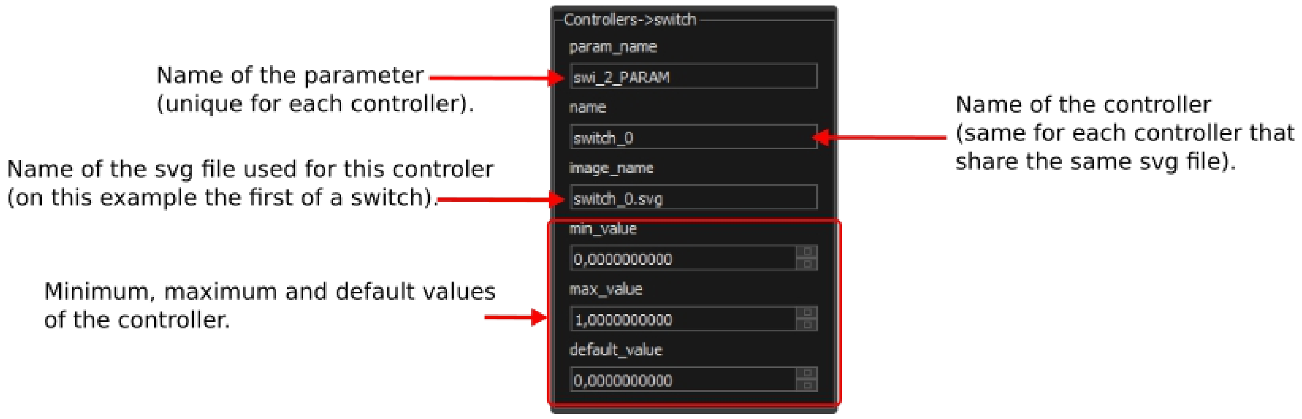


The parameter name is automatically but you can set this manually, it is also display on the tooltip of the contollers in the panel editor.

The maximum, minimum and defalut value can also be set. The number of positions of the switches and rotary switches depend on it too.
E.G : A switch with min value = 0 , max value = 2 and default value = 1 will create a three positions's switch set at the center postion.

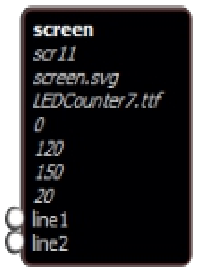
When unsing your own graphics with switches you have to name this with a _0, _1, _2,...suffix.
E.G : mySwitch_0.svg, mySwitch_1.svg and mySwitch_2.svg for a three positions switch.

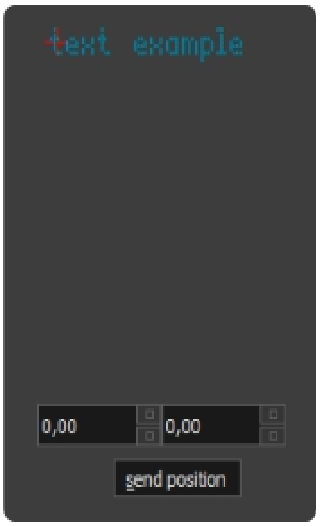
You just have to write the name of the first svg file (mySwitch_0) in the image name filed of the property panel and all the other will be used automatically for their corresponding position.


Controllers->switch
param_name
swi_2_PARAM
name
switch_0
image_name
switch_0.svg
min_value
0,0000000000
max_value
1,0000000000
default_value
0,0000000000

Name of the parameter (unique for each controller).
Name of the svg file used for this controler (on this example the first of a switch).
Minimum, maximum and default values of the controller.
Name of the controller (same for each controller that share the same svg file).

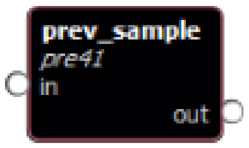
Geco offer screen widget too. The text color, size and font can be set from the property panel of it. The font has to be in the font directory of geco (as explaion in section 3.7.)
The two line inputs receive the string to display on the screen (the screen is used in the wave table example).


screen
scr11
screen.svg
LEDCounter7.ttf
0
120
150
20
line1
line2



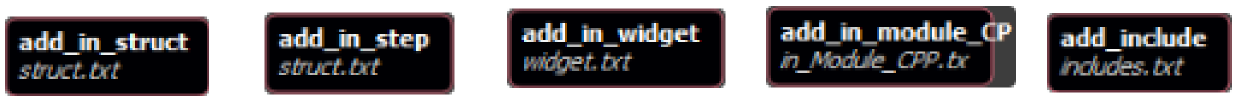
Previous sample (others) :

The previous sample receive a value and give his value at the previous step (n-1).



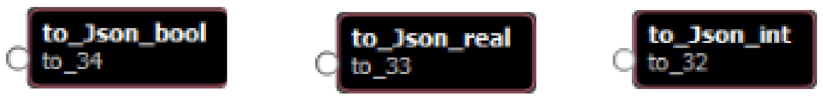
Add_in_... :

The add_in_struct, add_in_step, add_in_include,... allows to add code in the structure of the module, the step function of the module,.. or add include. The code parts that to be added have to be in a text file placed in the Geco folder.
The name of the used text file is settable from the property panel of the blocks.
This blocks are still a bit more experimental and could be used by advanced users.

