

How to make your own block in Gnu radio

Compiled by : Maunil Vyas

Before I start,

Let me ask you a simple question would you want to contribute to any open source project as a programmer? or Are you interested in signal processing stuff? but you find Matlab, Labview requires Licence then you are on right track to read this and I hope that I will be a part of your learning.

Pre-requisites: Knowledge of Python is required, Gnu Radio installed on Linux, Currently I have prepared this tutorial by experimenting each and everything on Ubuntu 14.04.(Gnu radio is also supported in MAC and Windows but I am not sure that they follow the same fashion as in Linux or different)

Abstract

Here I am going to explain you about, How one can build his/her own block set with python in Gnu radio open source software, mainly I could not able to cover each and everything but still I have tried my best to put as much as I can in this short tutorial but I do not guarantee that by reading only this one can build anything in gnu radio because I believe things required time, patience and hard work. This tutorial is basically an abstract overview of the whole process which is required to create a new signal processing block in gnu radio, my ultimate goal behind this tutorial is to just make you feel comfortable with Gnu radio block set procedure and want to help the society by spreading a tiny amount of knowledge.

I am going to address a mean computation block set in gnu radio which is capable of computing the mean of given vector.

Best luck. Let's kick start

❖ What is Gnu radio, do you have any idea?

If yes then it's great, if not then it's fine just read the below lines and you will get an abstract idea about what exactly it is.

Gnu radio is a free open source software development toolkit. which provides ample amount of signal processing libraries and block sets to test and implement band-limited communication standards. It has highly flow graph related interface which itself is an advantage for the user.

Applications Supported: Audio processing, Mobile communication, tracking satellites, radar systems, GSM networks, Digital Radios and much more. All are in computer software.

I hope you get abstract about what Gnu radio is all about if you are still confused or want further details then look at [1] and [2].

Great! Let's see another FAQ

❖ Why would I want Gnu radio?

The answer is simple **cost affordability, quality of performance and less complexity.**

Let see a specific radio communication case, suppose I want to implement a specific radio communication tool for certain application, so I need to implement a special hardware (We call it IC in VLSI) after finishing the designing part of IC I come to know that my application requires some changes so now I have to redesign the IC and need to redo the same process again. As one can clearly point out that to redo the same process again would have been a time-consuming job, not only that it certainly deteriorate the cost optimality constraint as well and eventually we end up having the worst outcome.

One may ask by curiosity, **Can I do testing and implementation on a universal device which supports majorly all my suitable applications, Can I change parameters in the middle of my implementation without bothering about the cost and complexity?**

The Answer for the above claim is **yes** with the help of **software-defined radio technology** you can do it. Software defined radios are basically hardware setups which are capable of communicating with computer software. It's like all signals have been passed through computer software and that powerful software is playing a role of black box which is capable of giving us desirable outputs based on given inputs. so here in software-defined technology, the major advantage we have is we can always replace software(Codes written for a specific signal process) without much bothering about above- mentioned hazards. Gnu radio is one of that powerful open source software. There are other tools like Matlab and Lab view are also available but as I mentioned earlier that they required a licence. Gnu radio is an open source software which itself is a major advantage, open source means one can use other people's work, one may contribute to the society.

