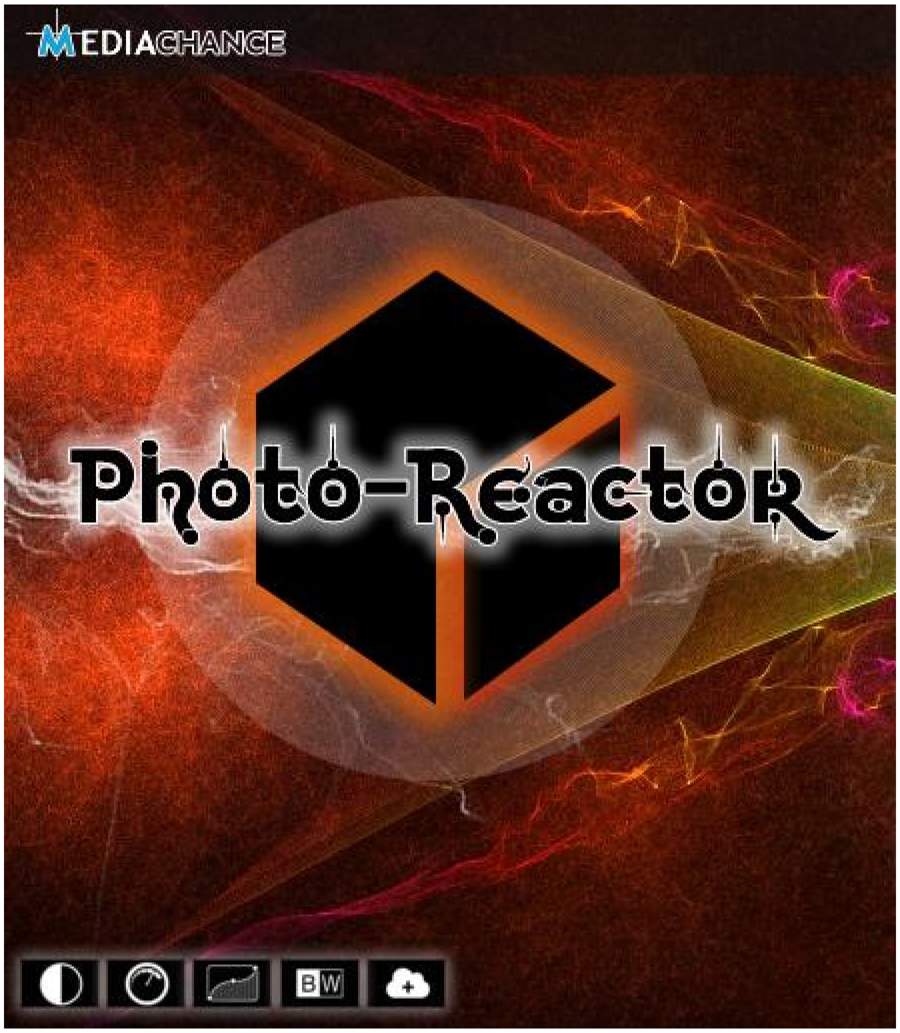


Photo-Reactor

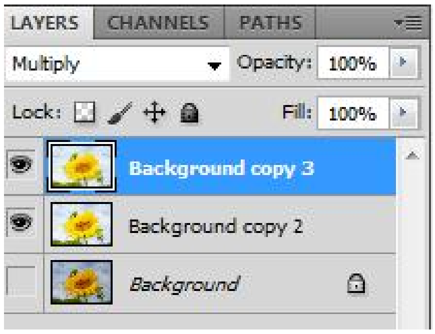
Table of Contents

1 Introduction

I could easily imagine a first-time-user opening Photo-Reactor and then just stare at the screen for a minute, having no idea what is going on. Yes, Photo-Reactor is different but it is not because I had a rough day and wanted to discourage any potential users. The goal is to have editor that is faster and much more flexible than any standard image editor. An editor that is open, expandable and any process is quickly reusable.

Standard image editors
Standard image editors come from a few decades old legacy of linear image editing step-by-step. In simple terms you apply a single adjustment that changes the image data, then you apply another adjustment or effect and so on. If you want to change the very first adjustment you can't, you need to practically start from beginning, and then repeat all the steps, if you remember what they were.
Layers, macros and live effects were introduced later to allow for better structure of projects and to fix the shortcoming of linear editing. But the basic problem remained. Any large multi-layer project becomes a big mystery after a while. Layers will become a black holes for the effects.

Here is an example of a simple "Orton" effect for a dreamy look.

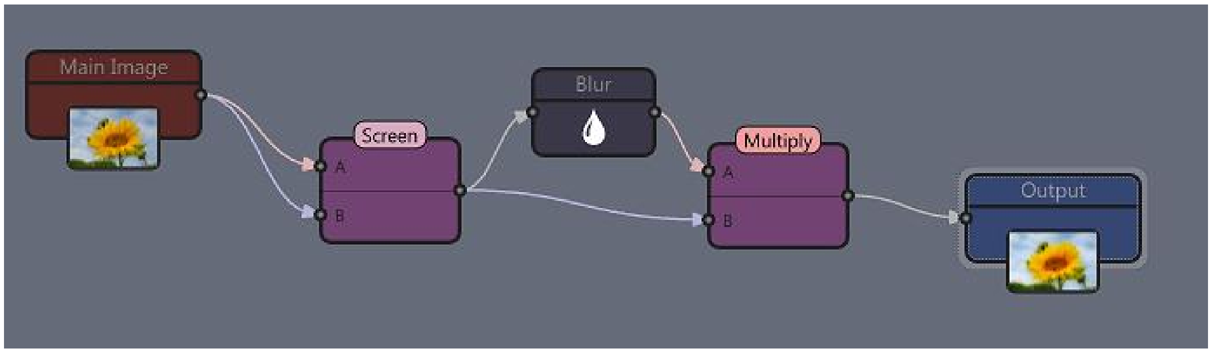


If you never done this effect before then looking at the final image or at the layers will not really give you the steps how to repeat it. How did I make each layer? What effect did I use and in what sequence?

Self-Documenting
This feature (or lack of) is called **self-documenting** and it is missing in any of the standard image editors.
What self-documenting means is that if you do something, others can replicate it by just looking at the final step. A yummy cake is not very self-documented item, you need a recipe to make one. (but you don't need a document to eat it, you can probably follow that part just fine)
Self-documentation also means that the author didn't have to spend any extra effort during making the "item" to document the steps, they become self evident when the item is finished.

If I want to teach you how to create the same dreamy Orton effect for your images, I will have to make extra documentation and write: Make duplicate layer of the image, set the blending to screen, flatten the two layers, make duplicate of the flattened layer and blur it. Set the blending of the top layer to multiply. Done. Quick and easy if you can follow it.
In such case of a simple effect it may work, but imagine if I want to explain how to do a much more complicated effect, like the painting style on the top of this page that was made in the Photo-Reactor and has more than thirty steps.
There is very little chance you will be able to follow it without an error.

In Photo-Reactor making the same Orton effect looks like this:



It not only applies the effect to the image, but it also explains how it is done. And here is another bonus created automatically: it also works as a "macro". I can just change the input image and the Orton effect

will be applied to that one as well.
It doesn't stop there. I can add or remove other effects and adjustments at any step. I can change parameters of everything regardless of when it was applied.

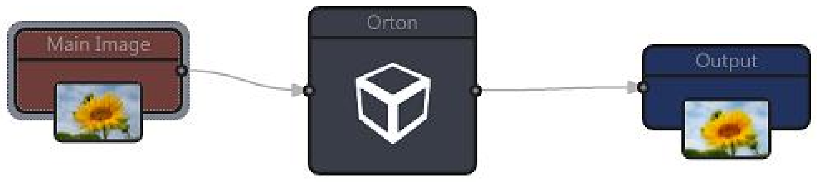
By eliminating the step-by-step linearity of the standard photo editors I removed the time from the equation and make the editing fully dynamic and live. It is no longer important in what sequence I apply the effects as I can always reorder them, delete or add new steps.

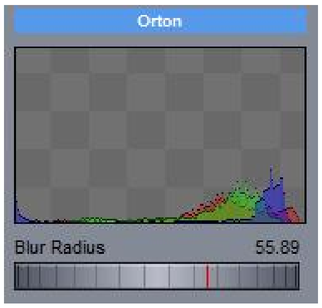
And that, my young Padawan, the essence of Photo-Reactor is.

1.1 Conquer the Spaghetti Monster

But doesn't nodal editing produce a spaghetti code with each increased level of complexity?
That is a good point and (as many engineers would happily agree) a real danger but there is another great feature of Photo-Reactor that comes to the rescue: Encapsulation.

Any series of steps can be simplified into various single items such as groups, devices or virtual effects. They all vary in different degrees of editability. In our case of Orton effect there is very little reason to keep all the objects visible all the time. There isn't much to do with them. The only need for the user would be to adjust the Blur.
We can simply turn the Orton effect into what I call a virtual effect box, with a single visible parameter - the Blur and from now use it as a single effect.





Features at a glance

Experiment
Everything in the Photo-Reactor is geared towards hands-on "discovery" approach. Some objects may be obvious what they do from their name, some may not be and others may be pure experimental, but you can simply connect them and see what they do. There are no different types of data for different objects, everything can be connected together.

Grows with your requirements
No feature can be too complex for Photo-Reactor! However the modular approach allows the user to fully choose his/hers own path of complexity.
The basics are pretty straightforward and you simply use what you need and ignore the rest but as you progress you will discover the full depth of the software.

Share
The goal of Photo-Reactor is to be modular and everything you do can be shared with others.

Scripting for the geeky types
There is a full object oriented C++ scripting language build in. You can actually write and compile a very complex code inside the reactor. But don't worry, it is optional!

SDK with UI aid
The SDK for creating plugins is extremely easy and the application itself has source code generator to help you started.

2 Basics

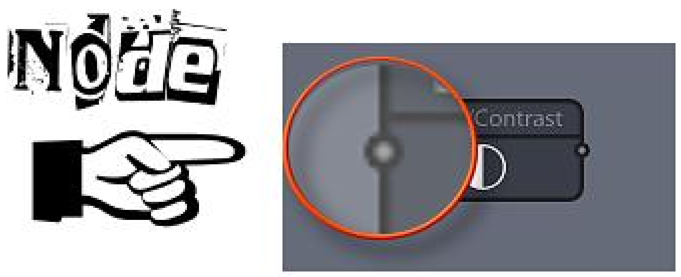
The basics are pretty simple. You find the desired effect in the Image Processing or Building Blocks list and drag it to the workspace, then connecting it between input and output or between other objects.

But first one important point:
The reactor really works only when a Main Image is loaded.

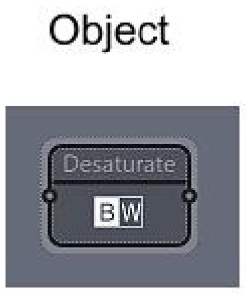
You can have many images in the reactor process (which I call flow), but one image is the most important - **Main Image**. The Main image is like a conductor of an orchestra, it unifies all the little pieces together by setting the tempo. (in this case it is the image size)

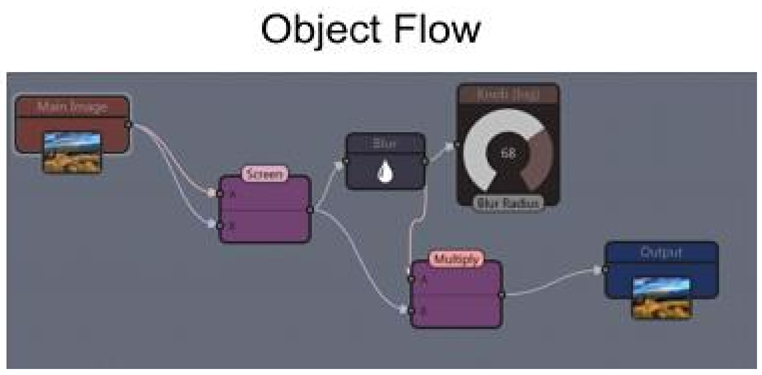
Lets agree on some terms so we don't have to use "connect that wobbly looking thing to a square looking thing".

What is the **node**?
Node is a connecting point.

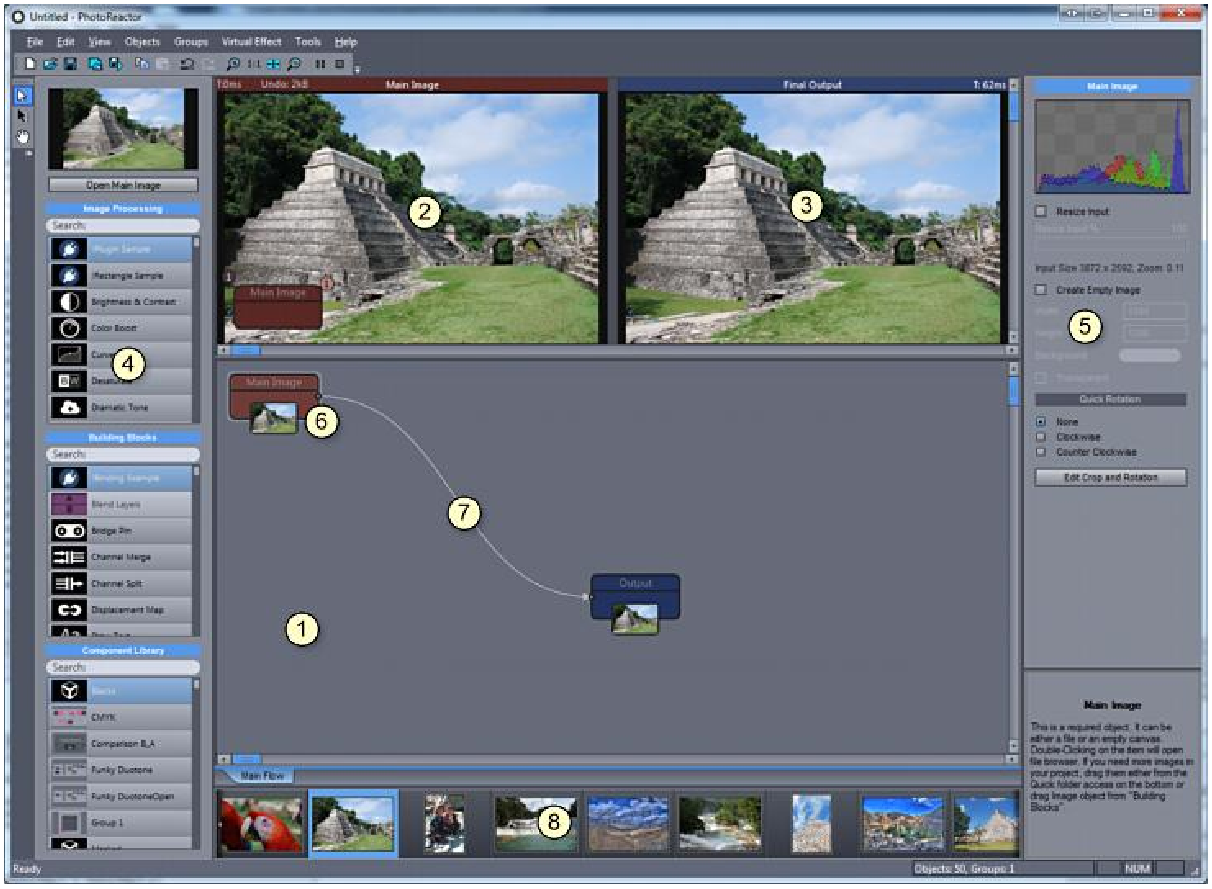


Objects and Flow
Object is the effect that is visualized as a box with nodes.
Flow are many objects connected together.





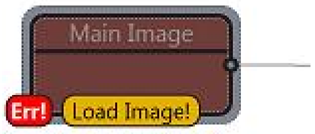
Now the Interface and its parts.



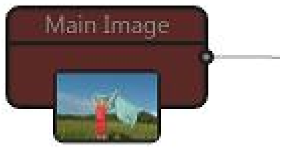
1 - Workspace. Here we place and connect the objects
2 - Left Monitor showing image of a currently selected object
3 - Right Monitor showing the final image of the output object
4 - Object lists
5 - Object parameters
6 - Object : Main Image
7 - Connection
8 - Quick Access panel

2.1 Main Image

As mentioned before the Main Image is the most important object in the flow.

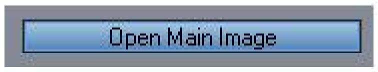


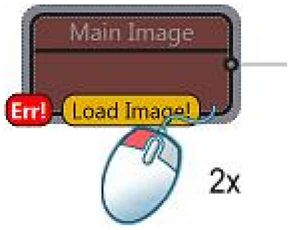
We can put many other images in the flow, but only the Main Image sets the flow document size.



Without the Main Image there the flow document is not set and the reactor will not operate.

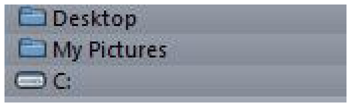
To load image into Main Image.
You can use button Open Main Image, menu (File menu) or double click on the Main Image object.



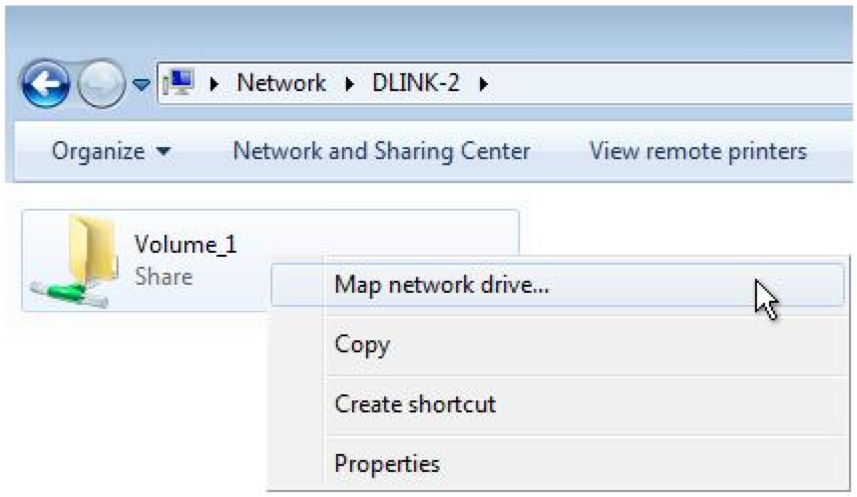


An Image **browser** will open where you can navigate to your folder.
You can scroll the browser using scroll bar, mouse wheel or by swiping up or down.

Desktop and My Pictures are added to the very top of folder list:



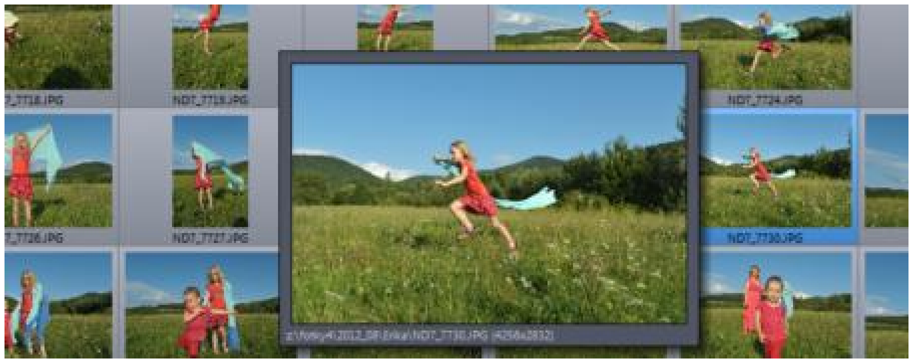
Network access
To be able to access images stored on a network volume, you need to first map the network folder to a drive in windows explorer.



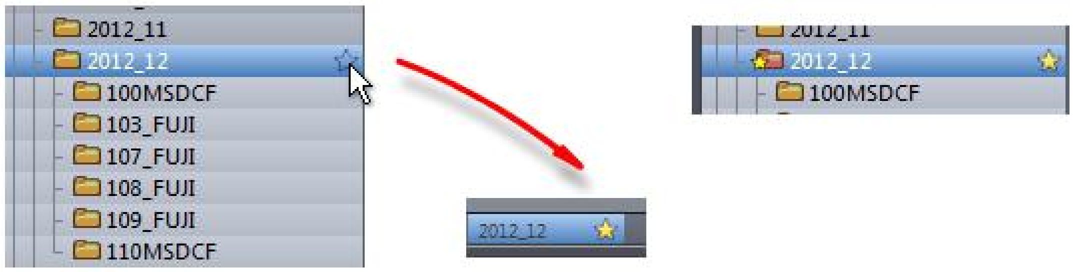
Note: This is done on purpose: Reactor does need a quick access to the images you use in its flow and mapped network drives will allow for a much faster access to them than using simple Network Names (as when you browse Network folders).