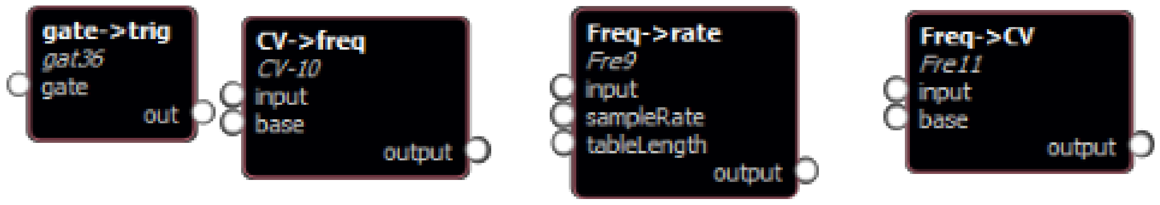


To Json :

Even if controllers positions are automatically saved in VCV projects, sometime it is usefull to save other values in a non volatile memory.
Json blocks allow to do it and save the incoming booleans, integers and floats.



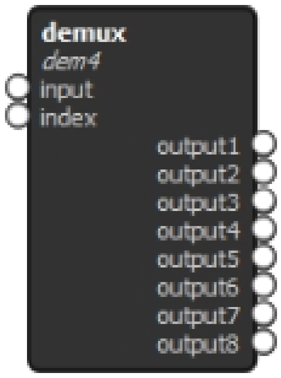
Converters :

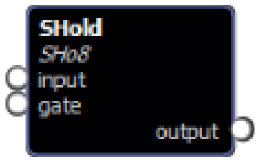


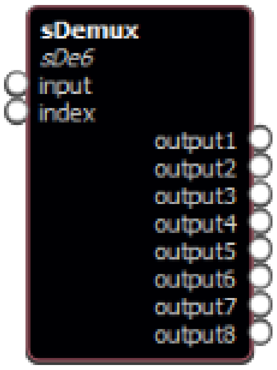
Geco offer many signal convertors : gate->trig, CV->frequency, frequency -> CV, frequency->rate.

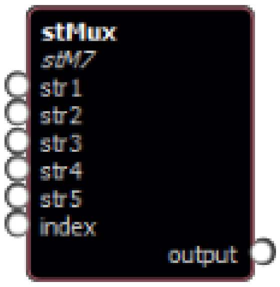
Routing :

Geco offer many signal routing tools : Demultiplexer, string multiplexer, string demultiplexer, Sample and hold, ...



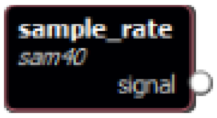






Sample rate :

This block gives the samplerate in hertz(genarally 44.100, 48.000 or 96.000 Khz). This can be usefull if you create a patch that react differently depending on the sample rate (set in VCV audio core module).



Generators :