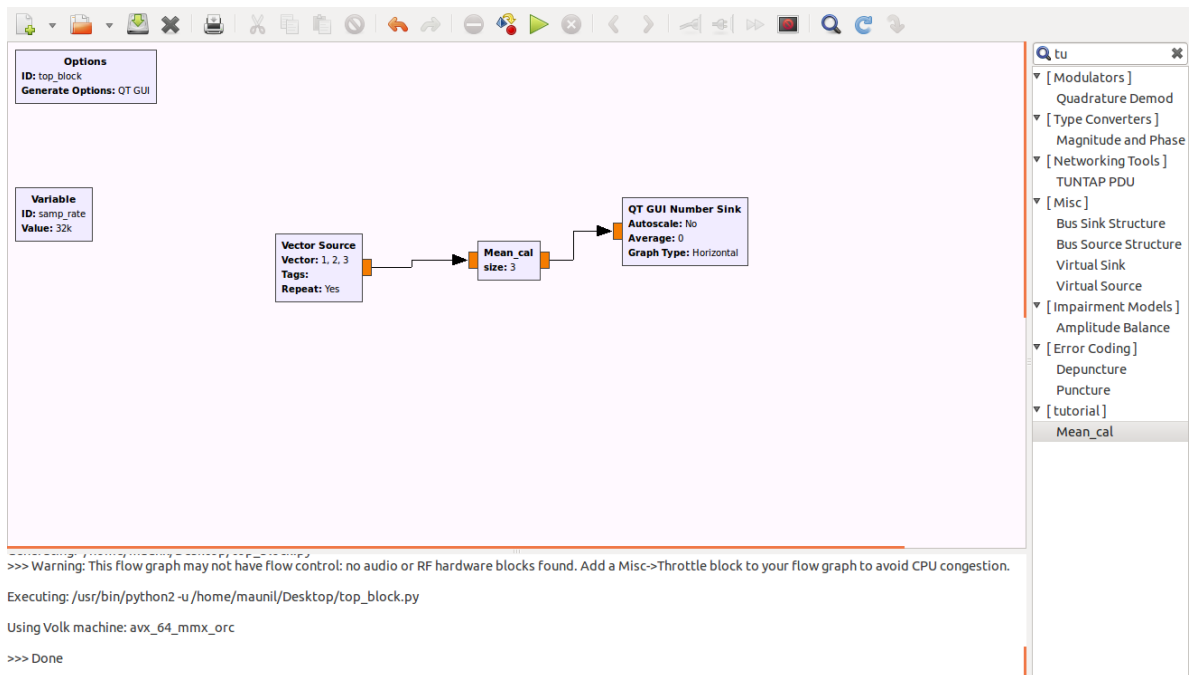
Enough motivation now let's focus on our main aim to design our own block set in gnu radio. here I am using the **out-of-tree module** to create modules for block set

FAQ

❖ What is an out-of-tree module?

It's basically gnu radio component that does not live within the gnu radio source tree.so if you want your own module for your own block set then this will help us to customise the gnu radio with your own blocks. To implement a block you required to implement a module which is a cumbersome task as one has to think of boilerplate code, makefile editing, etc. **gr_modtool** is a script which aims to help with all these things by automatically editing makefiles, using templates, and doing as much work as possible for the developer such that you can jump straight into the DSP coding. It's my suggestion to start with **gr_modtool** but if one wants full customization then one has to write each and everything on his/her own.

This is what the final outcome of this tutorial.



As I mentioned above that I am using readymade templet **gr_modtool** to make our job easy. first, we need to create a module where we want to put our block set

Module Creation

Following commands will be used to create module

```
cd ~
```

```
gr_modtool newmod tutorial
```

```
maunil@maunil-ThinkPad-L430: ~  
maunil@maunil-ThinkPad-L430:~$ gr_modtool newmod tutorial  
Creating out-of-tree module in ./gr-tutorial... Done.  
Use 'gr_modtool add' to add a new block to this currently empty module.  
maunil@maunil-ThinkPad-L430:~$
```

Here tutorial is a name of module. one can select anything unless it does not conflict with the currently installed modules.

To check whether the module has been created by system or not

Following commands will be used to check

cd ~

cd gr-tutorial

ls

```
maunil@maunil-ThinkPad-L430: ~/gr-tutorial
maunil@maunil-ThinkPad-L430:~$ cd gr-tutorial
maunil@maunil-ThinkPad-L430:~/gr-tutorial$ ls
apps  CMakeLists.txt  examples  include  MANIFEST.md  swig
cmake  docs            grc       lib      python
```

Here we can see that the module has been already created the system.

Block Creation

Now it's time to create a simple general block set using python and then we customise it.