Technical reason if you care: This is because in windows networking only one of the connected device is elected as a Master Browser which knows how to redirect all the network device names to their correct addresses. Unfortunately there is no easy control over this and the Master Browser may not be (and usually it isn't) the computer you are working on (it can even be your router or a NAS drive) and the access to network files will become prolonged. (FYI: Wrongly elected or slow Master Browser or the one that goes often to sleep is the source of many home network headaches)

Tip: You can right click on image in the browser to preview it in larger size

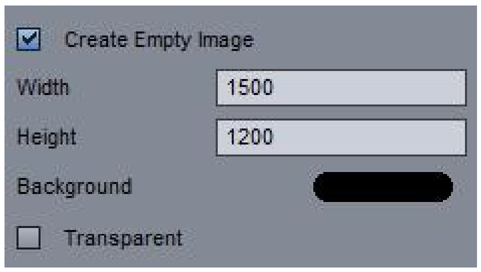


Favorites
A browser can remember folders as favorites. To add folder to Favorites, click on the outline Star at the right of selected folder.



The favorite will be added at the bottom of the browser as a tab and could be quickly accessed at any time in the future. To remove folder from favorites click again on the yellow star.

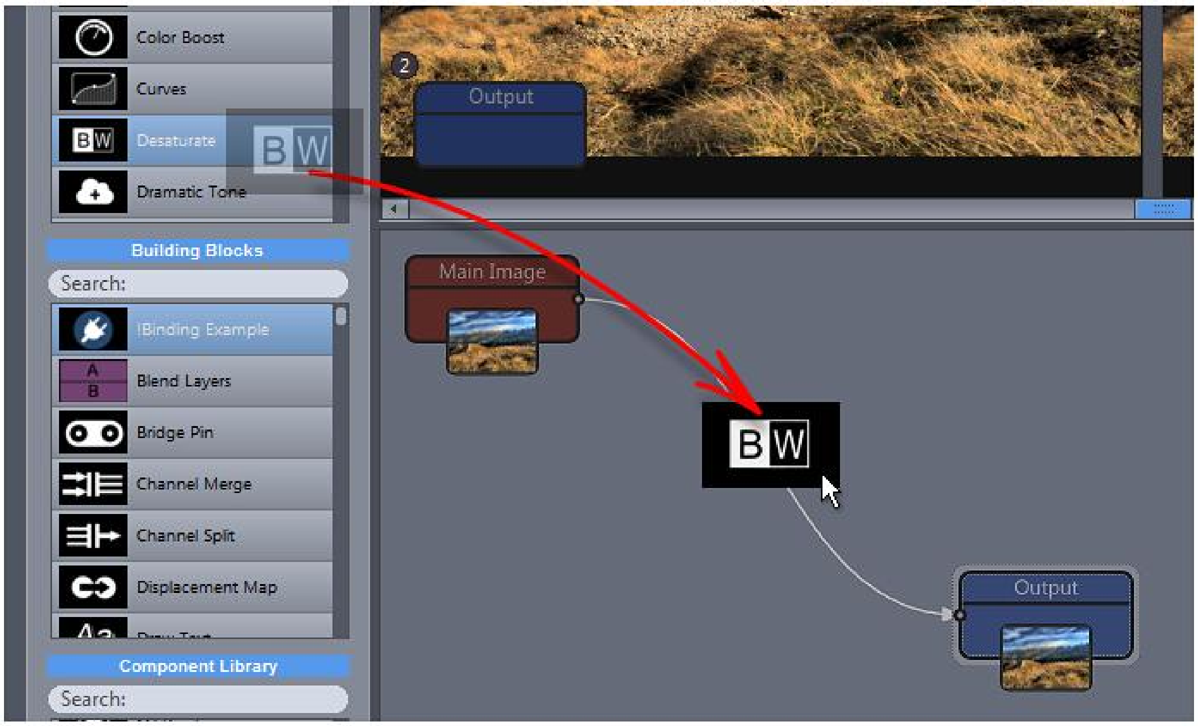
Empty Main Image
In a few special cases where no Main Image is needed (generating a texture for example) the Main Image document can be set as empty (but it has to be set)



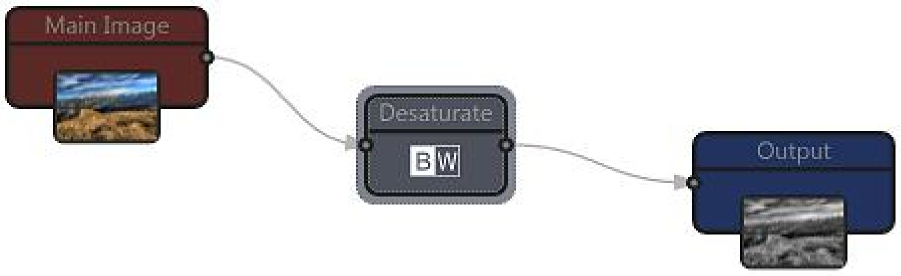
☑ Create Empty Image	
Width	1500
Height	1200
Background	●
☐ Transparent	

2.2 Placing Objects

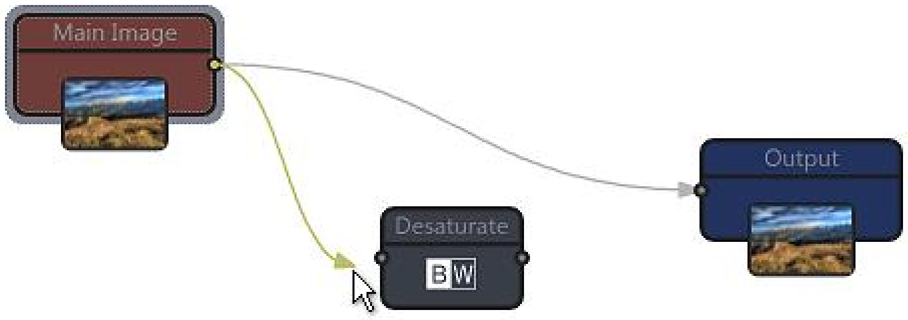
Adding objects is through Drag and Drop.



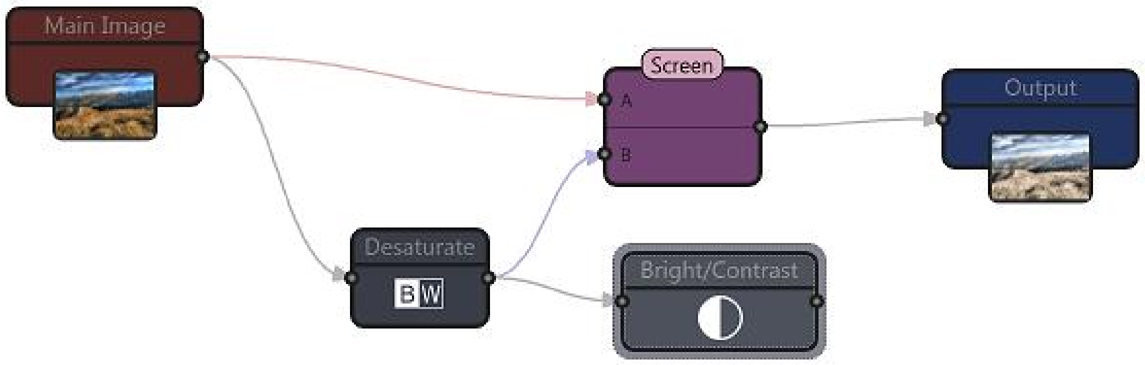
If you drop the object over an existing connection line, it will automatically "insert" itself in the connection.



If you want to connect objects manually, or disconnect them, you do it by simply pulling a line from an output of one object towards the input of another. To disconnect objects, just do the opposite: pull the connection from the input which would disconnect the object.



All objects in Photo-Reactor have maximum of one output node (they can have zero) and they may have maximum of two inputs. This is to make things very simple. Maybe you had seen some engineering products that use node editing and they would need to have many different inputs and outputs and each would require different type of data. Nothing like that here. If an object has a node, you can connect it to any other object..



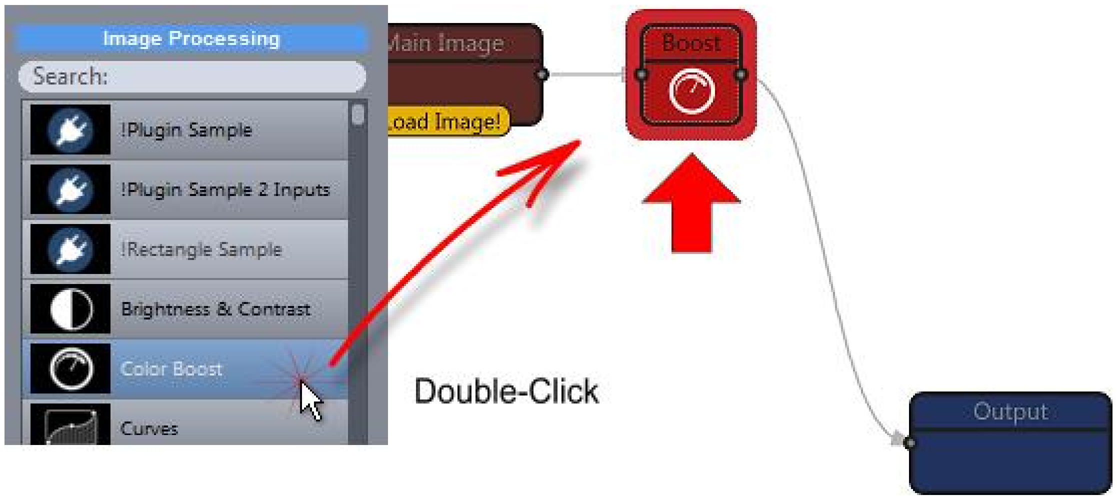
An important point is that you can connect as many objects to a single output node as you want. But each input node can have only one incoming connection which should be obvious when you think about this.
A simple way is to think about an extension cord. You can plug many devices in the extension cord outlets, but you definitely don't have extension cord with two input plugs.

Tip: You can also move the object with your keyboard using arrow keys. To move it faster hold down SHIFT.

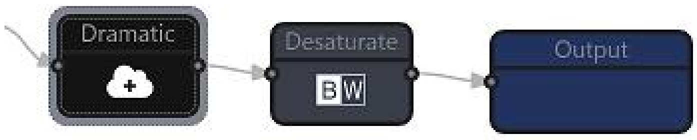
Important: In Photo Reactor all INPUT nodes need to be connected. An unconnected Input Node means error. Most objects have only one input node, few have two (Blend) or none (Generator). Some can switch between one or two (Segmentation). There is one exception to this and that is Group Masks.

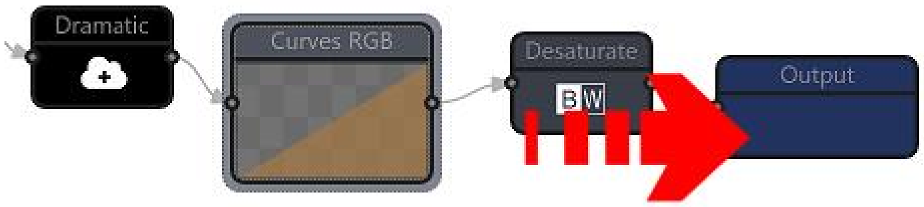
Line Insert
Objects can be inserted also by double-clicking on the item in the lists (Except the User Library). This is a very fast way how to add objects one after another.
This will add the new object **after currently selected object** and then connect it to the line. Also other objects will move right to make a space. If the the object will be inserted out of the screen a new line will

be created.
You can disable double-clicking feature in menu Edit - Add Objects with Double-Click.



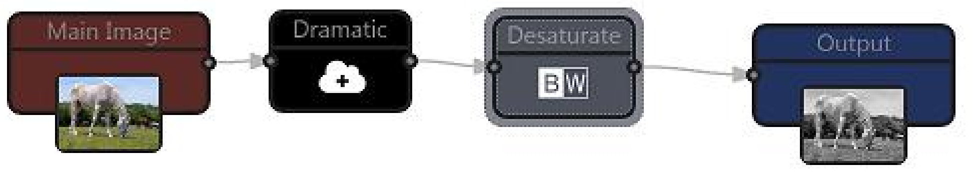
Auto-Bump
When you add objects in between other objects the existing items will move to the right to create space for the new one. In case you want to disable this feature, there is check box in menu Edit - Auto-Bump.

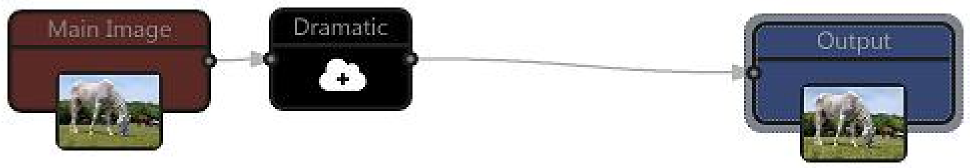




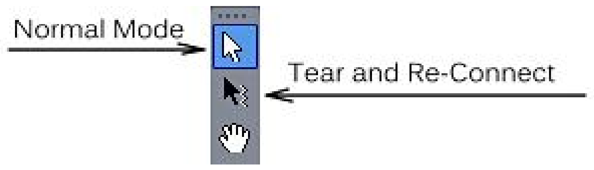
2.3 Delete, Move and Tear

To delete object simply hit the delete button. The other objects will reconnect.





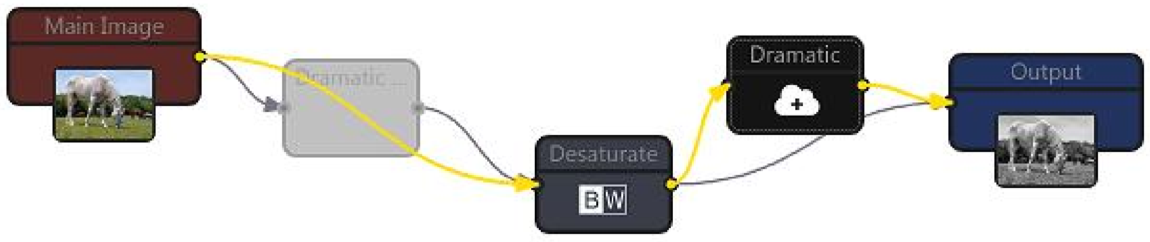
Tear and Re-Connect



There are two basic editing modes - Normal Mode and Tear and Re-Connect mode

The Normal mode is for standard moving objects and manually connecting them. If you move an object, all the connections will move with it.

The tear and Re-Connect mode is different. Moving object will **tear** it off the current connection and placing it over different line will re-connect it there.

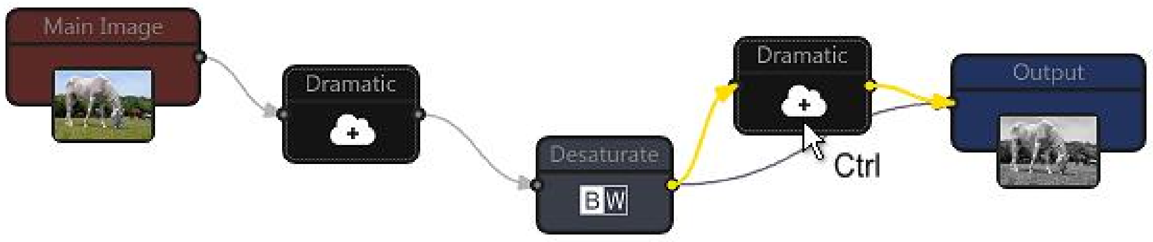


We can use it to quickly move the object to different place without the burden of disconnecting them and then connecting them again in new place. In the image above I moved the Dramatic effect from its first position to the position after Brightness/Contrast effect.

Tips and Tricks

Copy Object:

You can hold CTRL with Tear and Re-Connect and it will create a copy of the object you are moving.

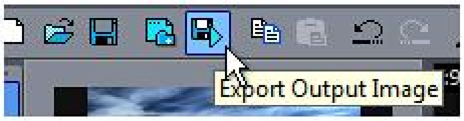


Auto-Bump
Similarly to the Auto-Bump feature when adding the objects, the items will move to the right to add space for new object. You can disable this option in menu Edit.

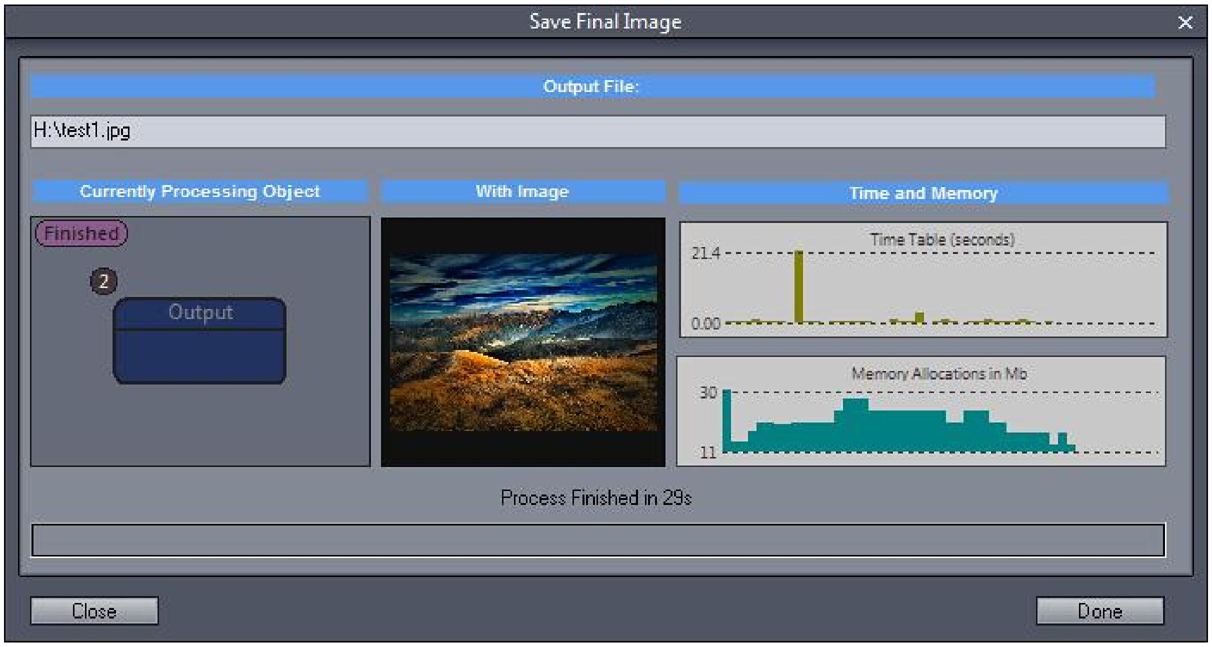
Move Object:
If you just want to move the object without tearing it up or breaking the connections, instead of switching back to Normal mode, use the arrow keys on your keyboard. Hold down SHIFT to move the object faster.

2.4 Export Output Image

To be able to see the changes immediately, Photo-Reactor Works on a Preview Image. That means we see how the output image looks like but it is still only a preview. We need to actually process the full image before we can save it.
This is done through the menu File: Export Output Image



Depending on the complexity of the flow and image size, this may take few seconds or even minutes.



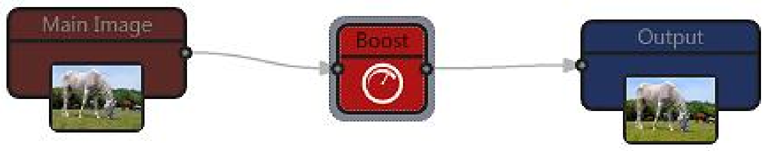
H:\test1.jpg

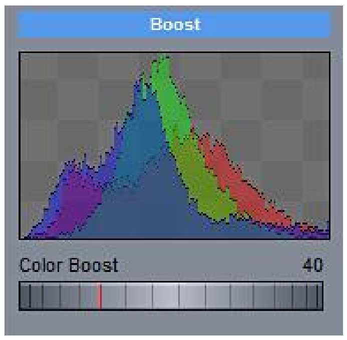
Process Finished in 29s

The Save Final Image dialog will entertain you with few screens and graphs during the final export.

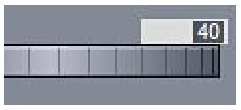
2.5 Object Properties

Object have their settings and properties. The object Properties will appear in the right panel as soon as you select any object on workspace.



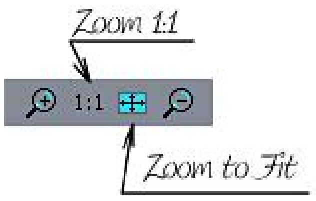


Numeric input
Sliders can be also adjusted by numeric input. You can just click on the number to edit it, or after you adjust any slider you can actually directly type the number and it will go to the numeric editing mode.