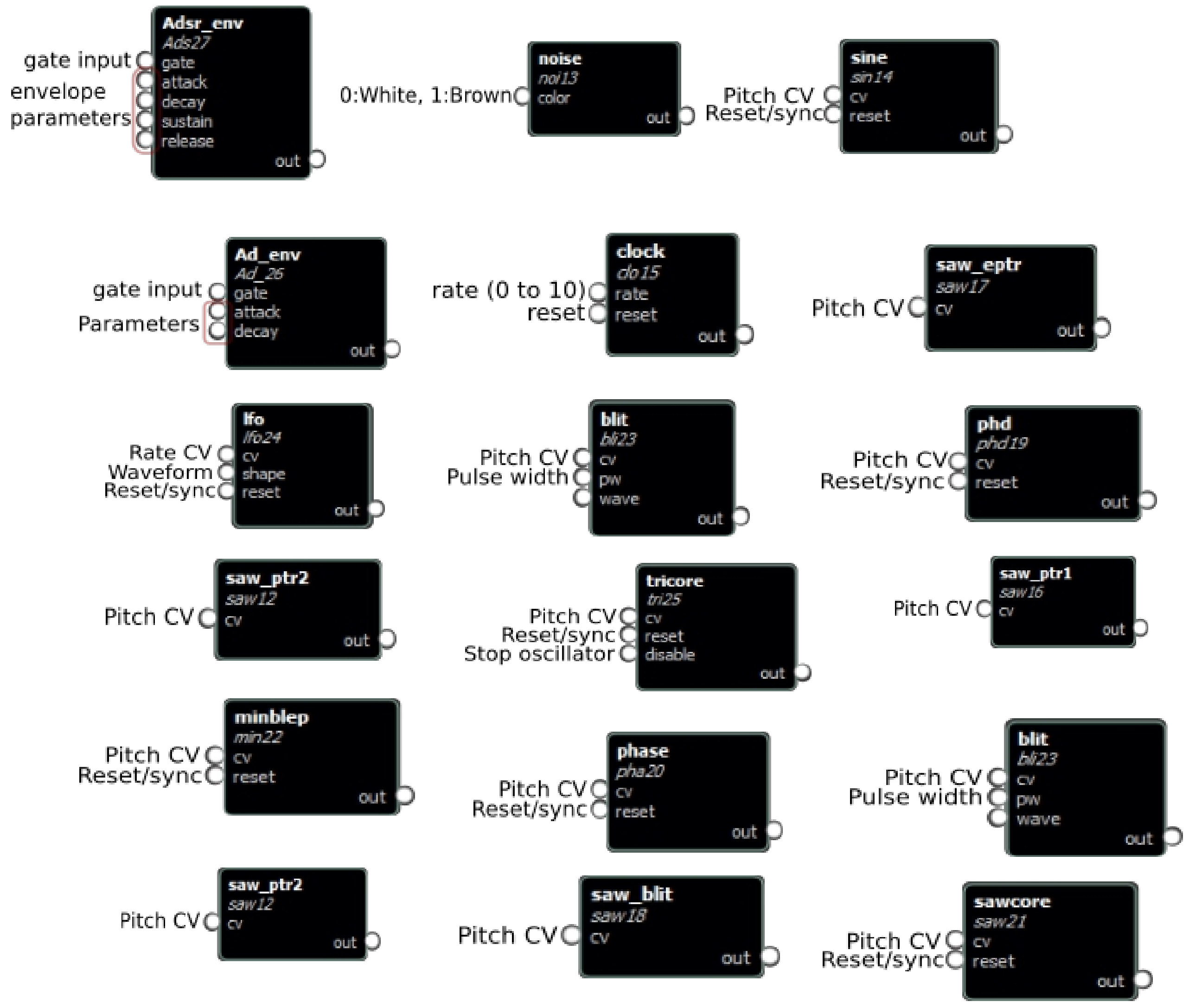
Generators and blocks relative to DSP work with value between -1 and 1 for audio and generally between 0 and 1 for control inputs.
This category contain oscillators, envelope generators, LFOs, clock,...

You can find more informations about the oscillators based on the Vult DSP transcompiler on this github page
https://github.com/modlfo/vult/tree/master/examples/osc

The vult objects names can be confusing :

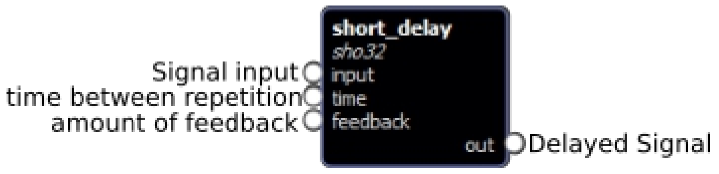
- **BLIT :** Generates a BW-limited pulse train given the phase and the number of harmonics.
- **Saw_ptr1 :** Saw oscillator using polynomial transition regions (PTR, W = 2)
- **Saw_ptr2 :** Saw oscillator using polynomial transition regions (PTR, W = 1)
- **Saw_eptr :** Saw oscillator using efficient polynomial transition regions (EPTR).
- **Phd :** Implements the resonant filter simulation as shown in http://en.wikipedia.org/wiki/Phase_distortion_synthesis
- **Phase :** Produces an unipolar aliased saw wave.
- **SawBlit :** Saw oscillator using BLIT synthesis.
- **Sawcore :** Saw oscillator with hard-sync using (PTR W=2).
- **Tricore :** Triangle oscillator with reset/sync.
- **Sine :** Sin oscillator reset control with reset/sync.
- **Noise :** Simple noise generator (with a « pinker » parameter).

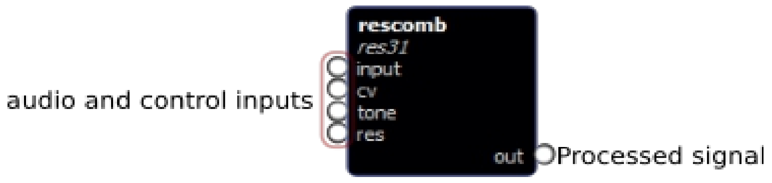
Get oscillators of all kind is pretty important since developers could need a basic aliased oscillator (that will used less CPU) or a high quality sound.