Following commands will be used

cd gr-tutorial

- Make sure you are in this directory

gr_modtool add -t general -l python

```
maunil@maunil-ThinkPad-L430: ~/gr-tutorial
maunil@maunil-ThinkPad-L430:~/gr-tutorial$ gr_modtool add -t general -l python
GNU Radio module name identified: tutorial
Language: Python
Enter name of block/code (without module name prefix): Mean_cal
Block/code identifier: Mean_cal
Enter valid argument list, including default arguments: size
Add Python QA code? [Y/n] n
Adding file 'python/Mean_cal.py'...
Adding file 'grc/tutorial_Mean_cal.xml'...
Editing grc/CMakeLists.txt...
maunil@maunil-ThinkPad-L430:~/gr-tutorial$
```

Let's break the above procedure into parts and then we will understand each and everything one by one

gr_modtool add -t general -l python

add means I want to add something **-t** means the type of block which one wants to add there are types like sync, general etc. we have used **general** here. Why general and why not other, honestly speaking sync would be also fine considering our current application focus but let's not bother about those things right now you can explore those your own once you have a basic idea about everything **-l python** indicates the programming language for a block set.

Mean_cal is the name of our block set, **size** is a parameter for vector size. It's not compulsory to give only one parameter. one can give more than one parameters as per their requirements I have selected **python QA code [Y/n] = n**. because it's basically asking me for the testing code which we generally write in python to check whether our block set works perfectly or not as we are working on small scale problem of finding mean I don't think we require it right now so I have put there as **n** means no,

I encourage you to think when it is required and how to customise it based on the current gnu radio knowledge you have.

🌈 Block Customization

Now let's customise the python code.

Where to customise?

Following commands will be used

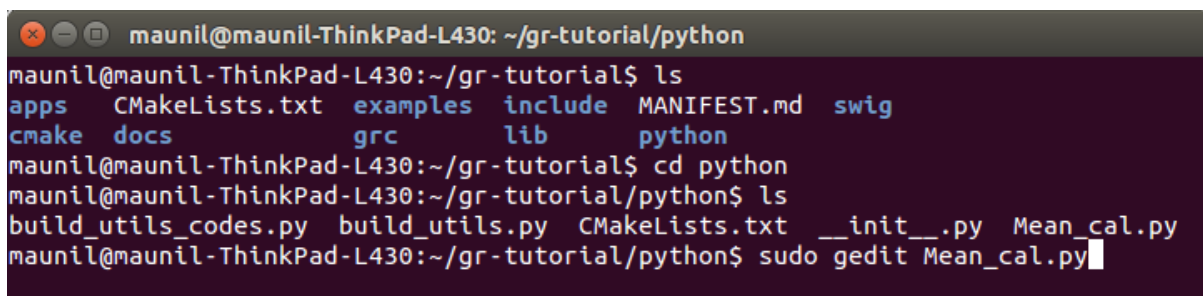
cd gr-tutorial (make sure you are in this path)

ls

cd python

ls

sudo gedit Mean_cal.py



```
maunil@maunil-ThinkPad-L430: ~/gr-tutorial/python
maunil@maunil-ThinkPad-L430:~/gr-tutorial$ ls
apps  CMakeLists.txt  examples  include  MANIFEST.md  swig
cmake  docs            grc       lib       python
maunil@maunil-ThinkPad-L430:~/gr-tutorial$ cd python
maunil@maunil-ThinkPad-L430:~/gr-tutorial/python$ ls
build_utils_codes.py  build_utils.py  CMakeLists.txt  __init__.py  Mean_cal.py
maunil@maunil-ThinkPad-L430:~/gr-tutorial/python$ sudo gedit Mean_cal.py
```

As you open a file in gedit you will find three functions in a python code

Constructor

```
"""
docstring for block Mean_cal
"""
def __init__(self, size):
    self.size = size; #Size indicate the vector size , recall we have add this while generating the Mean_cal block
    gr.basic_block.__init__(self, #Input and Output can be anytype as per your requirements here we have taken float32
        name="Mean_cal",
        in_sig=[numpy.float32],
        out_sig=[numpy.float32])
```

As you can see I have changed the constructor as per my requirements, I have mentioned sufficient comments too, so that you can understand what is going on.