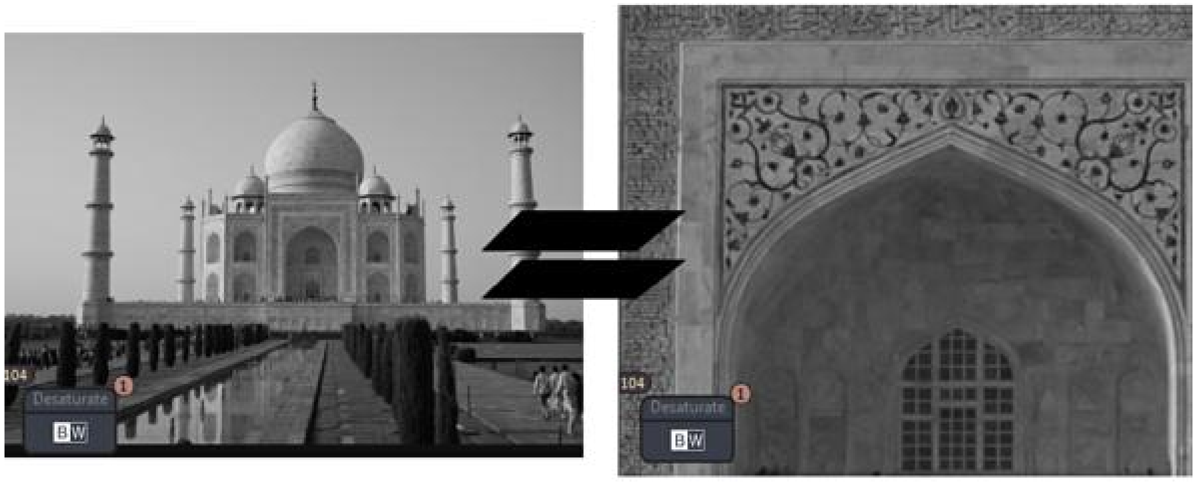


2.6 Zoom 1:1 and Fit Zoom

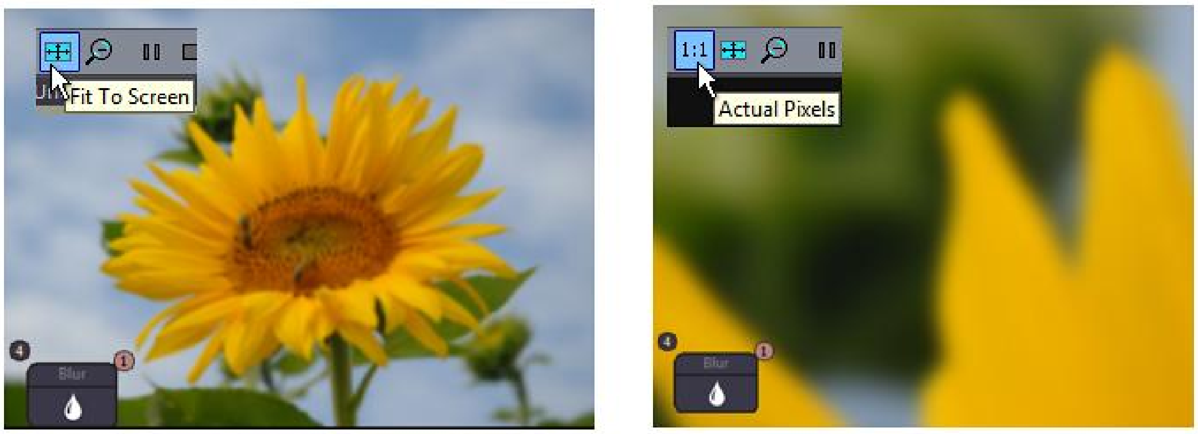
Most of the objects in Photo-Reactor scale their effects with zoom.
That means a preview does *try* to closely represent how the final exported image would look like. The keyword here is "try". In vast majority of objects this works very well. But there are exceptions.



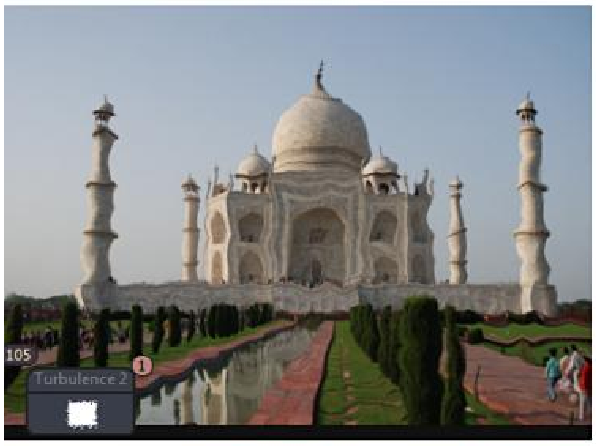
A typical perfectly sallable objects are the ones that work globally on the color, for example Desaturate, Contrast or Hue. There are no differences in effect of the small or large image. Desaturated small preview image looks equally desaturated at the large image.

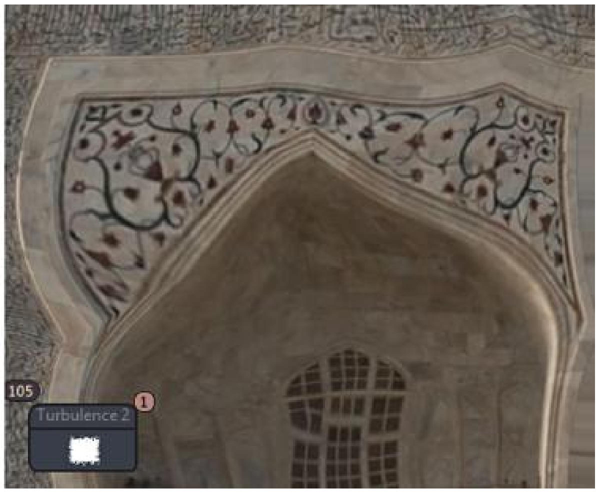


Other objects such as Blur use the surrounding pixels to do the effects and so the effect is affected by the zoom. When looking at the image in Fit to Screen the blur is less obvious as when looking at the 1:1 zoom (actual pixels). This is again consistent with the reality. If we export blur effect in a full image and then resize it to the same size as preview, the effect (its amount) should look about the same. Such objects are Blurs, Sharpening, Emboss and to some extent also displacement effects such as Turbulence... Those objects always benefit from looking at the preview in 1:1 to correctly judge the amount but they also show you how the amount looks "from distance".

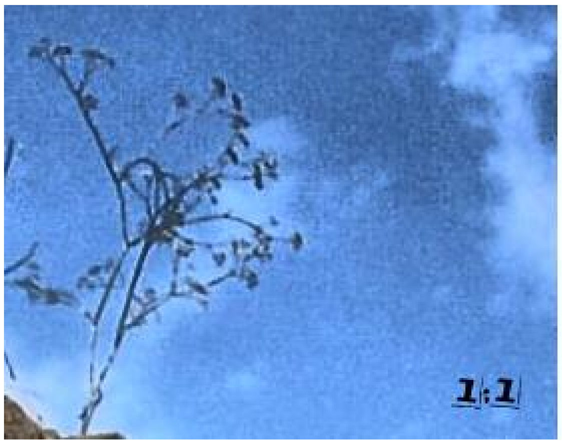


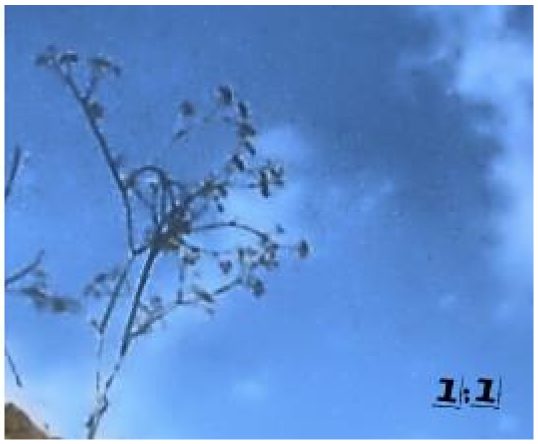
Some objects from this group that rely on randomness or placement may not exactly correspond to the zoomed in image. In this case of Turbulence, the amount and scale is correct but if you look closely the displacement is not exactly the same. (it is not perfect enlargement of the small image)



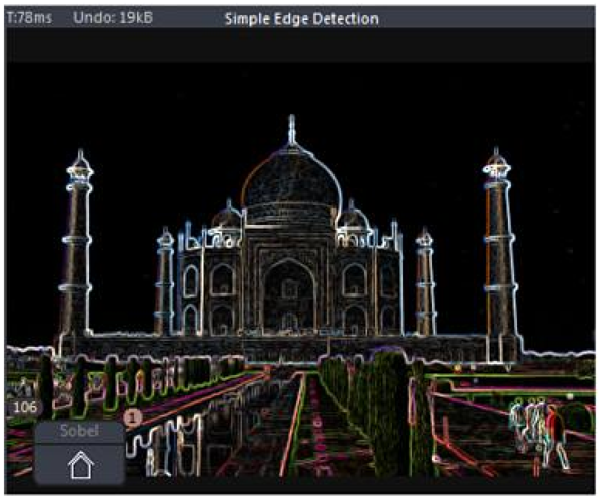


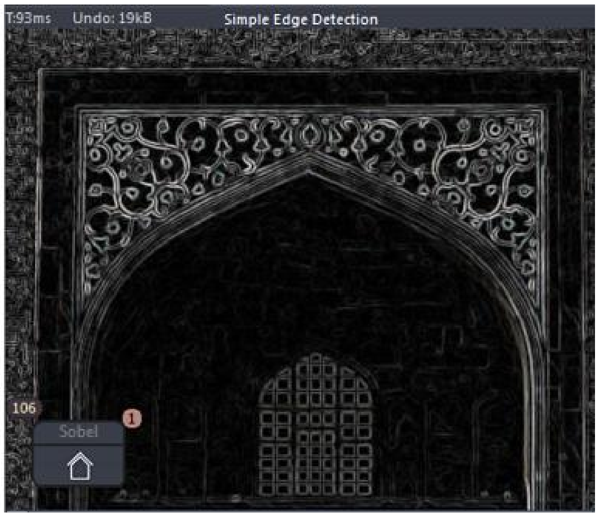
There are other effects that are inherently not scallable and they simply do require looking at them at 1:1 or the feel from the preview would be completely wrong. A typical example would be a Noise Reduction. We can't really see or judge the correct effect of NR when the image is fitted to screen. We have to zoom to 1:1 to see actual pixels, there is no other way around it.





Another good example of the effects that needs to be judged in 1:1 are various edge detection or smoothing effects. They are non-representative of the final output if looking at them in fit to zoom.


T:78ms
Undo: 19kB
Simple Edge Detection
106
Sobel
1


T:93ms
Undo: 19kB
Simple Edge Detection
106
Sobel
1

2.7 Image Processing and Building Blocks

There are two basic groups of objects. Image Processing (things like blur, hue etc..) and Building Blocks. There is a third called User Library for user created content.

Image processing blocks are typical filters and effects, like blur, sharpen, edge and many others. They all adjust the image data.

Building Blocks are objects that perform more advanced functions or functions that are more of a helper nature and may not always affect the data. Such example may be a Bridge Pin object that doesn't process its data.

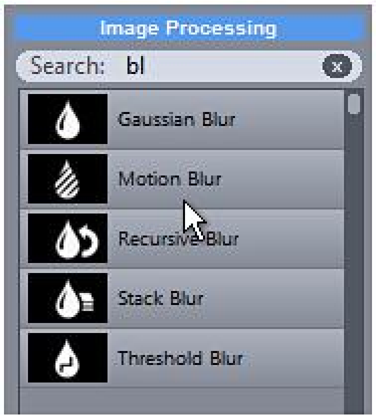
If plugins exist they will appear in either of those two lists.

2.8 List Search

Navigating the Lists.
Photo-Reactor is a modular software and as such it may over time gain a large amount of modules. The modules are sorted alphabetically and not in categories. I personally find many of the categories in other software more confusing than helpful if you are looking for particular effect but have no idea in what category it may be. While some filters are very easy to categorize such as blur or sharpen, many others are not - and photo-reactor vastly prefer those "other" effects. So instead of randomly stuffing them to not-so-descriptive categories such as Artistic, Render or Stylize as in other software, the lists use search function and objects use more descriptive names.

To search just move cursor over the whole list and start typing (you don't need to click inside the search box). Objects with any occurrence of the strings will be filtered.

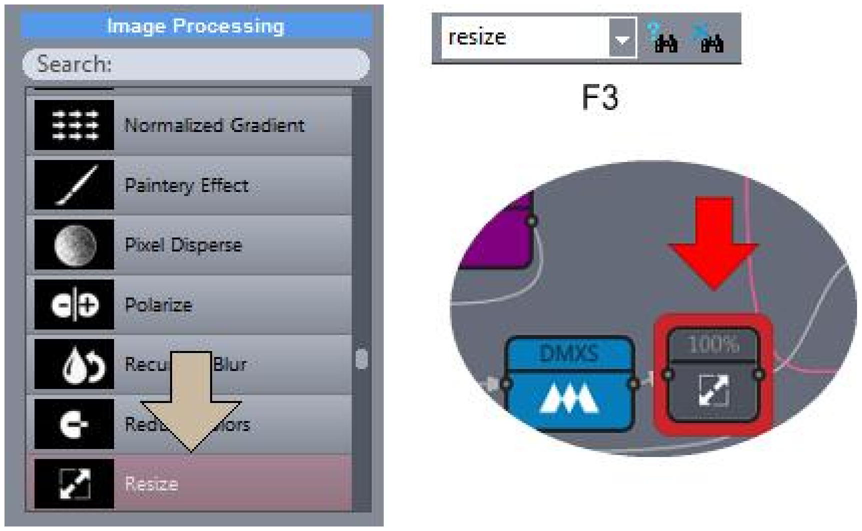
Here I want some blur so I start typing *bl* and immediately see the result of my search



The internal system is also smart and uses alternative name for search. For example if you search for *bw*, you will get to Desaturate and *circle* will give you the Simple Shape

2.9 Global Search

The global search helps you find an object by string both in workspace and all the lists. It is a great help to get your bearings with the software as it will show you position of searched object in the correct list. Pressing F3 multiple times will cycles through the objects in workspace that fit to the search string.

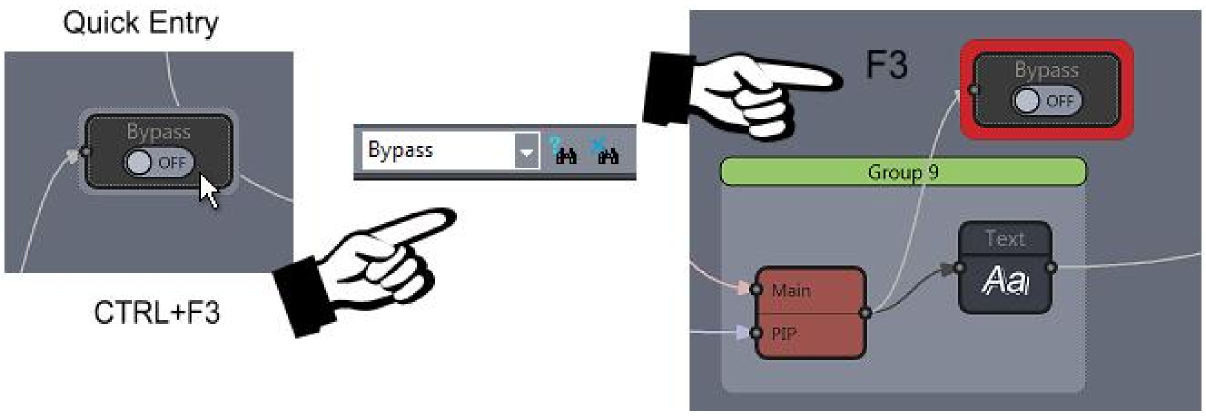


By Typing search string and pressing **F3** or the search button:

- the objects that fit to the search will be highlighted in all three component boxes
- each pressing of F3 will cycle through the workspace objects and highlight next object that belongs to the search

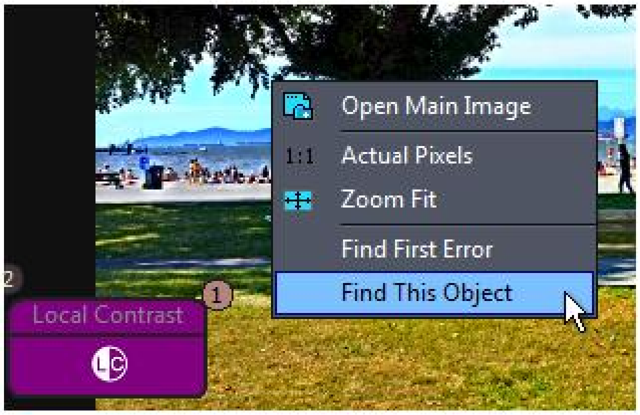
Note: The global search also search alternative names.

Quick Entry CTRL+F3
There is a quick entry with the global search. Select an object and press **CTRL+F3**. The name of the object will be entered into the search box and you can further cycle through the objects with the same name using F3. Also the object with that name will be highlighted in the object lists.

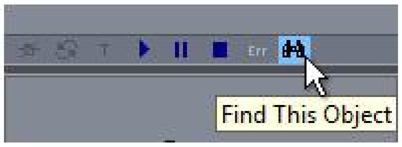


Find this Object
Third way of global search is Find this Object. This will highlight currently selected object on workspace

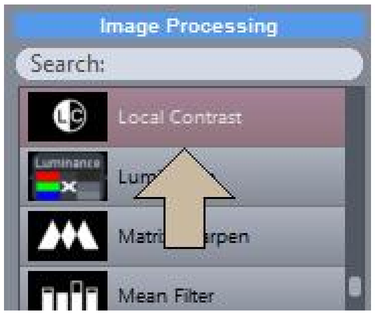
and also show you where the object came from in the lists.
Right click on the right hand monitor and select Find this Object

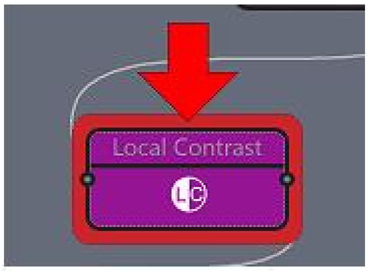


Or use the button in the object properties

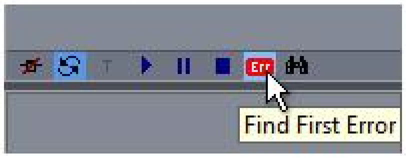


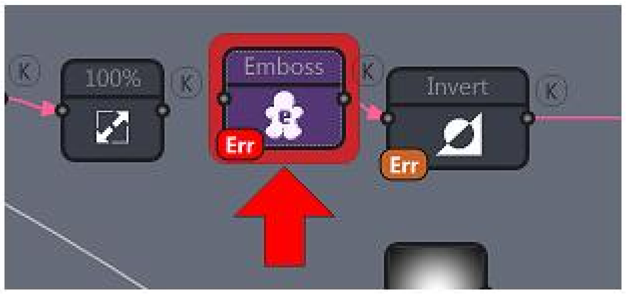
The object will be highlighted in the workspace and also where it is in the object lists





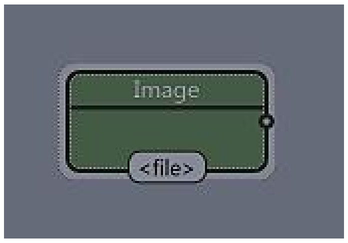
Find First Error
This function will find the first error in the flow.
First Error is the one with the red Err sign and it is where the flow error originates. It is either disconnected node or the image sizes are incompatible (for example we use resize object and then try to merge that resized image using Blend Layer with original image we will get an error.
The orange Err signs are derived errors (that means the object can't process the data, but it is due to the error of some other object or line of objects)


Err
Find First Error


100%
Emboss
Invert
Err
Err

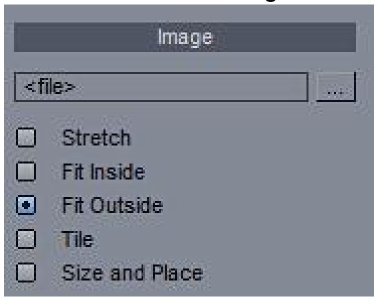
2.10 Secondary image

As described before, the Main Image must be always loaded. But you can use also more images in your flow. For example you may want to merge main image with a canvas texture.



Building Blocks: Image

There are two ways to add secondary image. Drag the Image object from Building Blocks, then use the browse button in settings to load the image file.

