Beaver's Guide on how to arrange

<u>GEGL Nodes in a C file</u>

GEGL C files have five distinct areas the user needs to remember.

1. The Properties: The list of the filters options that appear in GUI. The properties can change the filters parameters, hide parameters, and change the name and help description for each one of a filters editable options.

2. Potential Node List and filter defining area: List of all potential GEGL nodes and the ability to define all potential filters that will be used in a GEGL Graph.

3. Action defining area: Area where parts of existing GEGL filters are chained in different combinations. They are then specifically defined on what to do.

4. The GEGL Graph: A list of nodes that correspond to a literal GEGL Graph; like the one in Gimp. The graph is also a chain of different combinations but with no defining or only calling their parts.

5. About filter: Set the filter title, description, hash tag and the ability to make it hidden or not.

PLEASE NOTE!

There are small other areas in the C Files that may also need changing. The most notable is *#define GEGL_OP_NAME* and *#define GEGL_OP_C_SOURCE* that are critical to making GEGL filters. They require matching their name with the name of the C file.

This part of the guide will **not** teach the user about making GEGL Graph syntax. You can learn that on GEGL.org and GEGL Graph threads on Gimp Chat. We are just going to go over the important parts of the C file that one needs to know in order to make basic third party GEGL Filters.

--

2. (potential nodes/filter defining) is where I head after naming and describing the third party filter in **5.** . In here I am defining the full GEGL operations I want to use. I can also give them different names then the raw GEGL operation. I can even list nodes that may or may not or ever be used in the final filter if I want.

Potential nodes.

```
GeglNode *gegl = operation->node;
GeglNode *input, *hue, *gaussian, *rotate, *glow, *move, *fill, *behind, *output;


input    = gegl_node_get_input_proxy (gegl, "input");
output   = gegl_node_get_output_proxy (gegl, "output");
```

Defined Names   RAW GEGL INFO

```
gaussian    = gegl_node_new_child (gegl,
                         "operation", "gegl:gaussian-blur",
                         NULL);

hue      = gegl_node_new_child (gegl,
                         "operation", "gegl:hue-chroma",
                         NULL);

rotate      = gegl_node_new_child (gegl,
                         "operation", "gegl:rotate-on-center",
                         NULL);

glow     = gegl_node_new_child (gegl,
                         "operation", "gegl:bloom",
                         NULL);

move     = gegl_node_new_child (gegl,
                         "operation", "gegl:translate",
                         NULL);

behind    = gegl_node_new_child (gegl,
                         "operation", "gegl:dst-over",
                         NULL);

fill     = gegl_node_new_child (gegl,
                         "operation", "gegl:color",
                         NULL);
```

Don't forget the asterisk when chaining things like *blur, *median, *stroke

In later steps is we define the filters. This is in no particular order.

```
gegl_operation_meta_redirect (operation, "hue", hue, "hue");
gegl_operation_meta_redirect (operation, "gaus", gaussian, "std-dev-x");
gegl_operation_meta_redirect (operation, "gaus", gaussian, "std-dev-y");
gegl_operation_meta_redirect (operation, "rotate", rotate, "degrees");
gegl_operation_meta_redirect (operation, "strength", glow, "strength");
gegl_operation_meta_redirect (operation, "x", move, "x");
gegl_operation_meta_redirect (operation, "y", move, "y");
gegl_operation_meta_redirect (operation, "value", fill, "value");
```

The duel 'gaus' is what allows both the x and y St.D of gaussian blur to be controlled one slider.

```
property_double (gaus, _("Gaussian Blur for both XY StD"), 0)
    description (_("Standard deviation for the XY axis"))
    value_range (0.0, 70.0)
```