Forecast function, please read the comments to understand it's vitality

```
def forecast(self, noutput_items, ninput_items_required):    #This function is not useful right now but it is useful when you have more
    #setup size of input_items[i] for work call              number of inputs which lead to output, no change is required
    for i in range(len(ninput_items_required)):
        ninput_items_required[i] = noutput_items
```

General_work is the main function where magic happens, It's a space where programmers write their algorithms

```
def general_work(self, input_items, output_items):          #This is the main function where we do signal processing algo implementation

    #We want to find a mean so
    sum = 0;         #For addition
    for i in range (self.size):
        sum = sum+input_items[0][i]          #Here input_items[0] is for first vector and input_items[0][i] is for the
    elements of that particular vecotor, pointer array kind of thing

    sum = sum/self.size;    #Avg

    output_items[0][:] = sum          #Provide value to the output vector here it's going to be single element of single vector that's
    why output_items[0][:]

    #consume(0, len(input_items[0]))        #Put it in a comment
    self.consume_each(len(input_items[0]))  #Open comment here
    return len(output_items[0])
```

After customising the python code now it's time for .xml files. I have to make sure that everything will be in a properly synchronise manner.

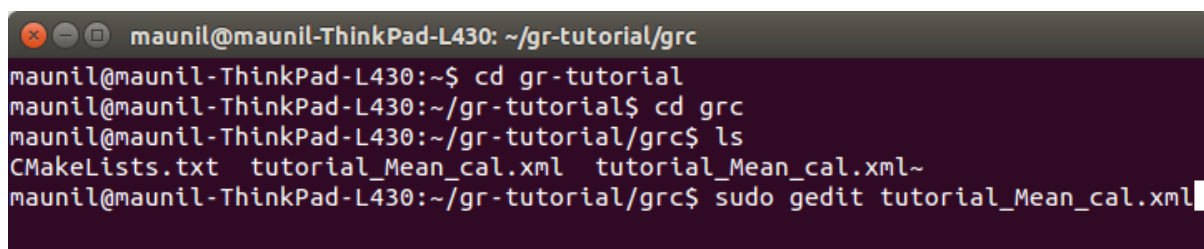
To open a .xml file for our interest on has to use following commands

cd gr-tutorial (make sure you are in this path)