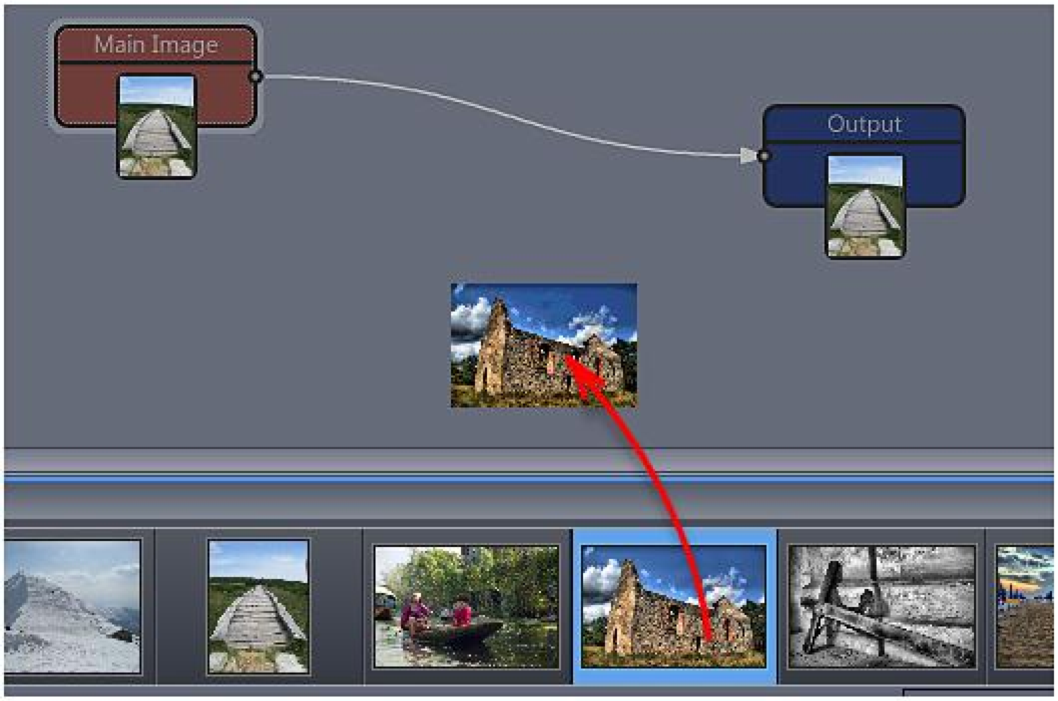


Image

<file> ...

- [ ] Stretch
- [ ] Fit Inside
- [✓] Fit Outside
- [ ] Tile
- [ ] Size and Place

.. or drag and drop the image from Quick Access bar.



The quick access bar displays images from the Main Image folder. You can load different folder to the Quick access bar using Open Quick Access Folder in menu File.

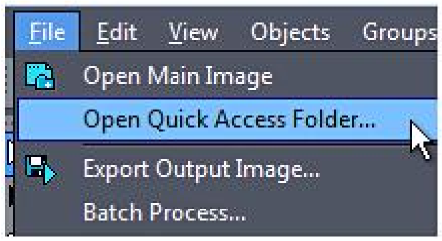
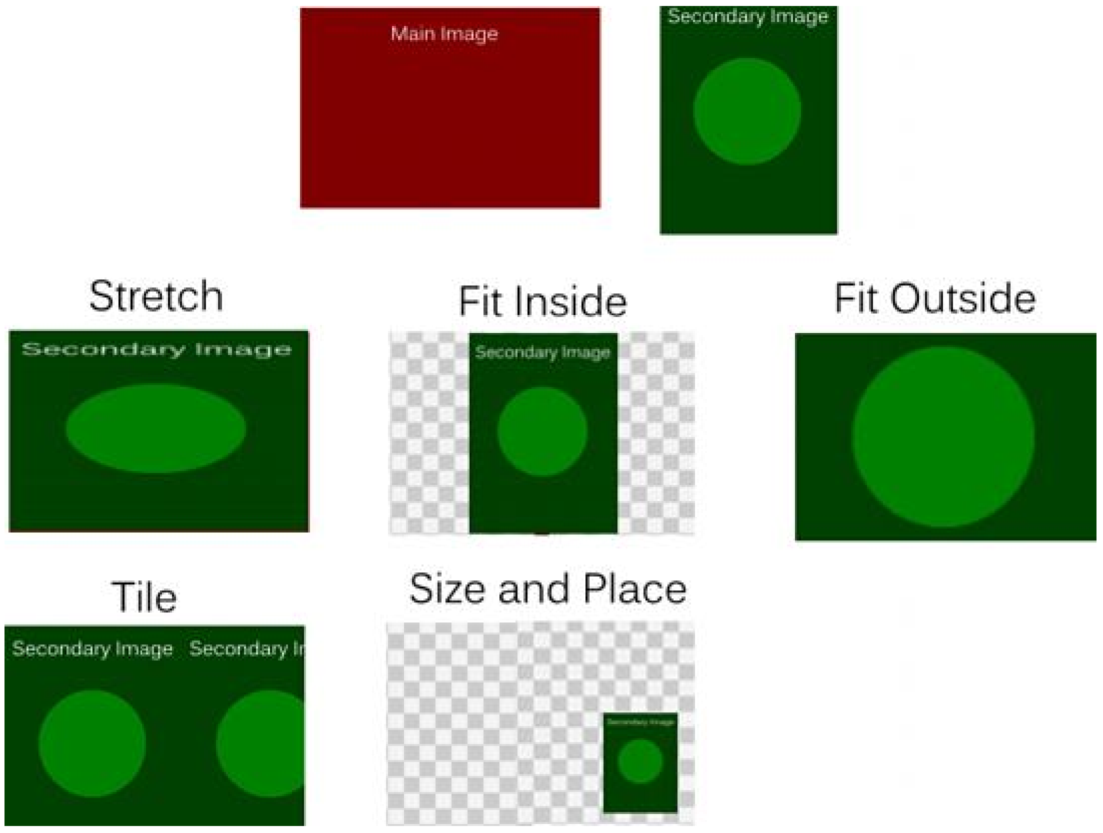


Image Size
The Main Image is the "orchestra conductor" and it sets the whole document size. The secondary image will be always fitted to that document size so the data can be easily combined. The various settings in the Image properties will determine how that resizing/fitting will be done.
It could be stretch, fit inside, fit outside, tile or place.



The fitting type depends on the application. For example if you want to add a canvas texture and you have seamless canvas file, then you use Tile which also lets you to adjust the tile size. If you would like to add a watermark or logo to the corner of your output, you use **Size and Place,** etc**.**

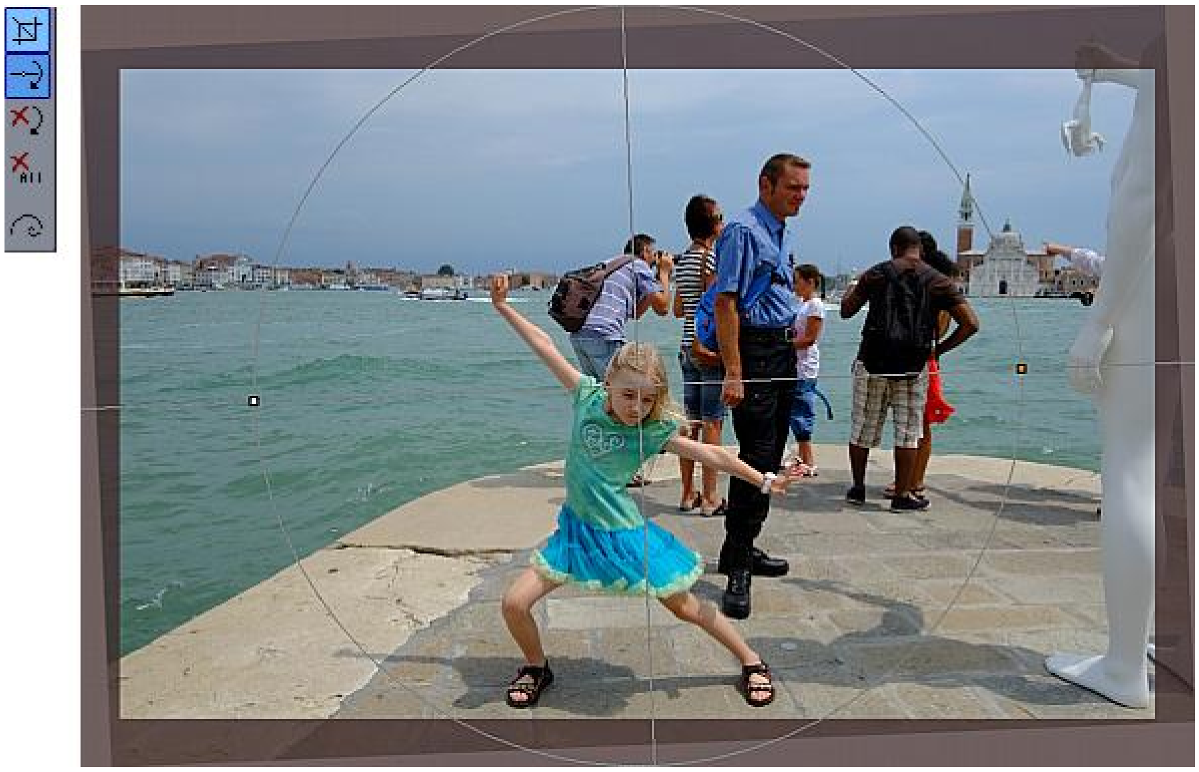
Absolute and Relative paths
The image file is normally set with an absolute path so it can be loaded from disk when you load the project.
However Reactor will look for the file also in the application folder **textures** subfolder if it cannot be found with absolute path. This way if you need a specific secondary image (for example a special texture) but you want to share the project with others, you can supply the texture image with the project and instruct

the other person to put it in his textures folder in the Photo-Reactor application folder. Example C:\Program Files\PhotoReactor\textures

2.11 Edit Crop and Rotation

Both Main Image and Secondary Image have this option.
Here you can crop the image and level the horizon. Cropping tool is the secret weapon of a good composition (of course it doesn't hurt if we start with already good framing technique during taking the photo). It is also important to check for level horizon (nothing says more about amateur photo than horizon that falls to one side)

Note: In a case of Main Image, changing crop will also change the Output Size and aspect to correspond with the new crop.



Fibonacci Golden Spiral
Last option in the toolbar is to overlay golden spiral.

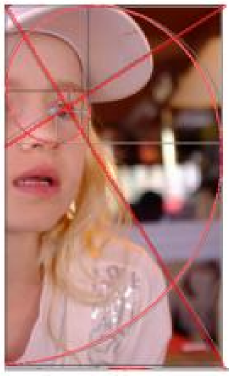
Why some composition work and some don't?

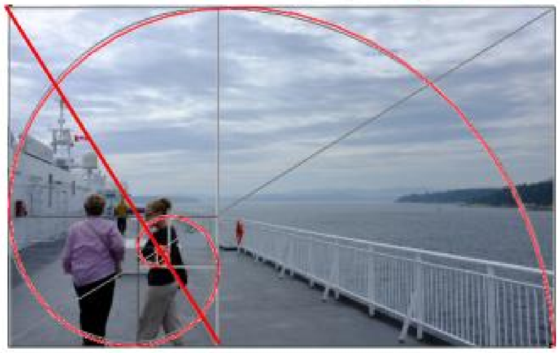
In this composition both the image was cropped and a special light was added so it follow the Fibonacci spiral.

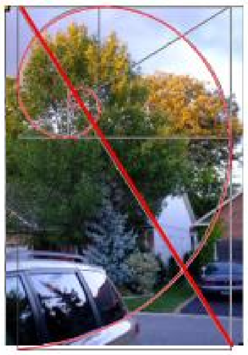
Our brain has a pattern seeking ability and it is always engaged trying to find known patterns in everything we look at. A composition that follows the golden ratio/spiral would register as naturally pleasing because our eyes are unconsciously drawn to the center of the spiral which is similar to the pattern of nautilus shell, rose, sunflower....
By cropping the image so it fits best into a spiral we can dramatically improve many compositions.

Pressing the Golden Spiral button will flip the spiral in 4 different ways. Flipping the spiral and adjusting the cropping, we can find the best fit where the eye of the spiral points to where we want the viewers attention. (often faces, eyes on the portraits, center of flower...) The goal is to have the action within the body of the spiral shell.

Most of the images below are just a snapshots, but they were either composed or cropped to follow the Fibonacci spiral to create higher impact.



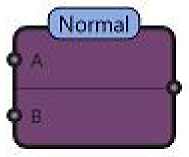


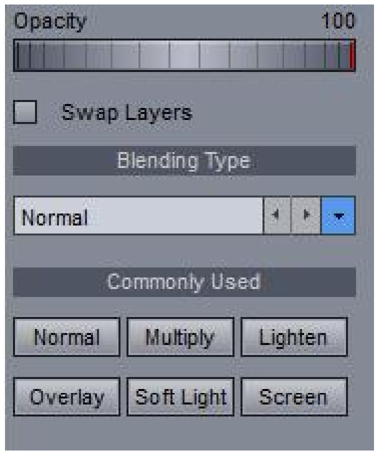


2.12 Blend Layers

Blend Layers is a key building block in Photo-Reactor and it allows you to mix two input data or images using various methods. Its functionality is the same as the layers in a standard image editor, except in Photo-Reactor the layers can be also used sidewise.
What it means is that in a standard editor the layers are each on top of another in one big pile. In most real life scenarios we don't want to blend every layer with every other layers so we have to result in flattening and duplicating some of the layers. This is where the standard editor suddenly lose all the dynamic aspect of a layers. In Photo-Reactor we can easily blend sidewise (more about this later in the chapter)

Blend Layer has two inputs and plenty of Blending types to choose from in the properties. The blending type is the way two images are mixed together. If it is Normal then the image A (top layer) is simply placed on top of Image B. If it is Multiply then the image A is multiplied with Image B and so on.



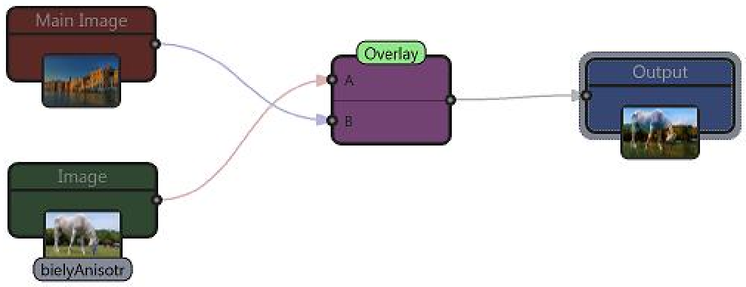

Opacity 100

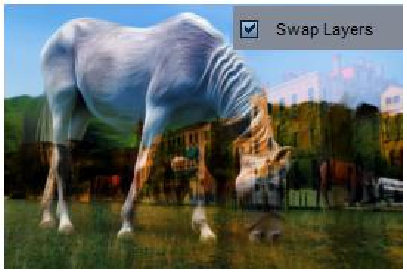
Swap Layers

Blending Type

Normal

We have also the very useful option to **swap layers**. Some blending types don't care about layer order - for example multiply or lighten is reversible, it doesn't matter which layer is on the top, the result is the same. But many of the blending types are not reversible. For example Overlay result depends on which layer is the top one and which is bottom. Instead of **reconnecting** the inputs and swapping the connections we have the quick options to simply swap the layers internally.

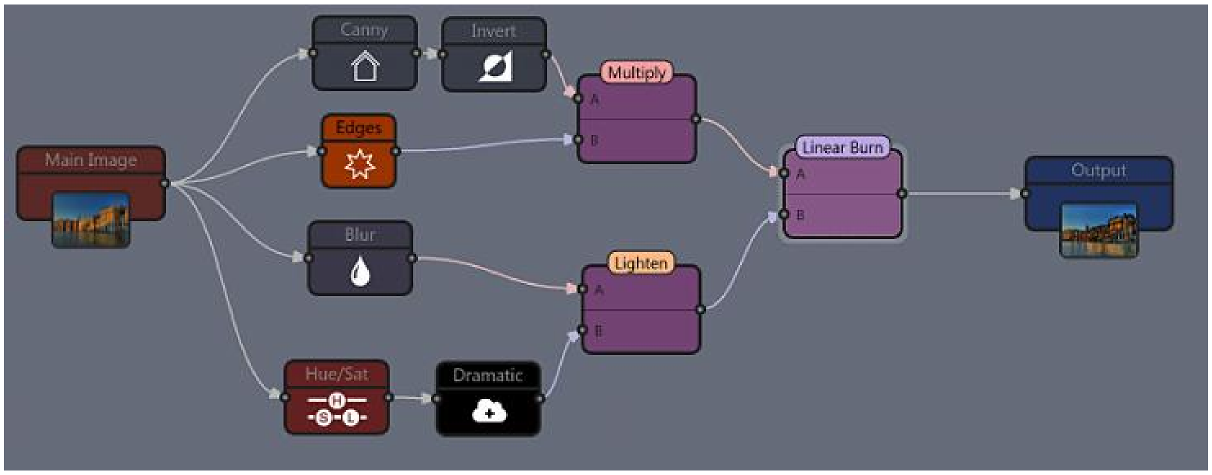




The second image is the reverse result when the buildings (Main Image) is the top layer A and horse is the bottom layer B, but we did this internally by simply setting Swap Layers to ON. We didn't have to swap the actual connections.

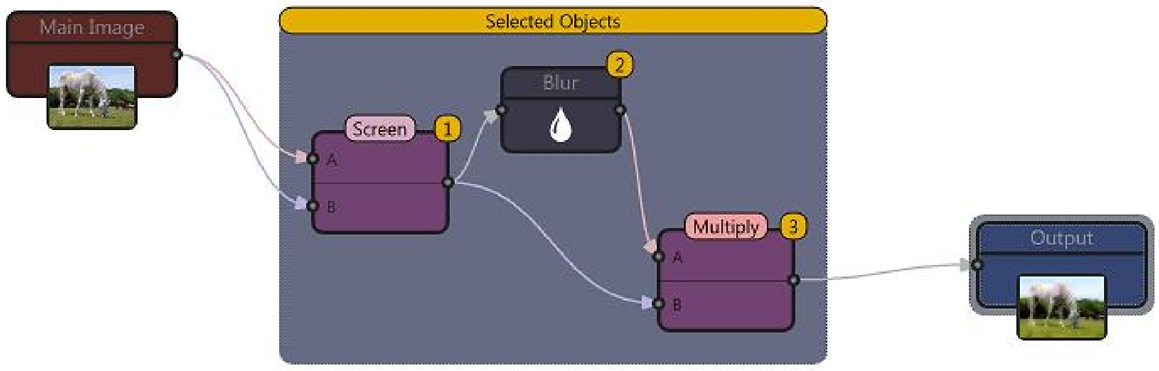
Blending and blending types are staple function of every image processing. They are often hard to describe verbally and even if you do, the usefulness of such verbal definition is very limited.
It is simply better to try them and get the feel of the effect. The Blend Layers object has also few buttons to quickly set the most used blending types, but there is far more hidden under the combo box. Many books and online tutorials deal with the blending and the underlined mathematics operations (which are in most part simple)

Sidewise vs Cascade
An example of a sidewise blending is the flow below, which can be used as a great exercise or a tricky puzzle for students of standard editors where the goal would be not to allow the students to use flattening of layers.



2.13 Selection

If you need to move more than one object or create a group, you need to use selection tool.
This is done simply by "drawing" a rectangle around the objects. Selections can be created also by holding SHIFT and clicking at the objects.
To move the selection use the yellow selection handle on the top.

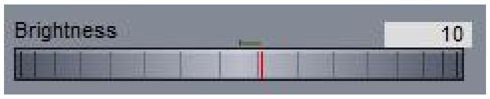


2.14 Editing Parameters

Parameters of selected object will appear in the right panel.
Most data entry is done through Dials.

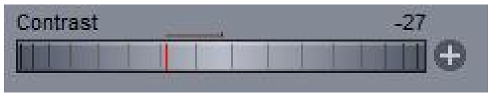
Note: The right panel properties will fully function only if there is Main Image loaded. Without the Main Image there is no document and no data and the properties may not react as intended.

Manual editing
You can enter the numbers manually by clicking on the number in the slider. Press Enter to update (Also clicking somewhere in the properties window works as enter)



Tip: Quick Entry: When you select a dial by clicking on it or moving it, you may just start typing the number (without clicking on the number in the dial)

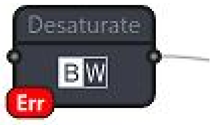
Parameters Group: Add
A Special case: Right Clicking on the control will show a plus sign. This is for adding the parameter to Parameters Group and it is described later.



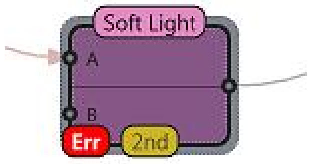
2.15 Errors and Warnings

There are few kinds of errors and warnings. An error occurs if you don't connect an input node.

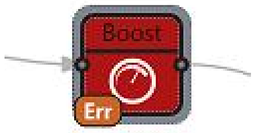
Error
This means the input is not connected. This the first occurrence of the error and it is marked by a bright red background.