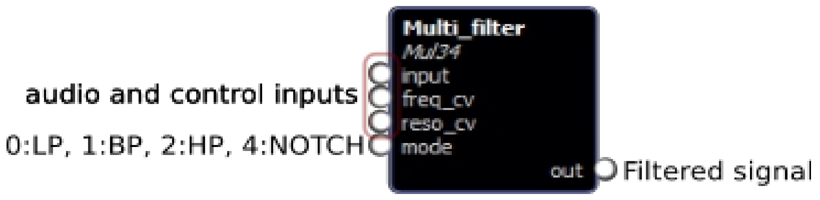

gate input
envelope
parameters
Adsr_env
Ads27
gate
attack
decay
sustain
release
out
0:White, 1:Brown
noise
noi13
color
out
Pitch CV
Reset/sync
sine
sin14
cv
reset
out
gate input
Parameters
Ad_env
Ad_26
gate
attack
decay
out
rate (0 to 10)
reset
clock
clo15
rate
reset
out
Pitch CV
saw_eptr
saw17
cv
out
Rate CV
Waveform
Reset/sync
lfo
lfo24
cv
shape
reset
out
Pitch CV
Pulse width
blit
bli23
cv
pw
wave
out
Pitch CV
Reset/sync
phd
phd19
cv
reset
out
Pitch CV
saw_ptr2
saw12
cv
out
Pitch CV
Reset/sync
Stop oscillator
tricore
tri25
cv
reset
disable
out
Pitch CV
saw_ptr1
saw16
cv
out
Pitch CV
Reset/sync
minblep
min22
cv
reset
out
Pitch CV
Reset/sync
phase
pha20
cv
reset
out
Pitch CV
Pulse width
blit
bli23
cv
pw
wave
out
Pitch CV
saw_ptr2
saw12
cv
out
Pitch CV
saw_blit
saw18
cv
out
Pitch CV
Reset/sync
sawcore
saw21
cv
reset
out

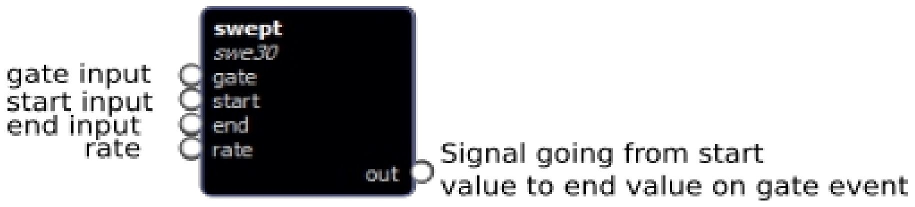
Processors :

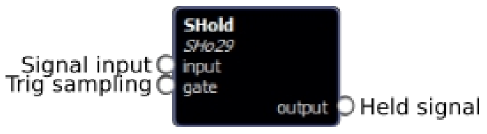
Processors and blocks relative to DSP work with value between -1 and 1 for audio and generally between 0 and 1 for control inputs.
This category contain signal processors such as filters, delay,... and usefull tools for effects making such as an envelope follower, a sample and hold,...