cd grc

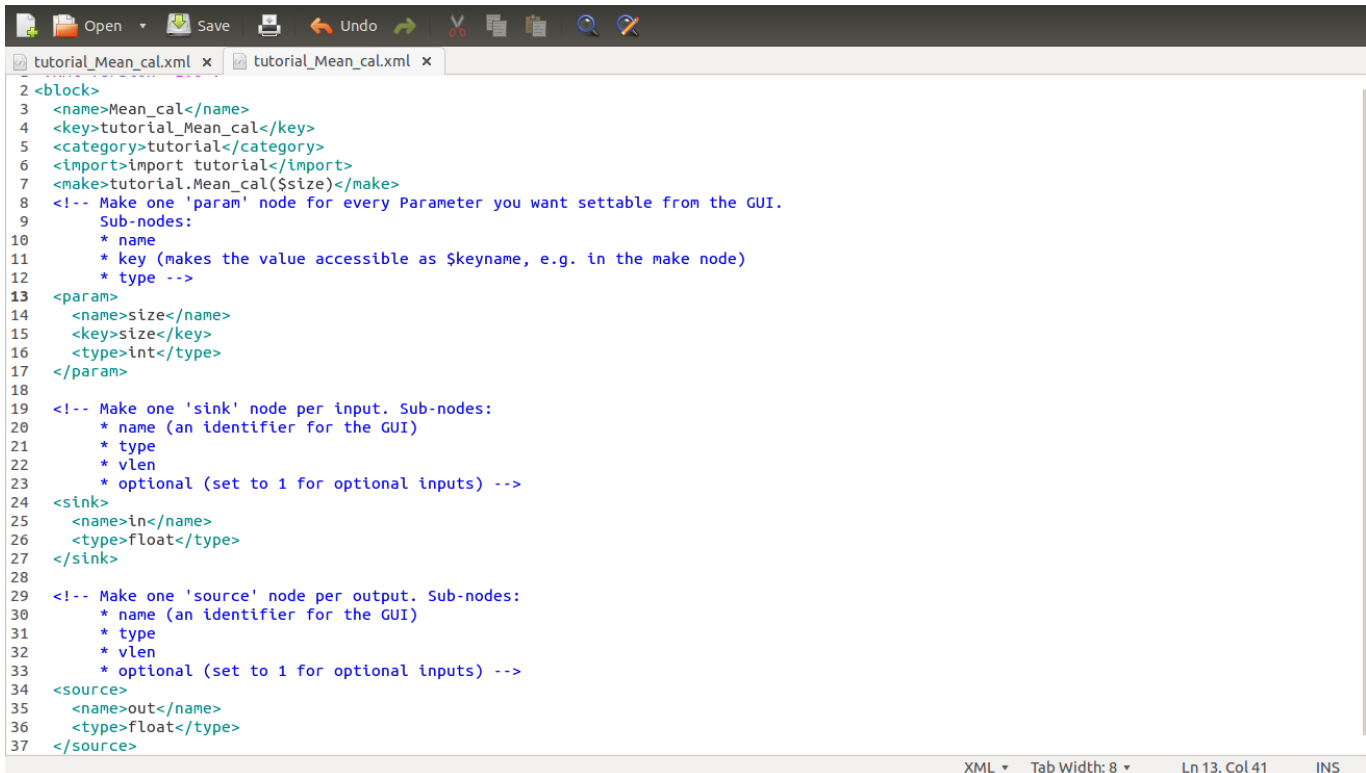
ls

sudo gedit tutorial_Mean_cal.xml



```
maunil@maunil-ThinkPad-L430: ~/gr-tutorial/grc
maunil@maunil-ThinkPad-L430:~$ cd gr-tutorial
maunil@maunil-ThinkPad-L430:~/gr-tutorial$ cd grc
maunil@maunil-ThinkPad-L430:~/gr-tutorial/grc$ ls
CMakeLists.txt  tutorial_Mean_cal.xml  tutorial_Mean_cal.xml~
maunil@maunil-ThinkPad-L430:~/gr-tutorial/grc$ sudo gedit tutorial_Mean_cal.xml
```

Modification in .xml file



```
2 <block>
3   <name>Mean_cal</name>
4   <key>tutorial_Mean_cal</key>
5   <category>tutorial</category>
6   <import>import tutorial</import>
7   <make>tutorial.Mean_cal($size)</make>
8   <!-- Make one 'param' node for every Parameter you want settable from the GUI.
9       Sub-nodes:
10          * name
11          * key (makes the value accessible as $keyname, e.g. in the make node)
12          * type -->
13   <param>
14     <name>size</name>
15     <key>size</key>
16     <type>int</type>
17   </param>
18
19   <!-- Make one 'sink' node per input. Sub-nodes:
20       * name (an identifier for the GUI)
21       * type
22       * vlen
23       * optional (set to 1 for optional inputs) -->
24   <sink>
25     <name>in</name>
26     <type>float</type>
27   </sink>
28
29   <!-- Make one 'source' node per output. Sub-nodes:
30       * name (an identifier for the GUI)
31       * type
32       * vlen
33       * optional (set to 1 for optional inputs) -->
34   <source>
35     <name>out</name>
36     <type>float</type>
37   </source>
```

As you can see that I have customised the type of input, output and parameter as per my requirements.

We are almost done now its time for installation.