Err 2nd
The second input is not connected

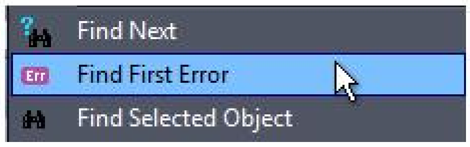


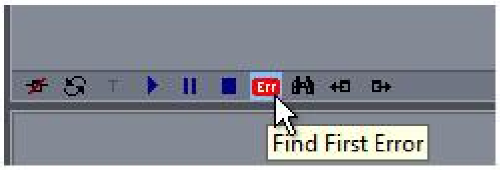
Inherited Error (Recursive error)
That means the object doesn't receive any data from the object(s) it is connected to. The error background is dark orange.
This error will be also triggered when you connect two or more objects in an infinite loop.



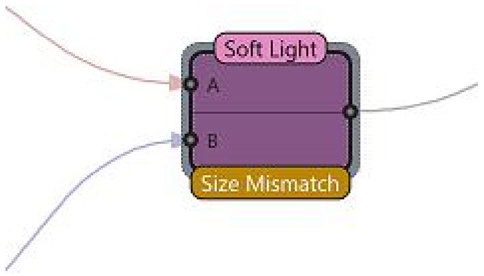
It is an inherited error - that means there is nothing you can do with this object, you have to find where the error comes from (bright red error).

Tip: To find First error, use menu Edit Find First Error or the button in the object Properties.



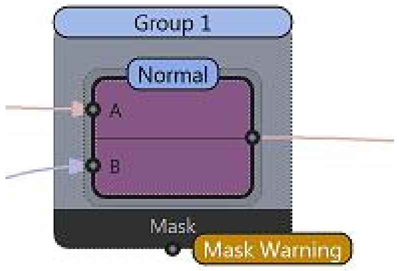


Size Mismatch (Warning)
This warning can appear on multiple-input objects such as Blend Layer
It means the A and B inputs don't receive images of the same size.
This is only a warning, both images will be automatically resized to the smaller one so the object can continue, however this may be a potential mistake in your flow design. (This can occur if you use Resize objects at various stages in the flow and then you try to merge two different sizes by mistake)



Mask Warning

This means the Group Mask Channel was not connected. This is only a warning and the objects inside the group will be processed as if there is no Group Mask Channel set.

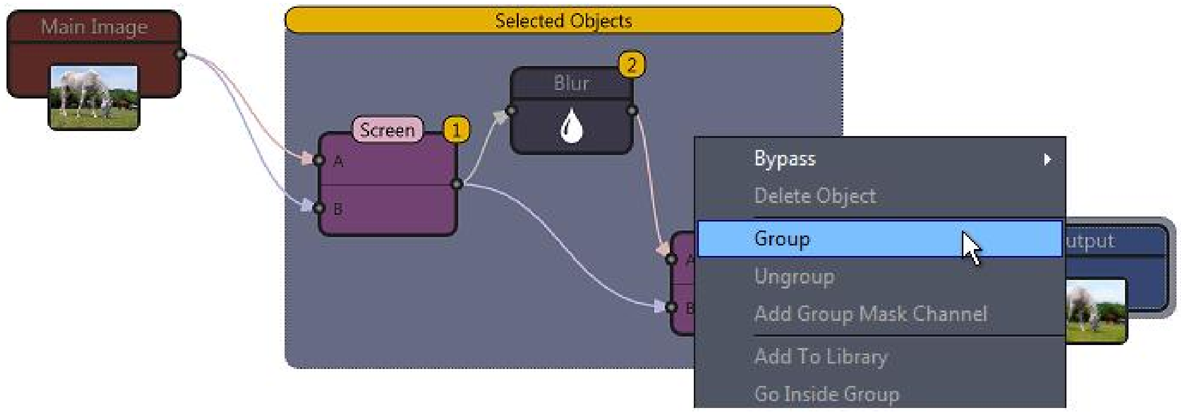


3 Advanced

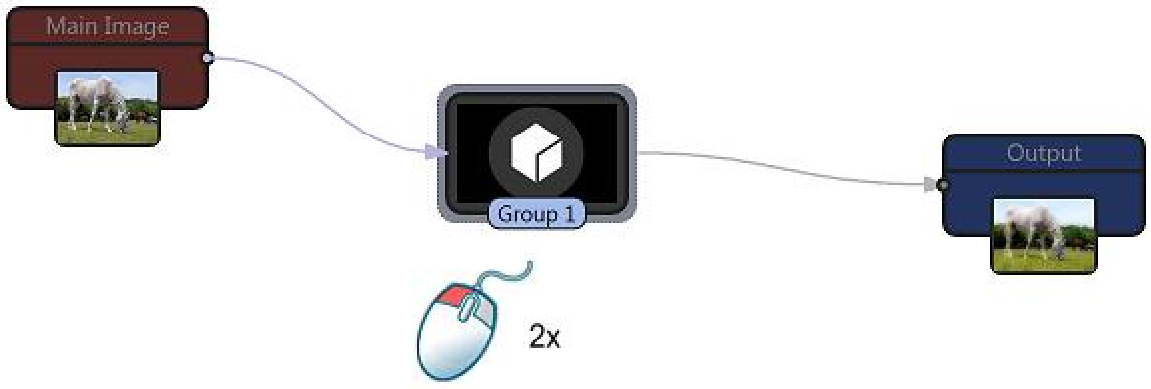
3.1 Groups

Groups are great way to simplify the spaghetti code on the screen yet still have editable flow.

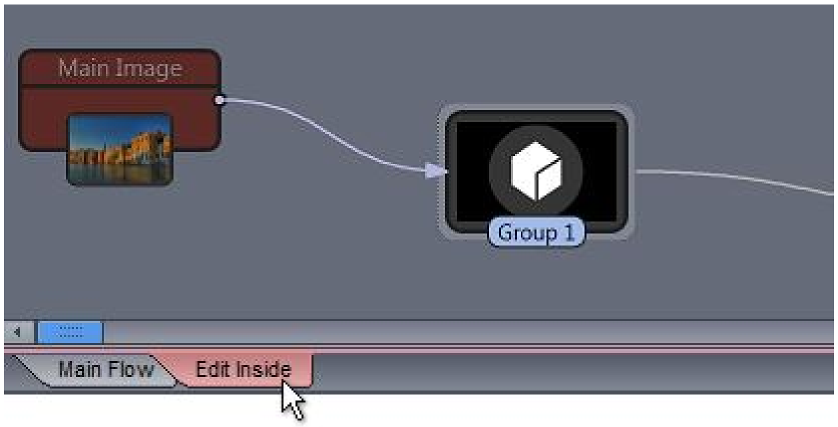
To create Group, select one or more objects by SHIFT click or by drawing a frame around the objects, then right click on the selection and use Group.
Note: There is a reason why a single object can be a group too, but that is for later.



The most interesting aspect of the Group is that they can be minimized, and that is done by simply **double-clicking** on the existing group.



Edit Inside
Groups can be also edited on a separate screen using the Edit Inside tab. Select a group and click on the Edit Inside tab.



This opens the content of the group in a separate workspace. When you are editing Inside the group you can easily add new object by drag and drop to the group.

Adding object to the group

- Drag and Drop object when you are Editing Inside
- or Click on the group to select it, hold Shift and click on the object to add it to the group.

Remove object from the group

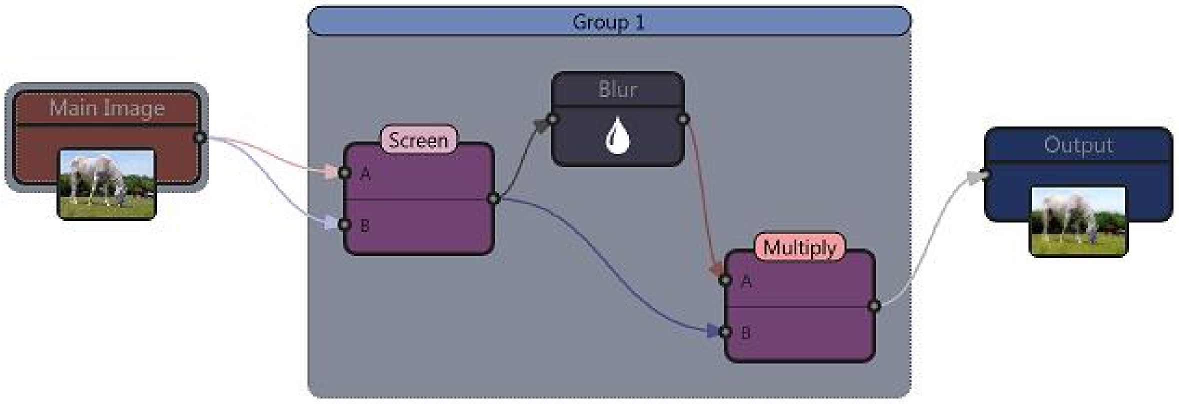
- Click on the group to select it , hold Shift and click on the object inside the group.
- or use right click menu

3.2 Adding Bridge Pins

Minimized groups are great space saver, but there is one problem, you cannot really connect or disconnect minimized group. You have to open it then connect or disconnect the objects.

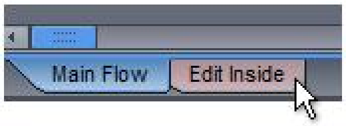
This can be solved by a small smart objects called Bridge Pins.

Lets work on the previously created group.

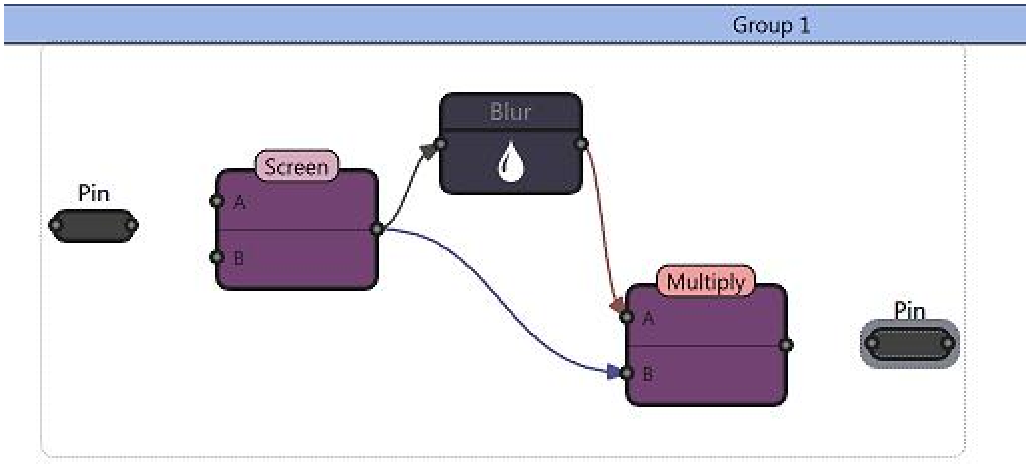


To add bridge pins we can either ungroup the group or use the previously explained Edit Inside feature. We will do it the second way.

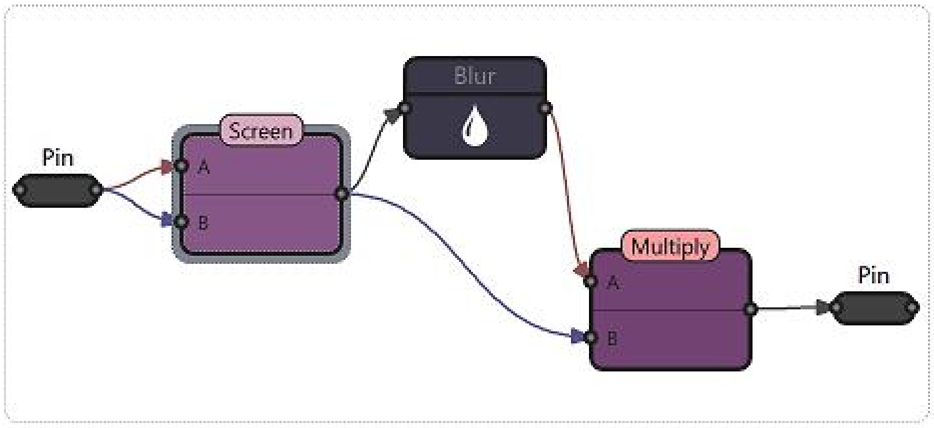
Select the group, and click Edit Inside tab



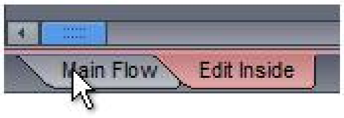
Drag and drop two Bridge Pin from Building Blocks



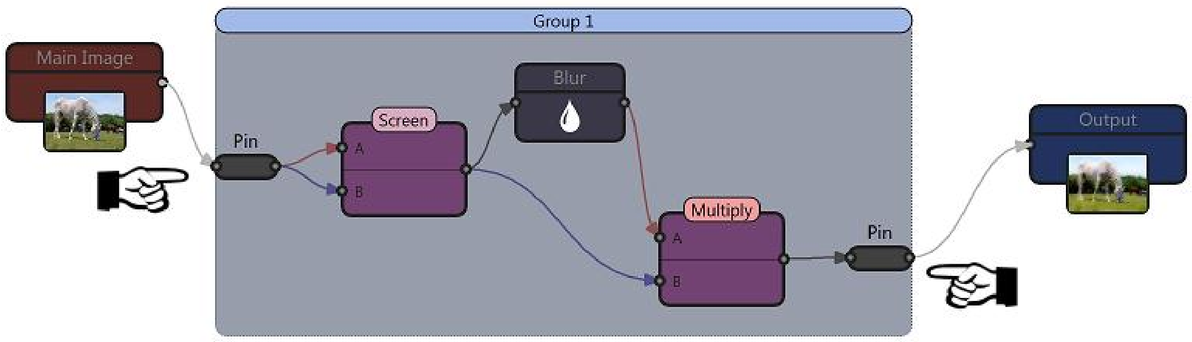
Connect the pins.



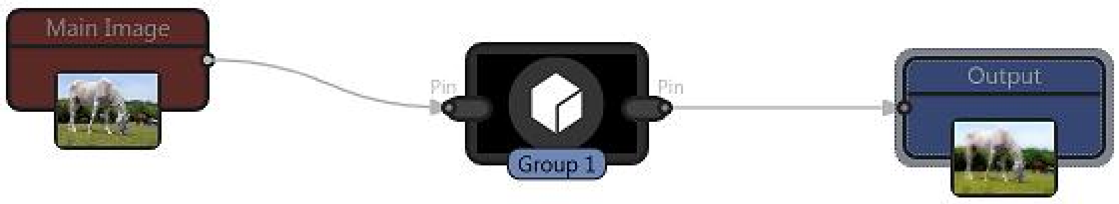
Most likely errors will appear because by connecting pins we disconnected the group, but that's fine. Click back on Main flow tab.



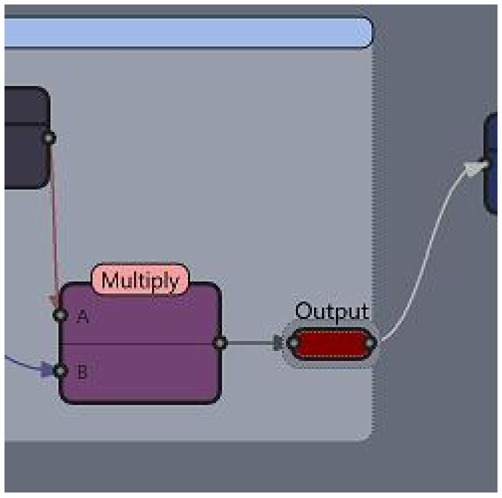
Connect the pins to Main Image and Output.

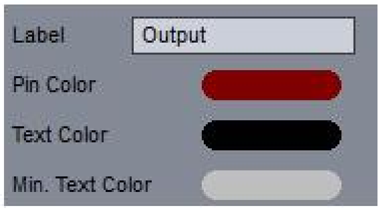


Now you can minimize the group and the pins will remain visible.



You can also rename the pins and change the colors: Open the group, select a pin and change the Label in the Pin properties. Press enter when you are done.





Label	Output
Pin Color	🟥
Text Color	⬛
Min. Text Color	⬜

- Pins can be used to create connection points on the edges of groups that work both in minimized and maximized state.
- You can have many connection points in a group

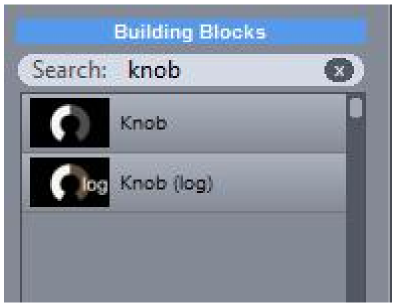
3.3 Binding Objects

We had created a group and by minimizing it we can easily tidy up the workspace.
But there is a tiny problem. If we would like to adjust the Blur radius of our object, we have to open the group, then click on the blur.

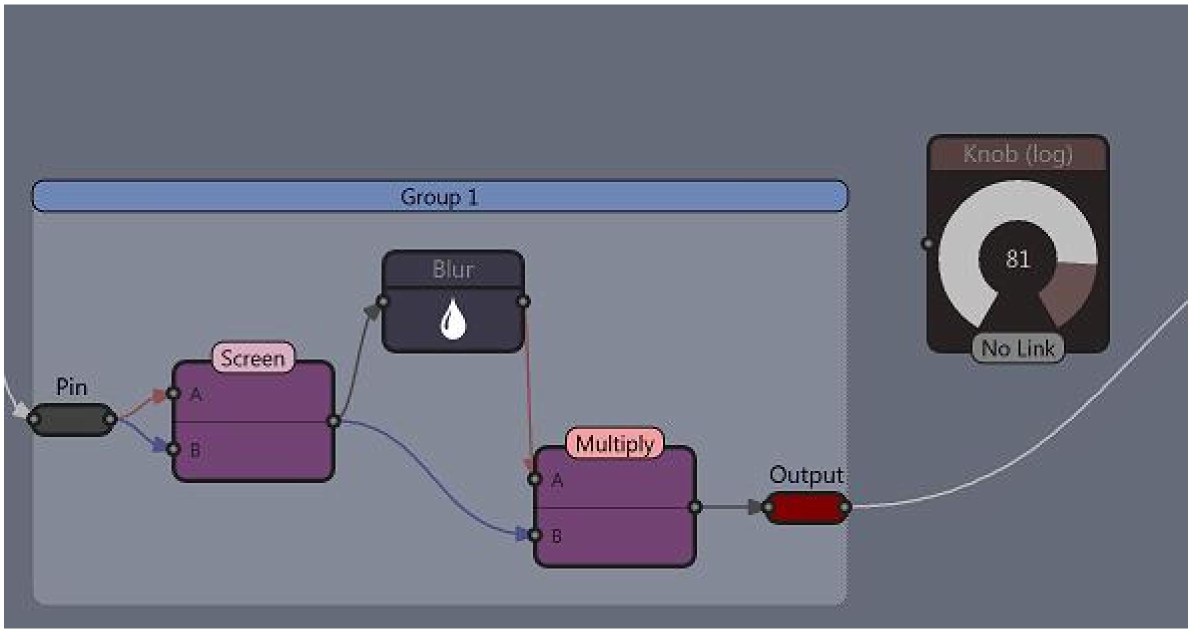
A special class of objects are called Binding Objects.
The Binding objects do not process the image, but attach to other objects and can change some of their values.

We will add one binding object outside the the group that will attach to the Blur Object.

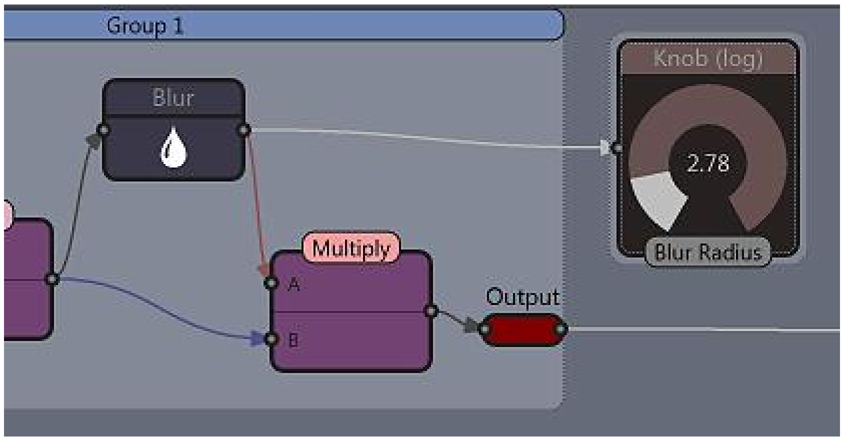
Go to Building Blocks and type knob. This will show two objects, Knob and Knob (log). We want the Knob (log) because the Blur radius is actually logarithmic value. You may not know that of course and normal knob will work as well, except you will have hard time to set small values. Even the knob at value 1 will already produce a significant blur.
The log Knob goes very easy from beginning and speed up towards the end of its scale.