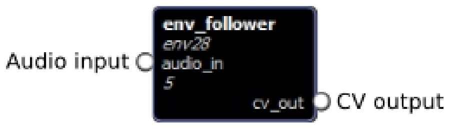


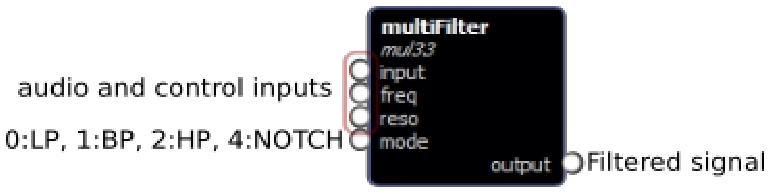


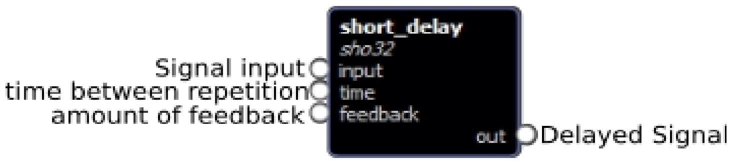












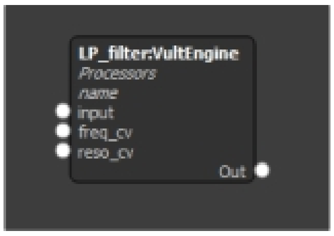
6. Create a block

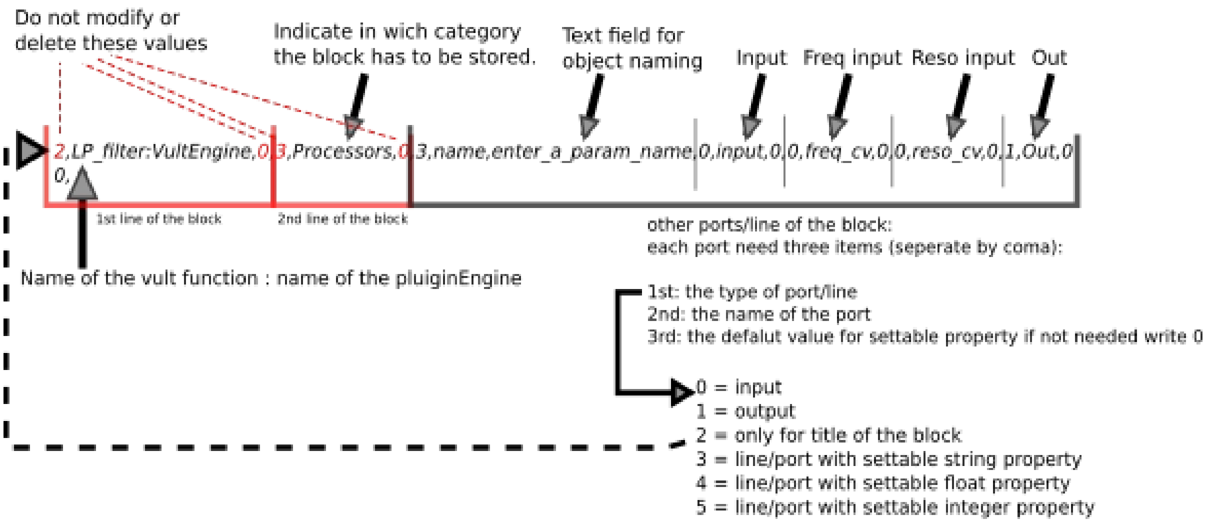
Create blocks from vult generated code :

New blocks can be created by using the vult transcompiler.
 I won't explain how to use vult, if you need to learn how it works go to
https://modlfo.github.io/vult/
Assuming you are able to create your vult codes for VCV module you need to :
– Create a blockModel and add it into the BlockModel folder
– Give to your vult codes/engine an name than other VultEngines aren't
using yet.

How to create a block model (can be done with the block creator):

A blockmodel is a text file that is used by the editor to defined a block, from
this he know the category of the block and add it to the good one in the « add
block » menu, create the inputs, ouputs, names and properties dialog boxes.
You nedd to add it ion the BlockModel Folder. Here is an example working for a vult
object that is part of vultEngine such as *fun LP_filter (input, freq, reso, out) :*




Do not modify or delete these values
Indicate in wich category the block has to be stored.
Text field for object naming
Input
Freq input
Reso input
Out
1st line of the block
2nd line of the block
Name of the vult function : name of the pluiginEngine
other ports/line of the block:
each port need three items (seperate by coma):
1st: the type of port/line
2nd: the name of the port
3rd: the defalut value for settable property if not needed write 0
0 = input
1 = output
2 = only for title of the block
3 = line/port with settable string property
4 = line/port with settable float property
5 = line/port with settable integer property

```
2,LP_filter:VultEngine,0,3,Processors,0,3,name,enter_a_param_name,0,input,0,0,freq_cv,0,0,reso_cv,0,1,Out,0
0,
```

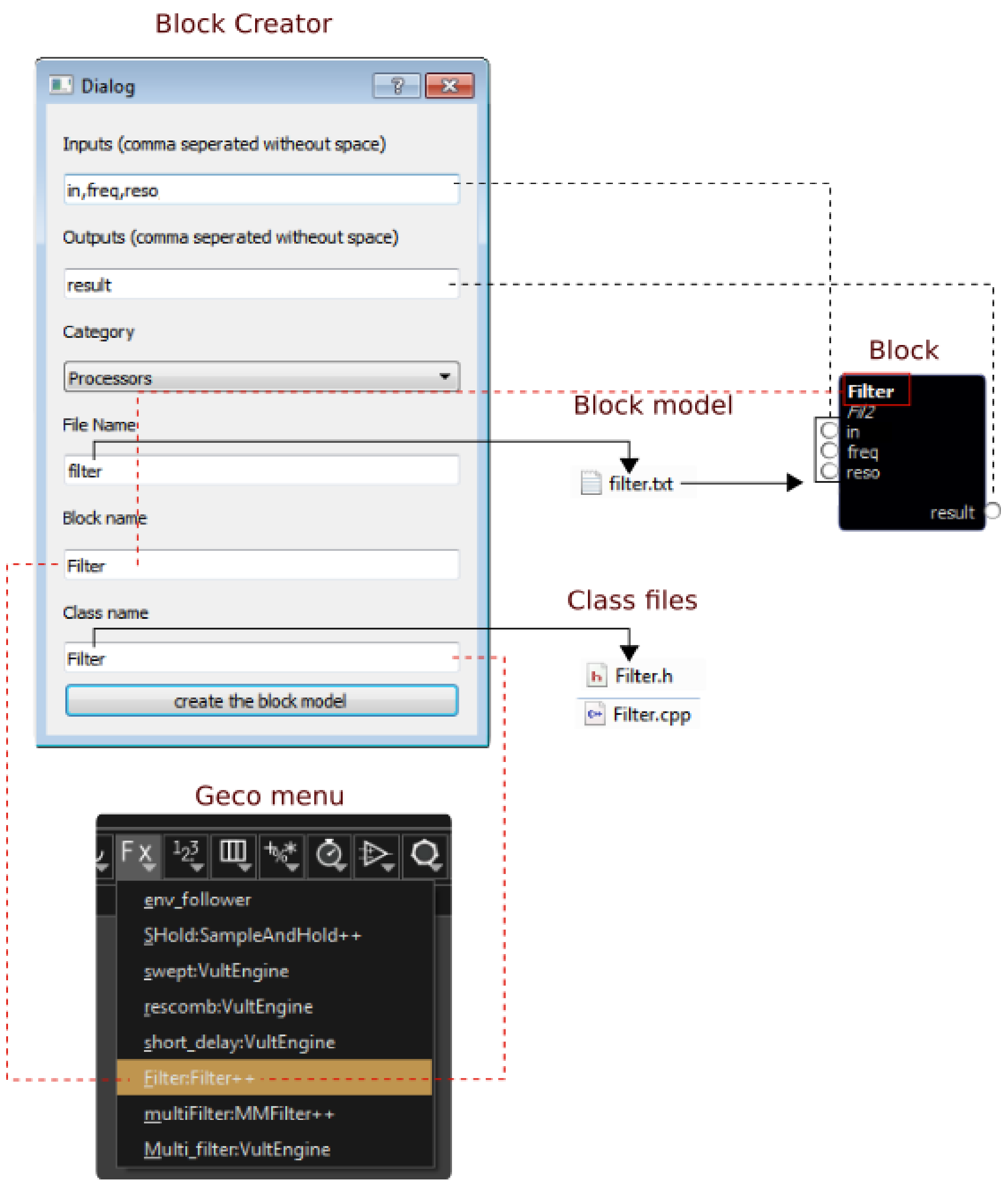
*Note: Vult blocks only allow to add inputs and ouputs, no parameters settable in the property panel.
This seems sufficient since vult accept float and integer arguments and Geco offers everething needed to set these within the nodes editor.*