Property

```
property_double (hue, _("Hue from GEGL Hue Chroma"),  0.0)
    description  (_("Hue adjustment"))
    value_range  (-180.0, 180.0)

property_double (rotate, _("Rotation's Property from GEGL Ripple"), 0.0)
    value_range (-180, 180)
    ui_meta     ("unit", "degree")
    ui_meta     ("direction", "ccw")

property_double (strength, _("Glow from GEGL Bloom"), 0.0)
    description (_("Glow strength"))
    value_range (0.0, G_MAXDOUBLE)
    ui_range    (0.0, 100.0)

property_double (x, _("Move Horizontal from GEGL Translate"), 0.0)
    description (_("Horizontal translation"))
    ui_range (-1000.0, 1000.0)
    ui_meta ("unit", "pixel-distance")
    ui_meta ("axis", "x")

property_double (y, _("Move Vertical from GEGL Translate"), 0.0)
    description (_("Vertical translation"))
    ui_range (-1000.0, 1000.0)
    ui_meta ("unit", "pixel-distance")
    ui_meta ("axis", "y")

property_color (value, _("Color fill from GEGL Color"), "#000000")
    description (_("The color to render (defaults to 'black')"))
```

We list in order the **property** name, **defined filter name** and **raw GEGL command.**
In example *"gaus", gaussian, "std-dev-y");*

Now we have to discuss about how to define actions in **3**. It requires going through C files of existing GEGL filters and copying and pasting their commands.

Examples are

```
property_double (hue, _("Hue from GEGL Hue Chroma"),  0.0)
   description  (_("Hue adjustment"))
   value_range  (-180.0, 180.0)

property_int  (radius, _("Radius"), 3)
  value_range (-400, 400)
  ui_range    (0, 100)
  ui_meta     ("unit", "pixel-distance")
  description (_("Neighborhood radius, a negative value will calculate with
inverted percentiles"))

property_double (scale, _("Effect strength"), 1.0)
    description(_("Strength of the sepia effect"))
    value_range (0.0, 1.0)
```

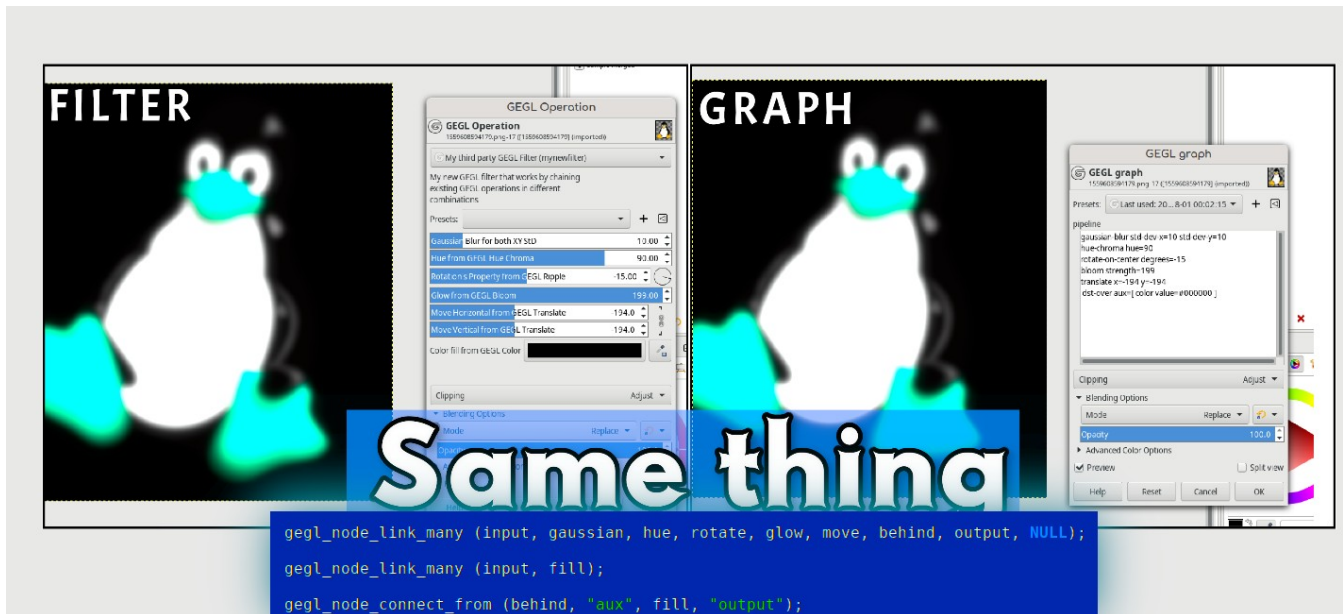Please take not that two properties can **not** have the same first () name