Drag the Knob (log) beside the group. Don't drag it on the group as it will disappear behind the group. You cannot add object to groups in Main flow (read the previous chapter about Edit Inside). In any case we want the Knob outside the group

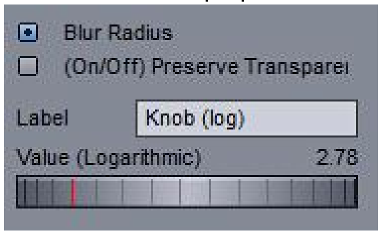


Notice the Knob says **No Link**.
Now Connect the output of Blur to the Knob (log)



Two things will happen, the Knob will change to Blur Radius and its value will change to the value of Blur Radius.

Look at the knob properties


● Blur Radius
☐ (On/Off) Preserve Transpare

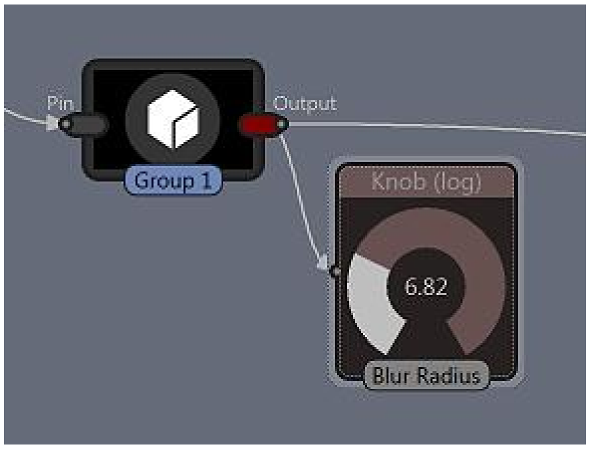
Label: Knob (log)

Value (Logarithmic) 2.78

The Binding objects will list all the parent object parameters and automatically choose the most appropriate when you connect them. Right now it is attached to the Blur Radius (that is anyway the only reasonable link, the other is a check box)

If you turn the slider in the properties or "rotate" the knob on the workspace with your mouse, the value of the Blur Radius will change.
The Binding object is attached to the Blur object.

Now you can minimize the Group and you can still adjust the blur value with the knob.



Note: Binding Parameters can bind to any object regardless if it is in a group or not.

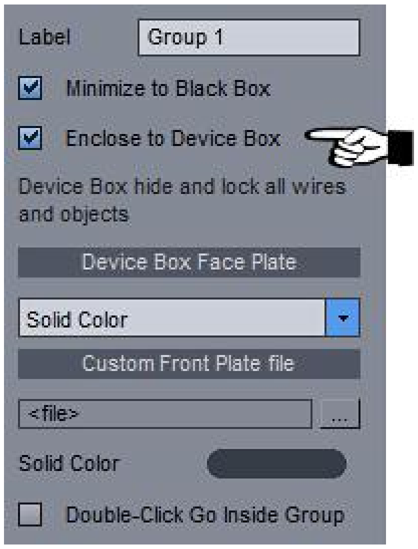
Type of Binding Parameters.
You can see all binding parameters by typing binding in the Building Blocks search box. There are Knobs, Switches and Sliders

3.4 Devices

By binding knob to the object inside group we now have two objects to drag around: the minimized group and the binding object. That is probably fine, but it isn't very stand-alone solution, especially imagine you have more binding objects attached to the group. It is like dragging kids around a toy store.

If you look at the Group Properties, you may have seen the check *Enclose to Device Box*

Set it ON.


Label: Group 1

✓ Minimize to Black Box

✓ Enclose to Device Box

Device Box hide and lock all wires and objects

Device Box Face Plate

Solid Color

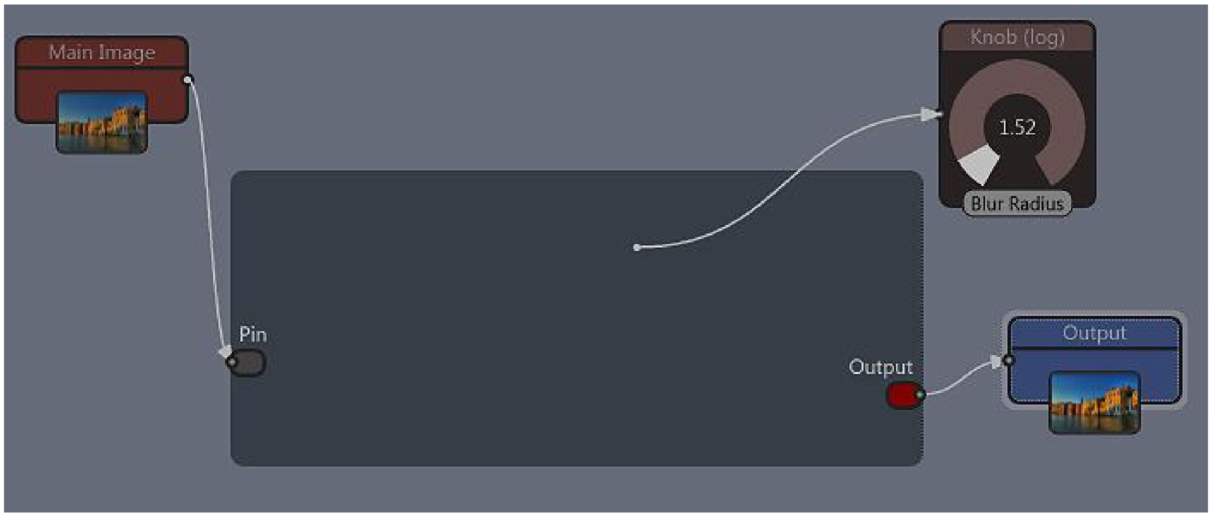
Custom Front Plate file

<file> ...

Solid Color

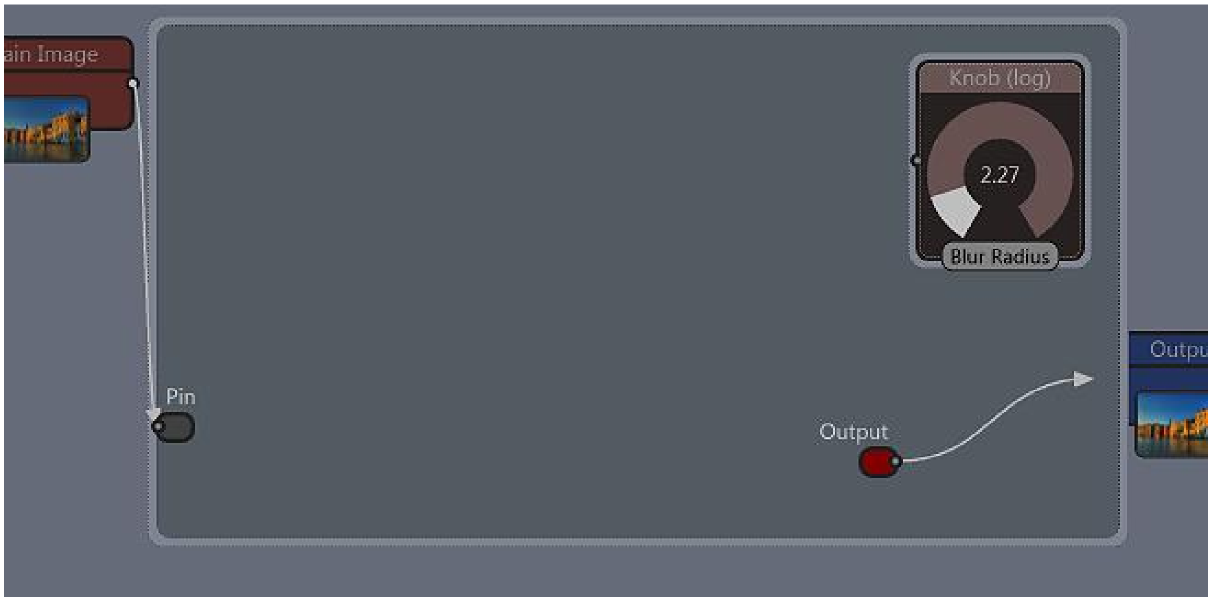
☐ Double-Click Go Inside Group

If your group was minimized, nothing much will change. But double-click on it to open the group and it will appear empty!



It is not really empty, switching to Device Box will hide all non-helper objects. In our case those were all the objects except Bridge Pins. You may see where I am going with this. The Knob is a helper object! So it can be inside the device and it will be visible while nothing else will.

Let's add it to the group then. Click on the Group (pardon, the Device) to select it. Hold Down SHIFT and click on the Knob. After a small pop quiz the Knob will be added to the device.

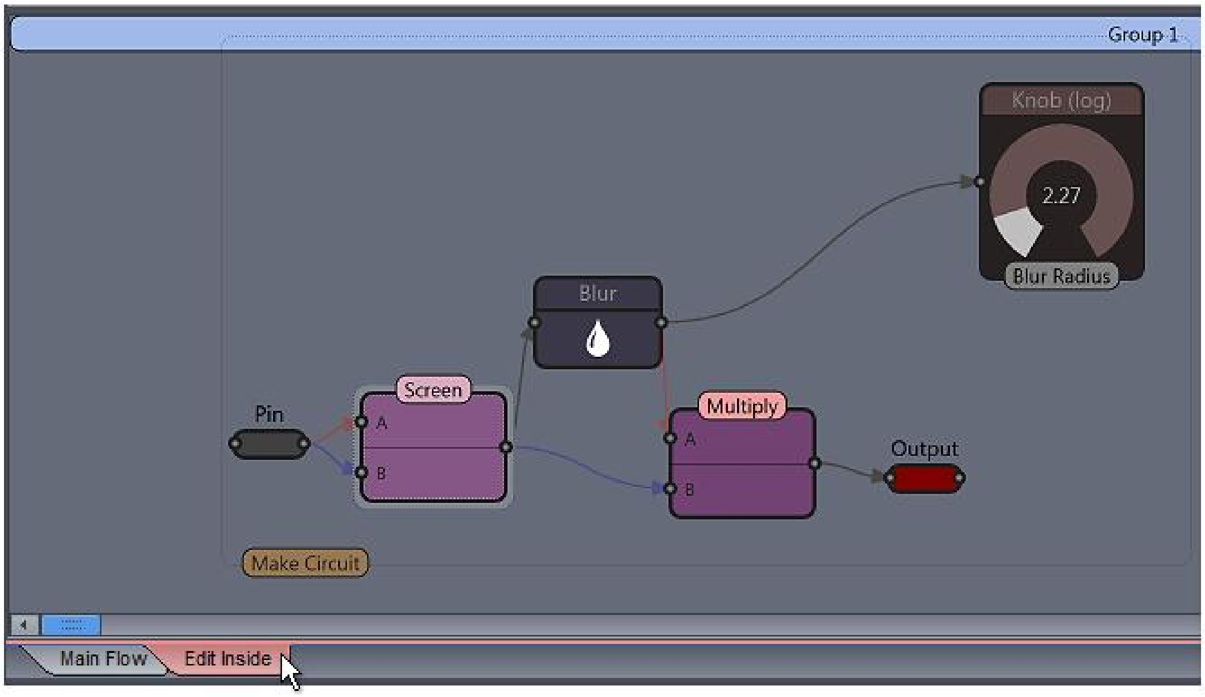


Well, that is cool, but we actually have created even bigger chunk of real estate than when we started!

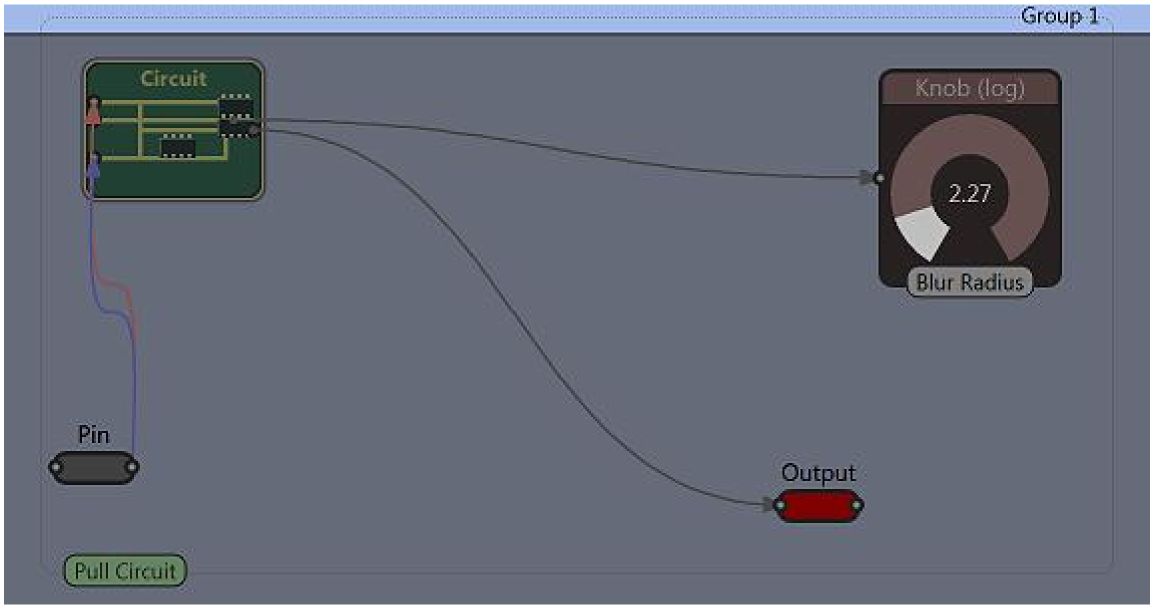
The Device is like a final group. Imagine you are electronics engineer (actually I am so I don't have to imagine anything) and you are making a device. At first it looks like a nest with various electronics attached to a test board and bunch of wires sticking everywhere and it takes half of your desk. But when it is debugged and it works fine, you want to compact it, solder the electronics on a circuit board and stuff it in the box, then put some knobs on the top plate to adjust few values that are worthy your attention. You don't expect to tinker with the insides any more, maybe only very rarely when things go wrong.

This is shockingly the same thing here.

We are going to compact the objects inside to make a small box, because nobody wants to have box that takes half of his desk.
If you paid any attention, to edit a group (and device is still a group), you use **Edit Inside** tab. So select the device and click on the red Edit Inside tab on the bottom.

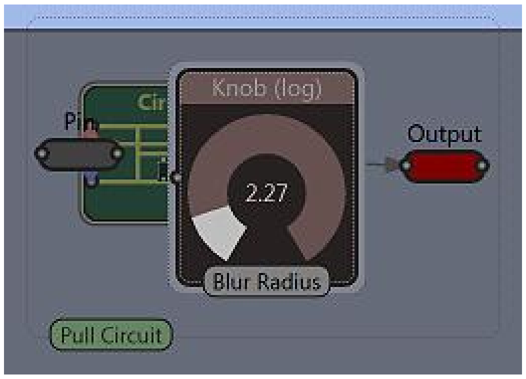


Now you are inside the box. You may simply move the objects on top of each other which is like making a tight ball from the electronics and wires (it does happens more often than you think), but a better way is to click on the small plate that says make Circuit.



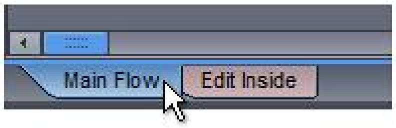
Really? Yes, really. I did indeed spend a considerable time to program a tiny little circuit board there and it does even change with different objects. Why? I don't really know but it may to do something with the fact that I am a geek. At least partial. The first version had a little box with just a circuit written on it, but how could I leave it like that? Now I have a c++ class to draw a fake circuit board.
Let's stop revealing trade secrets and let's go back to our device box. We have a neat little circuit and now we can make the box nice and small by shuffling the stuff around.

I'll go for this:

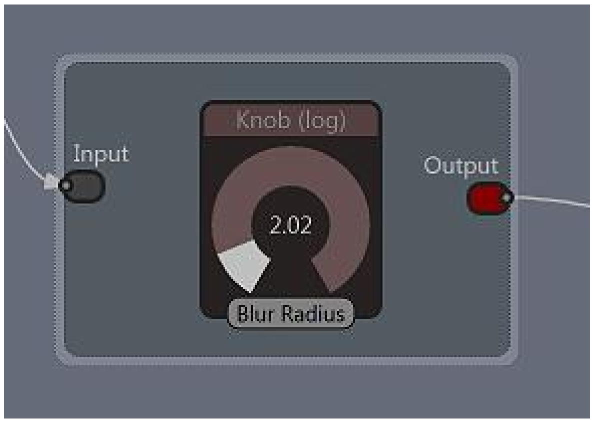


As you may guess, you can still pull the objects from the circuit if you want to change anything (but device should be more or less a final product)

Now click on the Main Flow Tab to go back to the main flow - or back to the real world after putting lid on the box.

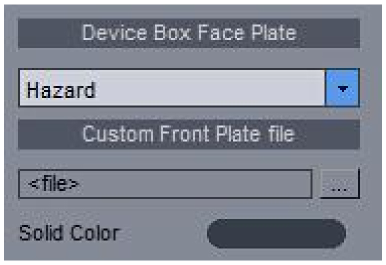


What we have here is a small-ish box with a knob and two pins. I still didn't name the input pin so I will do it now.

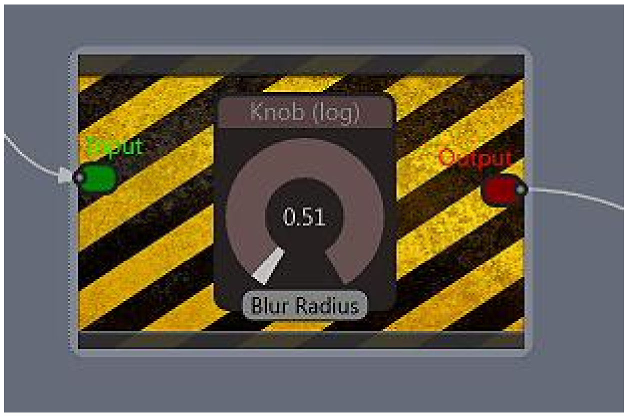


And that is our device.
It is plain and in the modern days of iSkins it does look a bit sad and unwanted.

If you click on the device box to select it and show its properties you will see that we have cure for that as well. I can indeed customize the face plate, even with my own design if I want.

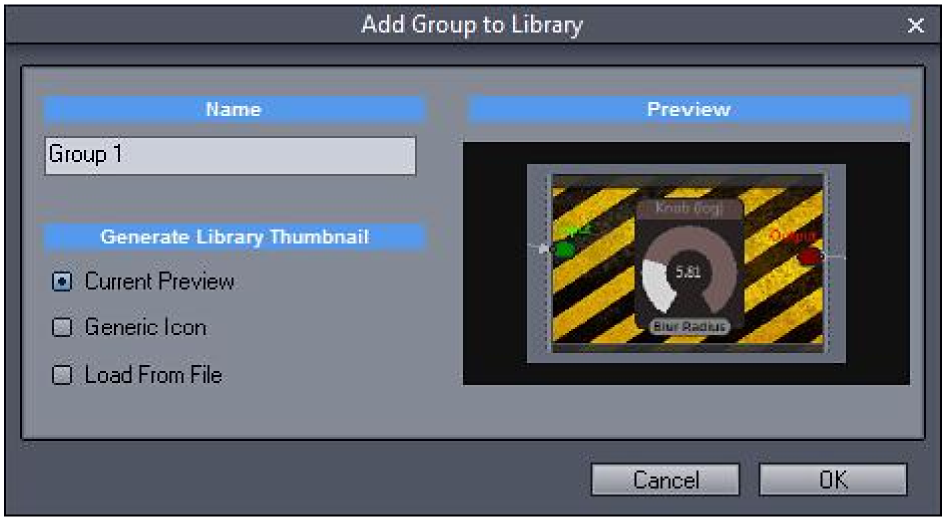


And this is how I imagine my special device to be:



So what have we learned so far?
- we can wrap a group into a device that hides everything except helper objects such as Pins, Binding Objects, Monitors, Text labels etc.
- we can shuffle things inside to make it nice and small and place the items onto a totally unrelated circuit board that Oscar spent a whole day creating in c++.
- we can customize the face plate
- it is still able to minimize
- we can still edit the inside if we need to

So what to do after all this charade? Save our masterpiece for future generations!
Right click on your shiny device and select Add To Library

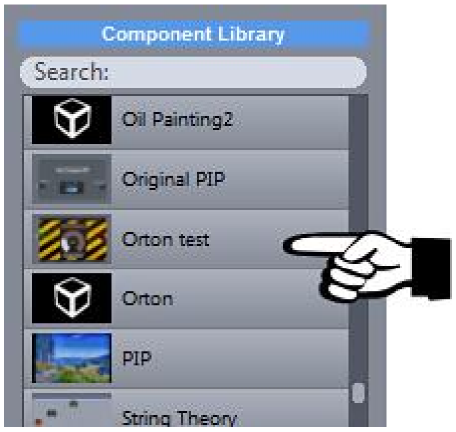


Name: Group 1

Generate Library Thumbnail: [✓ Current Preview] Generic Icon Load From File

This gives you some chance to rename it and add a preview icon (this is what will appear in the User Library list).