Create blocks from regular C++ classes :

You can find a blockModel creator on , this allow to create the block file and cpp and h canva files of the concerning class.

The generated cpp and h file create the method link to the input and output in a way that make it works with Geco.

Fig1. (block creator - filter example)

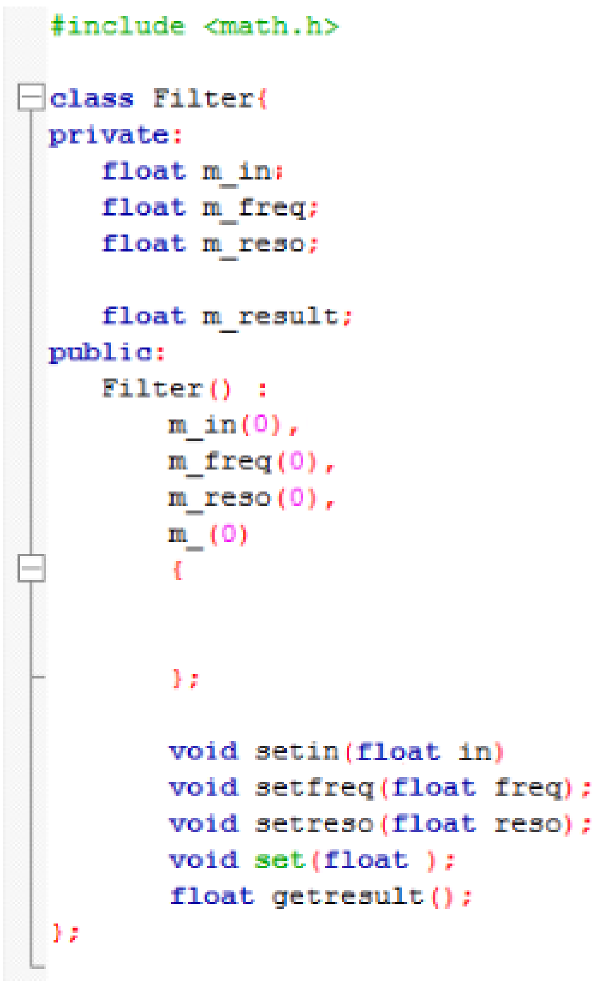


To add the created block, you just need to :

– Add the generated cpp and h file of the related class and of course write your code in it.
– Add the generated blockModel (the .txt file) in the BlockModels folder of geco.