Block & Module Installation

Use following commands

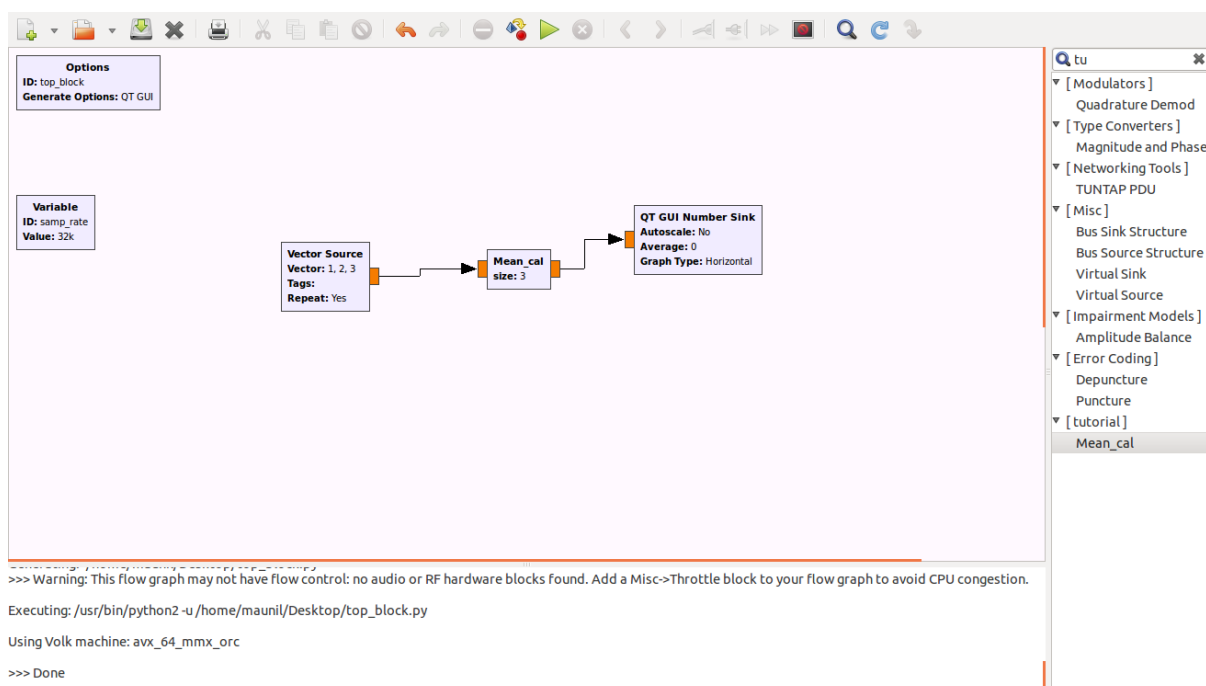
```
cd gr-tutorial          (make sure you are in this path)
mkdir build              (To build a directory name build)
cd build
cmake ../
make
sudo make install
sudo ldconfig
```

```
maunil@maunil-ThinkPad-L430: ~/gr-tutorial/build
-- found gnuradio-runtime, version 3.7.9.2
* INCLUDES=/usr/local/include
* LIBS=/usr/local/lib/libgnuradio-runtime.so;/usr/local/lib/libgnuradio-pmt.so
-- Found GNURADIO_RUNTIME: /usr/local/lib/libgnuradio-runtime.so;/usr/local/lib/libgnuradio-pmt.so
GNURADIO_RUNTIME_FOUND = TRUE
-- No C++ sources... skipping lib/
-- No C++ sources... skipping swig/
-- Found PythonInterp: /usr/bin/python2 (found suitable version "2.7.6", minimum required is "2")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/maunil/gr-tutorial/build
maunil@maunil-ThinkPad-L430:~/gr-tutorial/build$ make
Scanning dependencies of target pygen_python_2cf72
[ 33%] Generating __init__.pyc, Mean_cal.pyc
[ 66%] Generating __init__.pyo, Mean_cal.pyo
[ 66%] Built target pygen_python_2cf72
Scanning dependencies of target pygen_apps_9a6dd
[ 66%] Built target pygen_apps_9a6dd
Scanning dependencies of target doxygen_target
[100%] Generating documentation with doxygen
[100%] Built target doxygen_target
maunil@maunil-ThinkPad-L430:~/gr-tutorial/build$
```

The above screenshot is after writing **make** command and if you find 100% means there is no error in your steps.

We are done, hurray!

Open GRC and check your block



Output

```
Data 0 2.000000
```

🔧 An alternative Approach:

It's not compulsory to know python to implement block set in Gnu radio. You can do the same with the help of c++ as well only thing changes here is the coding part instead of customising python code you need to customise c++ code. If you are interested and want to a detailed knowledge then read [3].

Reference

1. https://en.wikipedia.org/wiki/GNU_Radio
 2. <http://gnuradio.org/redmine/projects/gnuradio>
 3. <http://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules>
-