And here we have it:



Tips and Tricks

Move Locked Objects
Visible objects in the Devices are locked - you cannot move them with the mouse. However you can reposition them using a keyboard, use CTRL+ arrow keys to move locked object

3.5 Effect Masks

AKA Group Mask Channel
So far we were applying effects to the whole image. There is a way to apply effect only to a part of the image. In general this concept is called masking but you may also call it selection.
Example in standard editor: draw a selection (rectangle or circle) on the image, invert it, apply feather. Now apply Desaturate and Blur. What this does is blur and desaturates the edges of the image, but the middle stays sharp and colorful.

We can do the same in Photo-Reactor with even better strategy.

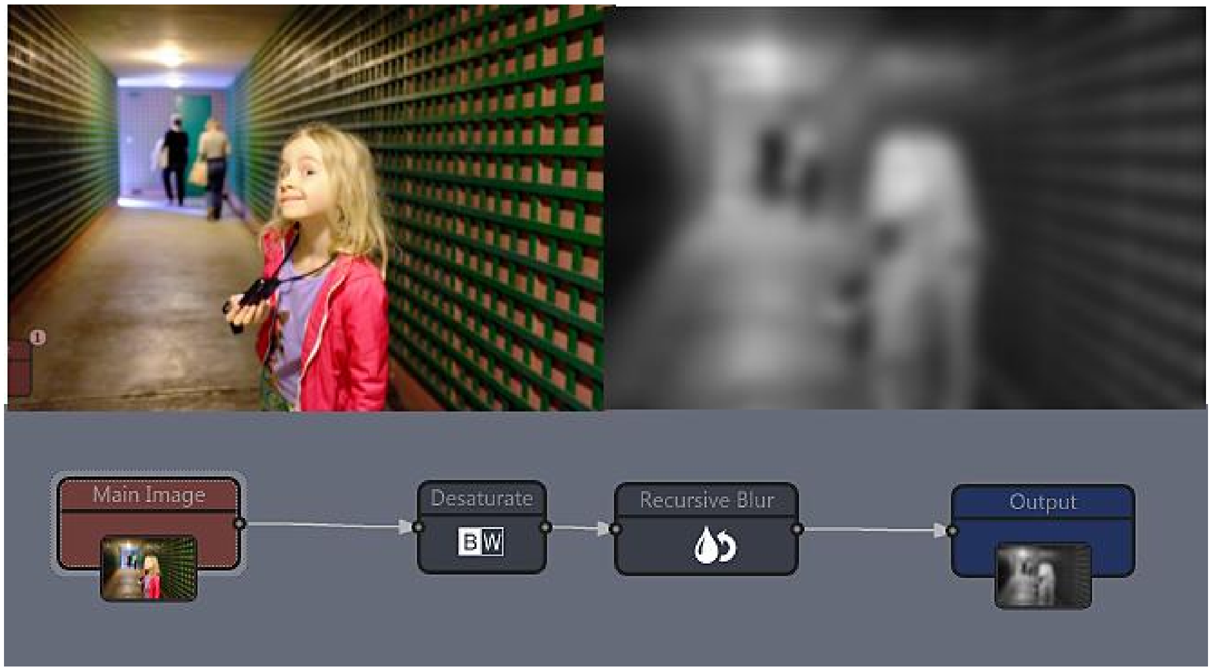
Add Group Mask Channel

The feature is called Group Mask Channel and as the name may suggest, it works by applying a mask to a group.
Besides the fact that it is dynamic (unlike a standard editor where selections are only temporary) you may soon realize how the concept of group masking makes it superior, and as photo editors go, rather unique.

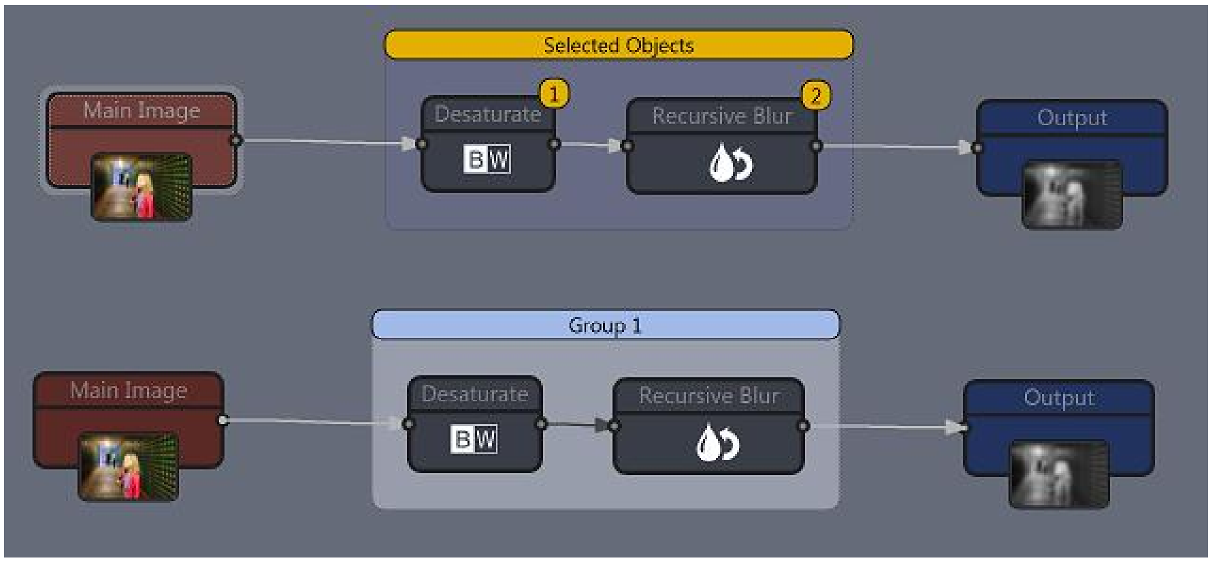
Let's just start with the example.

The effect is: apply desaturate and blur.
The mask is: apply to borders.

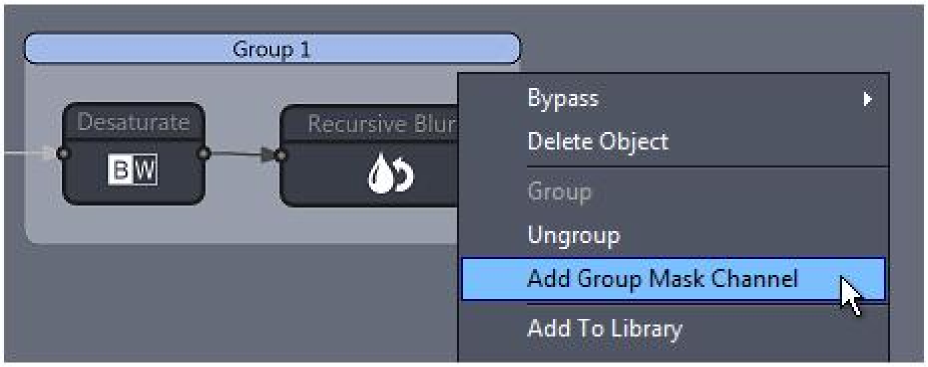
So first let's build our effect.



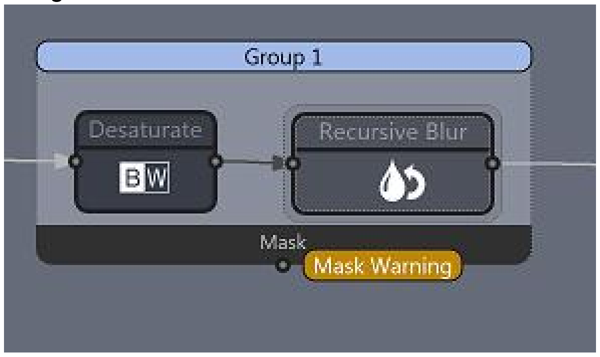
Now the fun part: select the two objects (draw a rectangle around them) and Group them. (Right click, Group)



Add Group Mask Channel
Right click on the group and select Add Group Mask Channel.

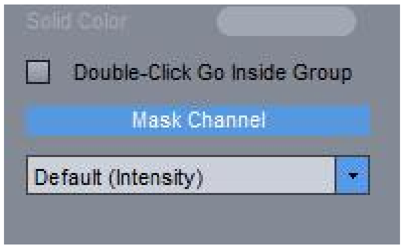


We get this:



The Group masks, unlike any other object in Reactor can stay unconnected and it will show only a warning. That is it will work as if there is no mask. Every other object if it has unconnected input node will show error.

Now whatever we connect to the Mask node will work as a mask. Masks are by definition a gray-scale images, but the group itself will take care of this. Reactor is not picky. Select the group and look at the properties.



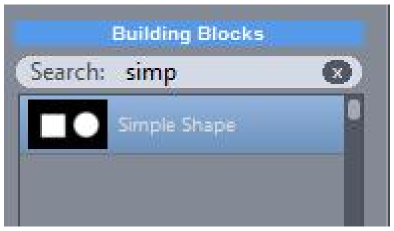
☐ Double-Click Go Inside Group

Mask Channel

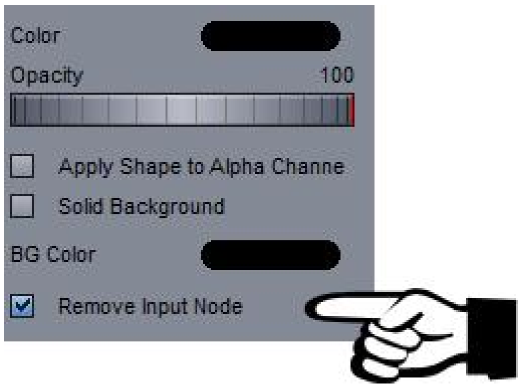
Default (Intensity)

There is a new control for Mask Channel that can switch from Intensity to Alpha Channel. Leave it on Intensity. That means whatever we connect to the mask channel will be first converted to a gray scale so we can easily plug in a color image. You can test it right now and connect the output of Main Image to the Mask Channel and you will get an interesting Orton/BW like effect.

But we want to stick to the plan.
Find Simple Shape object in Building blocks (start type "simple") and drag it to workspace somewhere under the group.

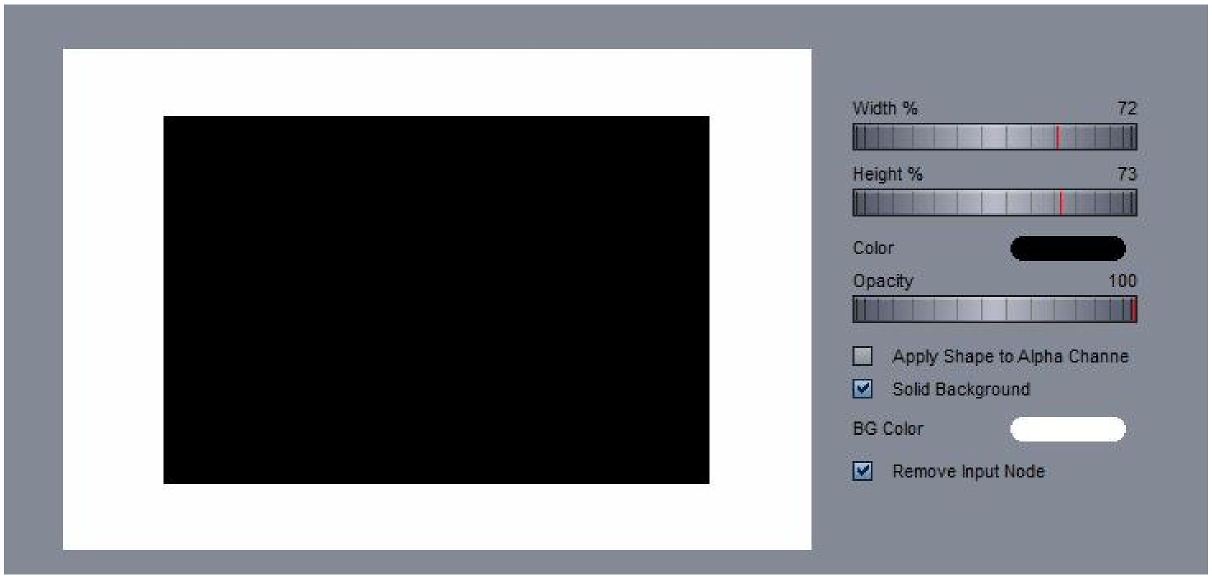


Simple Shape is one of those funky objects that can change its number of input nodes (from zero to one) so it can be used both as a starting "generator" type of object and also in the middle of an effect as an additive object.

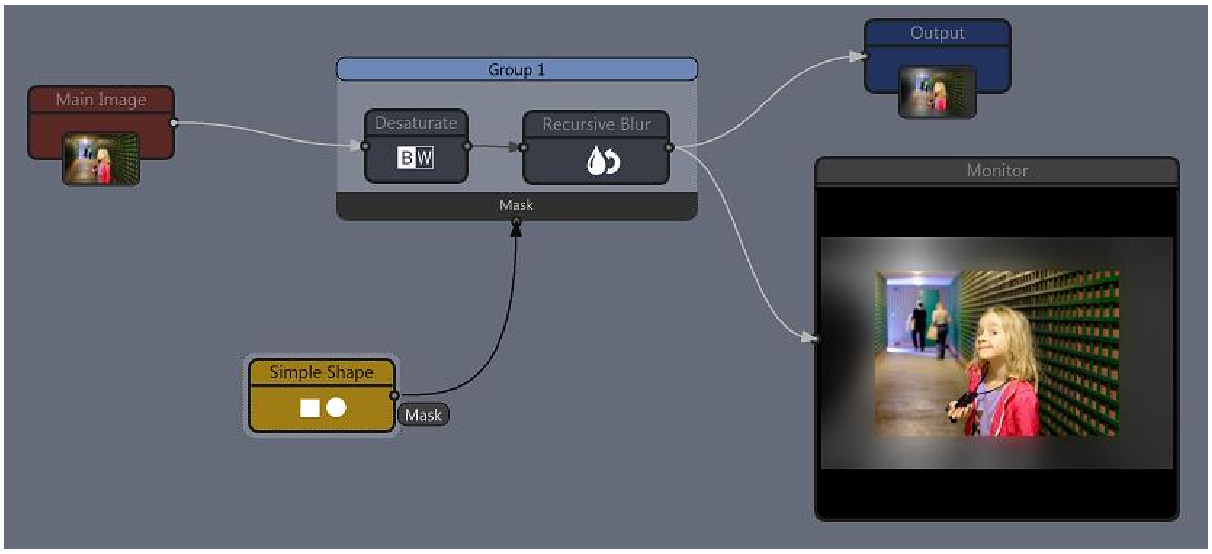

Color
Opacity 100
☐ Apply Shape to Alpha Channe
☐ Solid Background
BG Color
☑ Remove Input Node

We want to use it as a generator so it has no input, Click the Remove Input Node. If you forget this step, the object will remain in error as there is no input connected

Now we need to set some parameters so it is a black rectangle on a white background. Set Solid Background and change the color of background to white. Resize the rectangle with the Width and Height sliders.

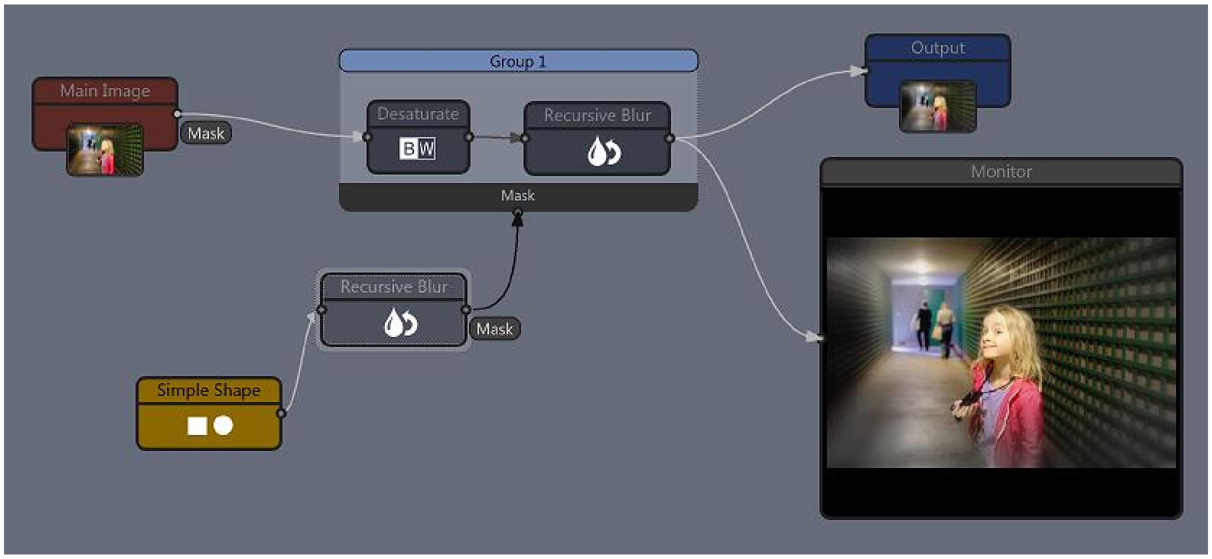

Width % 72
Height % 73
Color
Opacity 100
☐ Apply Shape to Alpha Channe
☑ Solid Background
BG Color
☑ Remove Input Node

Now we can connect it to the Mask Node of Our Group. As expected the effect (desaturate and blur) is now applied only where the mask is white and remain the same where the mask was black.



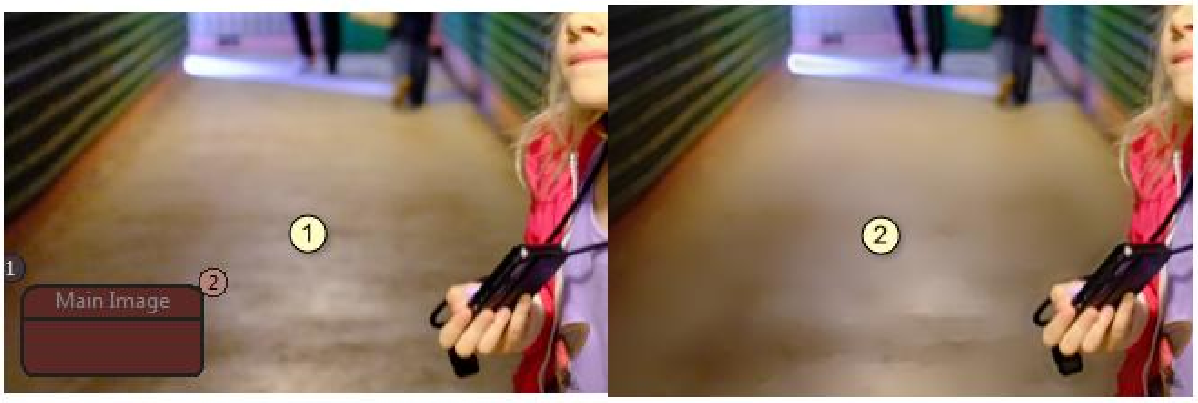
BTW: I added a **monitor** object from Building Blocks. Monitor can piggyback on any output node and display the current image at that node.

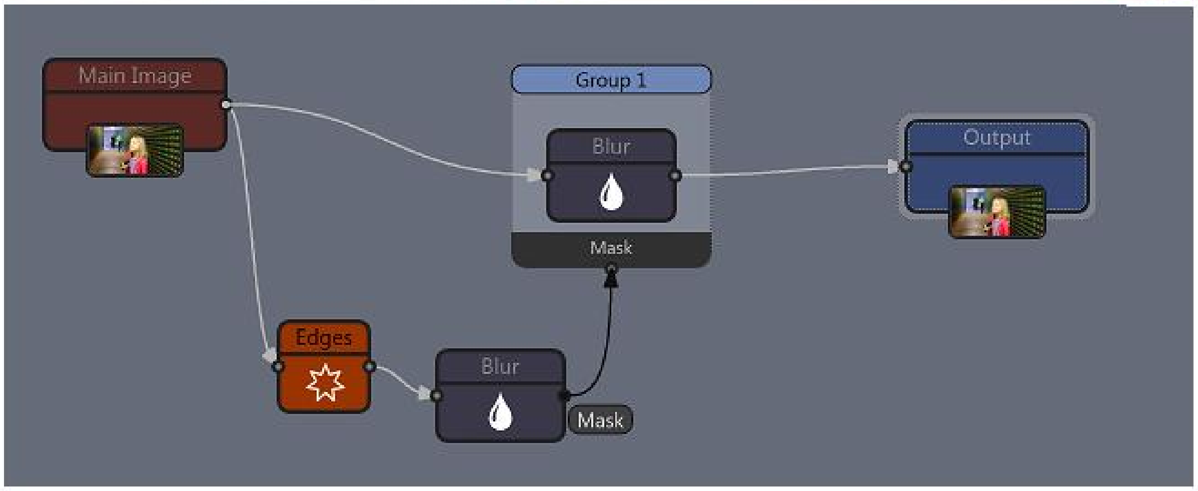
To blur or "feather" edges of our mask, all we need to do is to insert blur object to the mask line.
Now by changing the blur, the rectangle size or even its shape (ex: circle) we can fine-tune the effect.