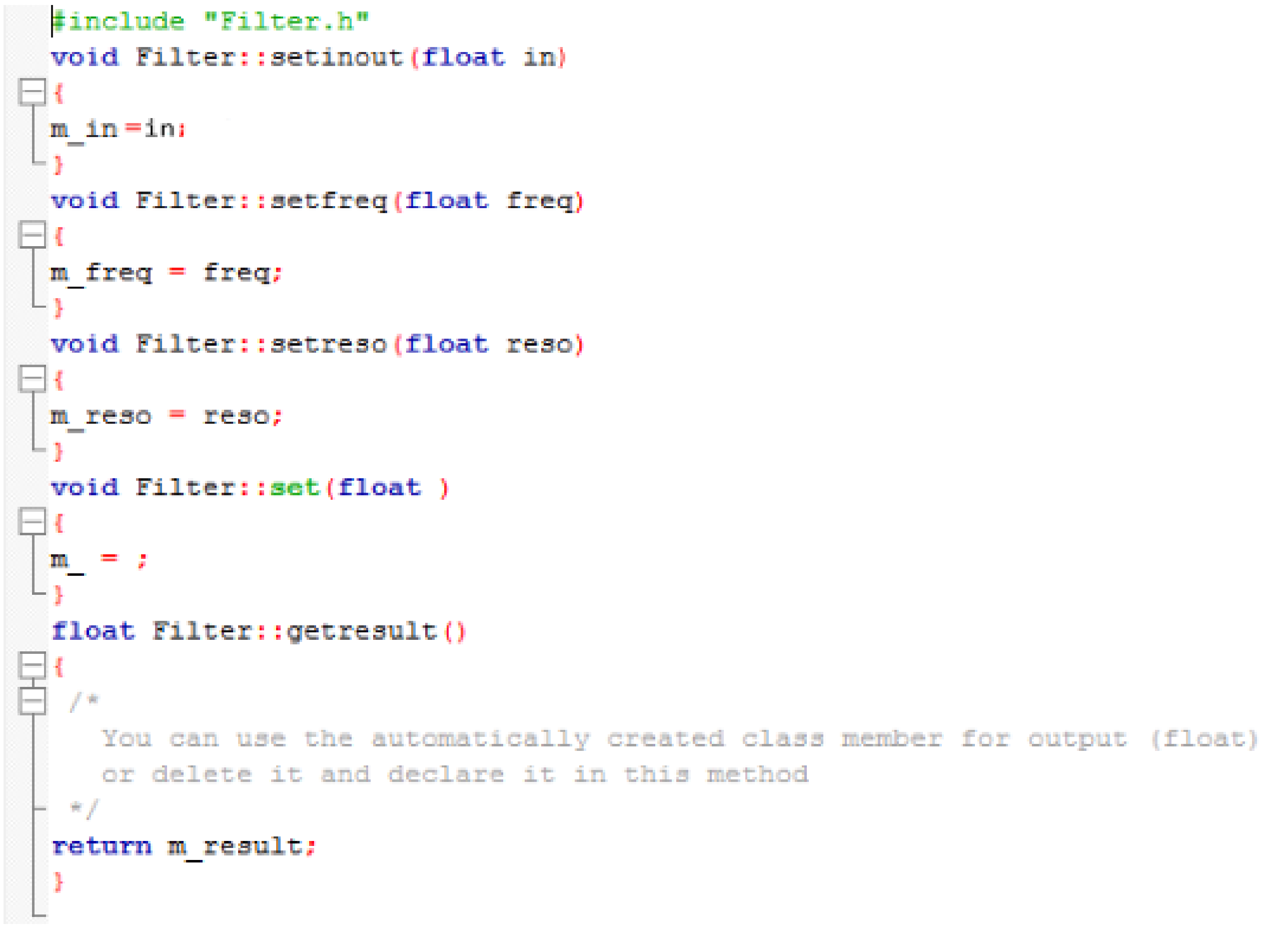
Here are the automatically generated files for the same block than in fig1. (filter example).

Fig2. (header File – filter example)



```
#include <math.h>

class Filter{
private:
    float m_in;
    float m_freq;
    float m_reso;

    float m_result;
public:
    Filter() :
        m_in(0),
        m_freq(0),
        m_reso(0),
        m_(0)
        {


        };

        void setin(float in)
        void setfreq(float freq);
        void setreso(float reso);
        void set(float );
        float getresult();
};
```

Fig2. (CPP File – filter example)



```
#include "Filter.h"
void Filter::setinout(float in)
{
m_in=in;
}
void Filter::setfreq(float freq)
{
m_freq = freq;
}
void Filter::setreso(float reso)
{
m_reso = reso;
}
void Filter::set(float )
{
m_ = ;
}
float Filter::getresult()
{
 /*
   You can use the automatically created class member for output (float)
   or delete it and declare it in this method
 */
return m_result;
}
```

This said, if needed or for information this is how the classes integration work :

You have to use setNameOfTheInput() function to get the value send to this input and use getNameOfTheOutput() function to return a value to an ouput. Once the class is writed you ll nedd to write the corresponding block model that will be use by geco to create the final block. The blockmodels are text file that define how the add the block. As you can on figure 1 the parameters of the block are define by coma seperated values. The first define what kind of parameter is defined :

0 = input
1 = output
2 = only for title of the block
3 = line/port with settable string property
4 = line/port with settable float property
5 = line/port with settable integer property