Example of a invalid command

The GEGL Graph expressed in two ways

The Graph for the default third party filter in the GEGL C file is

```
  gegl_node_link_many (input, gaussian, hue, rotate, glow, move, behind, output,
NULL);

  gegl_node_link_many (input, fill);

  gegl_node_connect_from (behind, "aux", fill, "output");
```

and Gimp's GEGL Graph filter

```
gaussian-blur std-dev-x=0 std-dev-y=0
hue-chroma hue=0
rotate-on-center degrees=0
bloom strength=0
translate x=0 y=0
 dst-over aux=[ color value=#000000 ]
```

-

They are both the same thing expressed in different ways. But only the C file will give you a GUI of it.

This is an example of GEGL syntax blend modes (and potentially ID and ref) being used in a C file. We will not be going over ID and Ref in this basic guide. Just the basics on how to add GEGL blend modes in a graph.

In Example the behind blend mode that makes the color fill behind Tux.

```
gegl_node_link_many (input, gaussian, hue, rotate, glow, move, behind, output, NULL);
gegl_node_link_many (input, fill);
gegl_node_connect_from (behind, "aux", fill, "output");
```
The behind blend mode is connecting to the operation gegl:color.

We can remove the blend mode "*behind/dst-over*" by removing this.

```
gegl_node_link_many (input, gaussian, hue, rotate, glow, move, behind, output, NULL);
gegl_node_link (input, fill);
gegl_node_connect_from (behind, "aux", fill, "output");
```
Removing this will remove the behind blend mode and color fill layer.

Custom filters with many blend modes and ref's and id's can be confusing. This guide will not address any further on this. The more blend modes fused with filters your graph has - the more complicated *gegl_node_connect's* will be.

Naming Filters in **5** and the C file name.

Ctrl F search for *GEGL_OP_NAME* and replace the name with both the name of the C file you are editing and the name of the C file with .c on the latter line. Each respectively to this image.

```
#else

#define GEGL_OP_META
#define GEGL_OP_NAME     mycustomfilter
#define GEGL_OP_C_SOURCE mycustomfilter.c

#include "gegl-op.h"
```
The name of the custom filter must be the same as the C files name.

Now we have to name the filters. This can also be considered the first step of all of them. But I decided to list it last.

In **5** list the following

```
    "name",          "gegl:mynewfilter",
    "title",         _("My third party GEGL Filter"),
    "categories",    "ThirdPartyFilters",
    "reference-hash", "331o1gabt4hk10fjf25sabax",
    "description", _("My new GEGL filter that works by chaining
existing GEGL operations in different combinations "
                     ""),
```

Name is the name of the new raw GEGL Operation

Title is the title of the filter that will appear in GEGL Operations

Categories doesn't do anything unless you specify **hidden** for hidden gegl graph operations that only exist to power a more complicated filter. This guide does not discuss this but that is what my ZZ filters do.

Hash I think the hash has to do with Gimp's help manual. I randomize it with keyboard presses each time for good measures but it may be trivial to modify or not modify.

Description the description of the new GEGL Operation that you made.

```
I hope this is enough information for someone to
make a new third party GEGL filter. You do not need
to be a C programmer to do this. Learn GEGL syntax
at GEGL.org . This guide does not cover that.


                    Beaver wishes you the best!
```