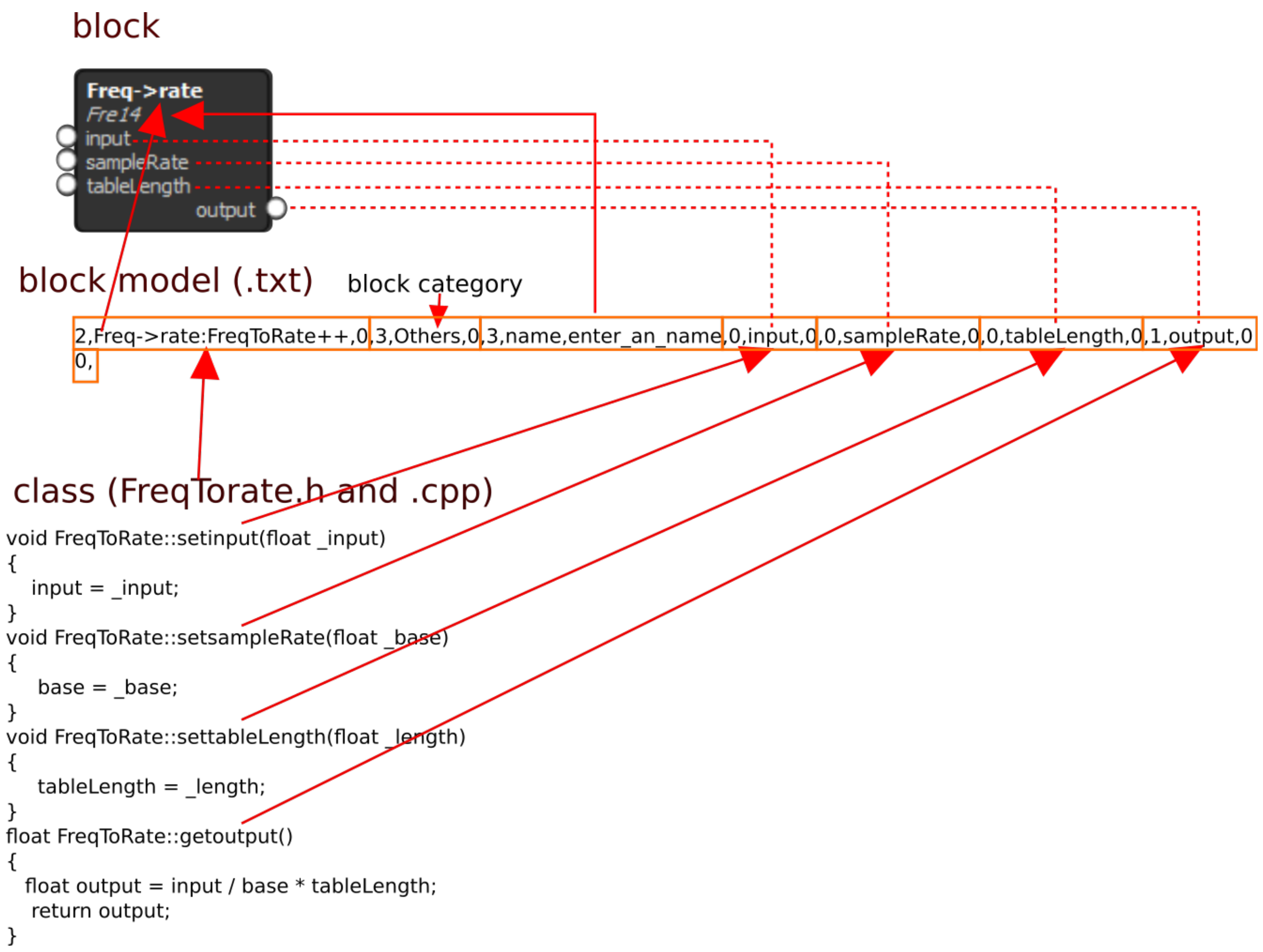
The second is the name of the parameter and the third is the default value of this.
The first parameter is alaways the title of the block (or blockname) and then the first value is 2, the second is the block name and the third is always set to 0.
The soncd value for block made from c++ classes is built like this :
blockname:classname++ (in figure 1 : freq->Rate :FreqToRate++).

The second paremeters is the category of the block . Then the first value is 3, the second is the category of the block and the third is always set to 0.

The thirdd paremeters is the created object name (fre14 in figure1). Then the first value is 3, the second is the name of the parameter that will be display in the label upon the corresponding text field in the parameter panel, the third (the default value) can be anything since the object name will beautomatically set by Geco.
The next are the inputs and output. The first value is 0 for the inputs and 1 for the ouputs, the second value is the name of the input/outpout that will be displayed on the block. The third value is always set to 0.

Here is an example for the « Freq->rate » Block :

figure 1.



```
2,Freq->rate:FreqToRate++,0,3,Others,0,3,name,enter_an_name,0,input,0,0,sampleRate,0,0,tableLength,0,1,output,0
0,
```

```
void FreqToRate::setinput(float _input)
{
    input = _input;
}
void FreqToRate::setsampleRate(float _base)
{
     base = _base;
}
void FreqToRate::settableLength(float _length)
{
     tableLength = _length;
}
float FreqToRate::getoutput()
{
   float output = input / base * tableLength;
    return output;
}
```