Note:
The masks don't have to be just a shape.
Some image processing effect may use the modified image itself as the mask for various special effects.

In the quick example below, I am creating a simple edge preserving blur that blurs only large areas of image but keeps the edges intact.

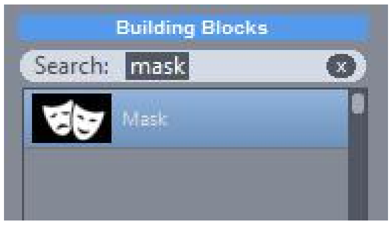

1
2
1
Main Image
2


Main Image
Group 1
Blur
Output
Mask
Edges
Blur
Mask

3.6 Manual Mask Object

An obvious question, after reading the Effect Mask chapter is, how to draw custom mask. And yes we can.
There is a mask object for that very purpose.
Note: It is important to note that Photo-Reactors strength is in the effects that can be then applied over and over. Once we draw a custom mask the effect become too specific to a single image.

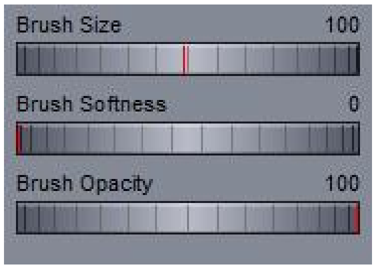
Find mask tool in the Building Blocks and drag it on to the workspace.



Select it and look in the properties window on the right side, there is Edit Mask Button.

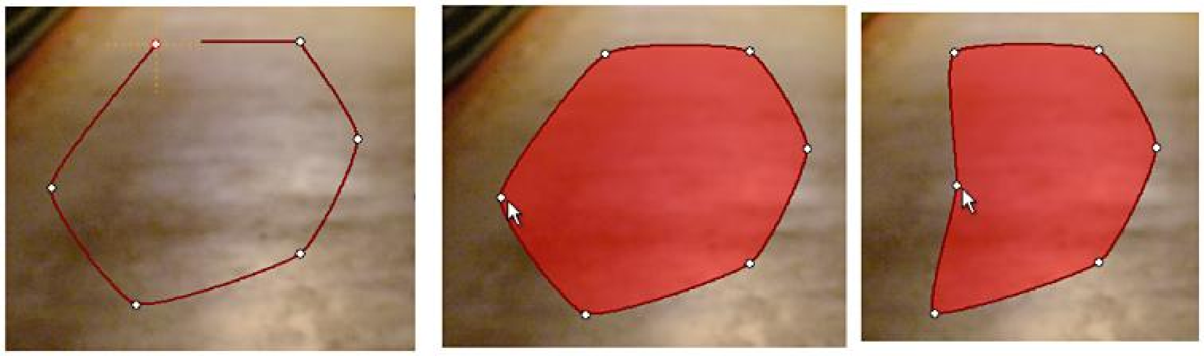
The mask is painted over the main image as a guide.
There are many tools, the most obvious, but less mechanical choices are Paint and Erase.

Paint and Erase works like a brush and it has also typical brush setting in the right panel



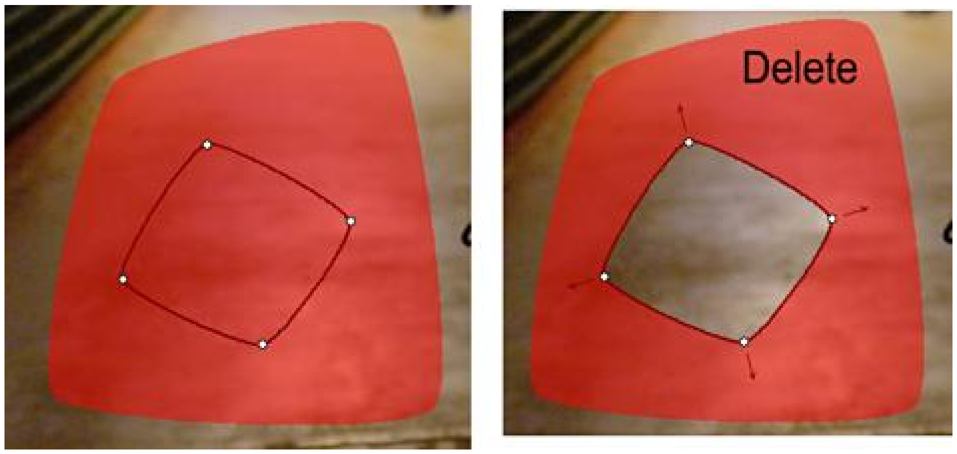
Brush Size 100
Brush Softness 0
Brush Opacity 100

Tip: You can erase without switching to Erase by holding SHIFT.
A more interesting may be the **curve** setting.



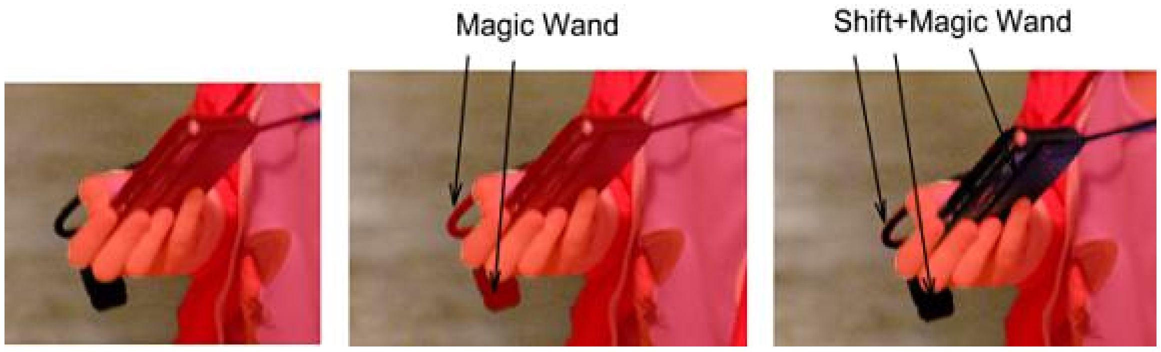
You simply place points that would create smooth curve until you return to the first point when the curve become a live mask. It is live because you can still adjust its points until you press Enter or start drawing a new curve.
There are few shortcuts:

ESC - cancels or removes the mask
Delete - during the drawing it will delete the last point, after the curve is create Delete will create hole in already painted selection.



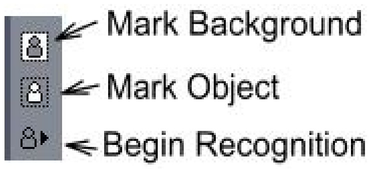
SHIFT + Click - when placing last point if you hold shift the mask will be drawn in inverse (same as if you press Delete after the curve is created)
Enter - apply mask

The next tools are magic wand and Mask from color. Magic wand is like a flood fill, it select colors that are similar in the neighborhood of where we click and Mask form color would select similar colors on the whole image.
The magic Wand is best used to add details to already painted mask
ESC will remove the last created mask by this method
SHIFT and click will instead of creating mask, remove the mask

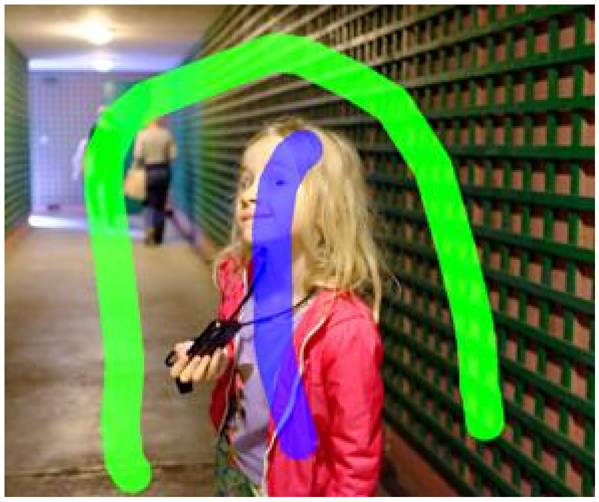


The automatic mask.
If you used our other software called Photo-Blend you know it has few great automatic masks. One of the masks is borrowed here, it is automatic masks with guides.

You paint a guide brush strokes to mark what is the background and what is the object and then let the software do the rest.



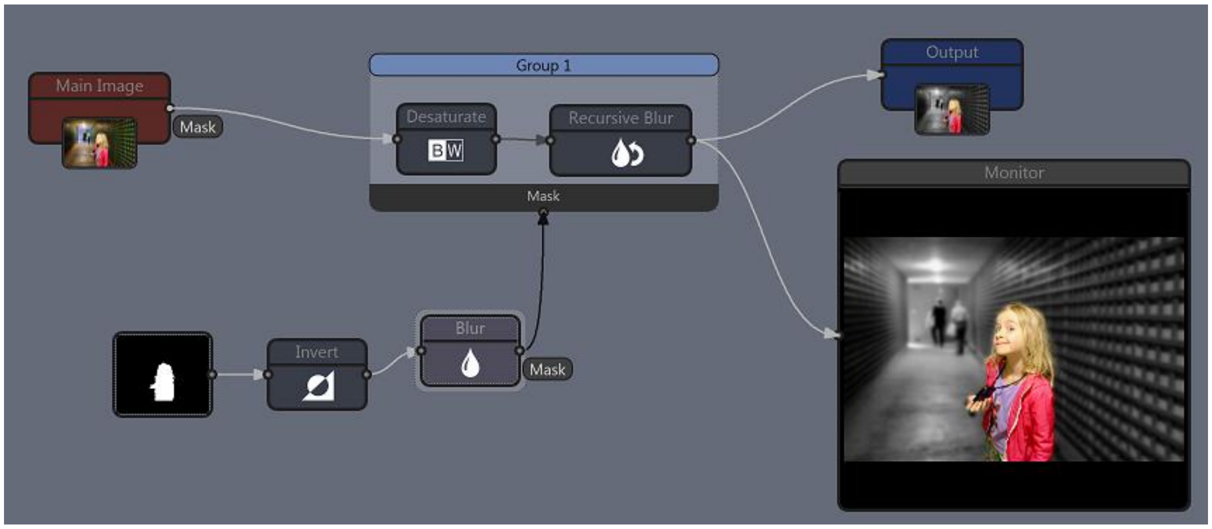
This is what we are going to use. With the top icon, we draw few GREEN strokes that mark background. With the Mark Object we do the same for the object we want to mask.



Now press Begin Recognition. The mask is created.
There are few trick:
If there are still large areas missing or there are false areas masked where we want background, just click the appropriate icon for background or object and touch up your guides where they are needed. Then press the Recognition button again.
Don't try to create perfect mask, it is often easier to touch up small parts with Brushes (Paint or Erase)

In our example I ended up with this mask as pictured above.

To get the desired effect we would need to invert the mask. I can Invert the mask at the Mask Editing level using the invert button but I don't do it as it is easier and more natural to work on the mask as it is (for touch ups etc). I will Invert it at the flow level by applying Invert Object.
Either way would produce the same result.
Here is the modified flow from the previous chapter.



Note:
If you are wondering why I am using different types of Blur, it doesn't matter that much, it is just blur like the Recursive Blurs can go really smudgy very fast (but it is not very precise), different blur like Stack Blur is limited to shorter distances and the Gaussian Blur (or Blur) is the most precise and best for blurring with the most control. It is more of a preference than a rule.

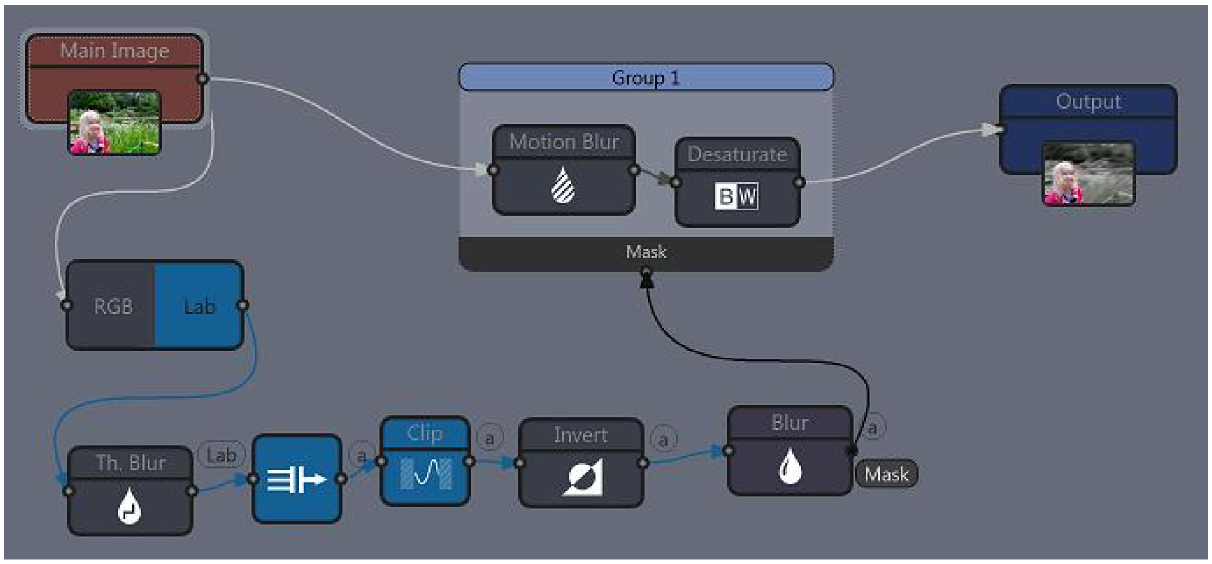
Embedded project and Mask
When you export JPG file a partial project signature is recorded in the jpg marker which allows you to extract the flow from the jpg file later. (it is like a soft save) The embedded project can be used as an archive of ideas or exchange with other users because the output file works as the the project.

However few things are not stored in the embedded JPG file and mask (the data, not the object) is one of them. An extracted project from embedded JPG file will have empty mask data.

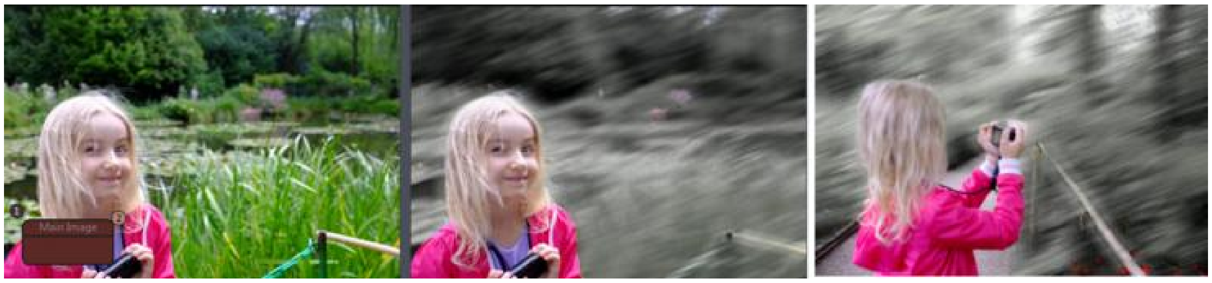
If you are using masks in your project and want to keep it then you need to save the project using the standard way "**Save Project**".

Tip: To create universal projects that can be applied to many images and batch try to create the masks algorithmically if you can (by using Color spaces, Hard Clip, Threshold with Blur etc on the main image)

Here is an example of using LAB color space to filter out green then using it to blur and desaturate background on outdoor pictures.



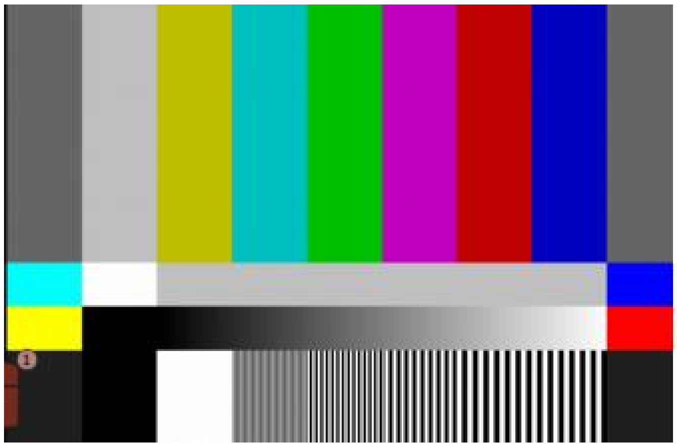
This flow would work on all similar images without the need of making the mask.

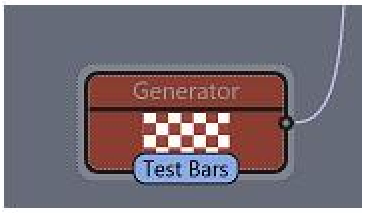


4 Even more Advanced

In this section we will look at couple of non-typical objects with advanced functionality.

4.1 Generator





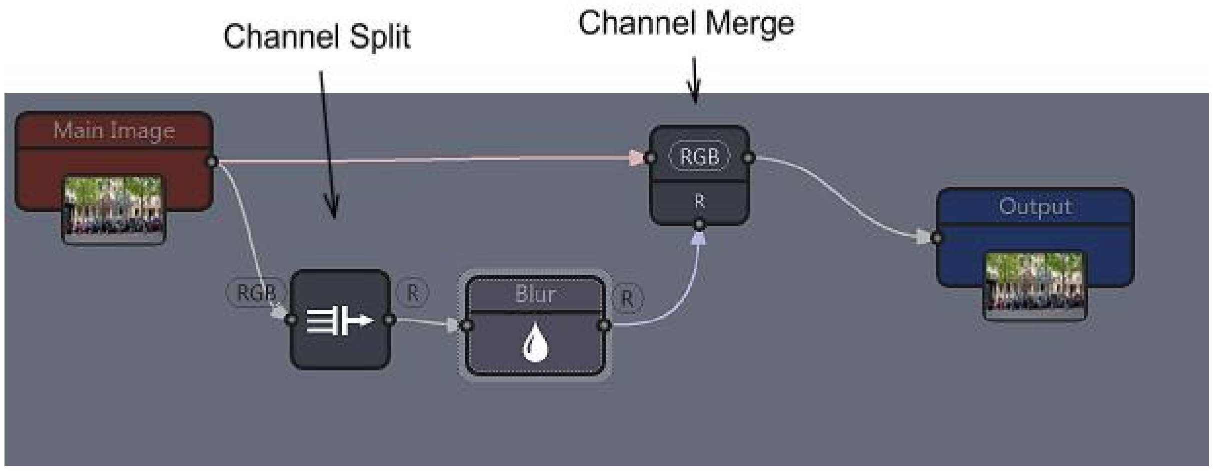
Generator is a special object that can create few checker board patterns and testing images. This can be greatly used for debugging the flow to see correctly how the process changes colors and details. Checker board patterns can be used to debug alpha image process.

There are other type of images Generator can generate - and those are all derived from the input image. That means we can use it in complex flkow as a "remote" Main Image.

4.2 Channel Split, Channel Merge

Sometimes, especially when we work in different color space we need to process only a single channel.

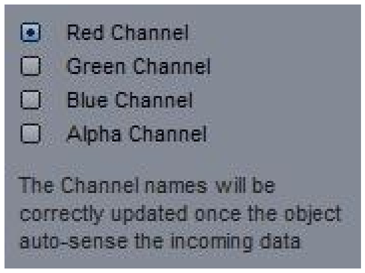
A typical example of such arrangement would be:



Object Crumbs:
ID112, 8, 1, 0, 228, 55, 0, P, 0.000
ID113, 31, 112, 0, 289, 121, 0, P
ID114, 12, 113, 0, 383, 117, 0, P, 2.500, 1.000
ID115, 32, 112, 114, 476, 44, 0, P, 1.000

In this example we are blurring only red channel.
Channel split/merge can be used for many things for example add alpha channel to image, to swap channels etc...

Let's look at the Channel Split properties

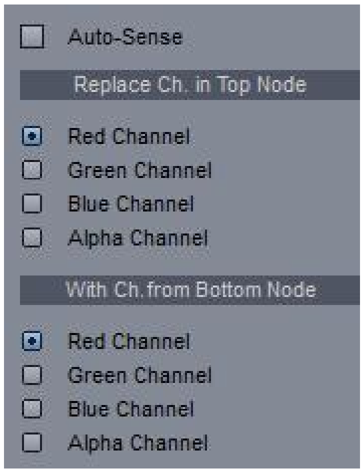


[✓ Red Channel]
Green Channel
Blue Channel
Alpha Channel

The Channel names will be correctly updated once the object auto-sense the incoming data

Here we set what channel will be at the output. The object is auto-sensing, that is it will correctly name the channels depending on what type of data are incoming. For example if the Incoming data are LAB color then we will see L and a & b channels.

Channel Merge is more complicated



☐ Auto-Sense

Replace Ch. in Top Node

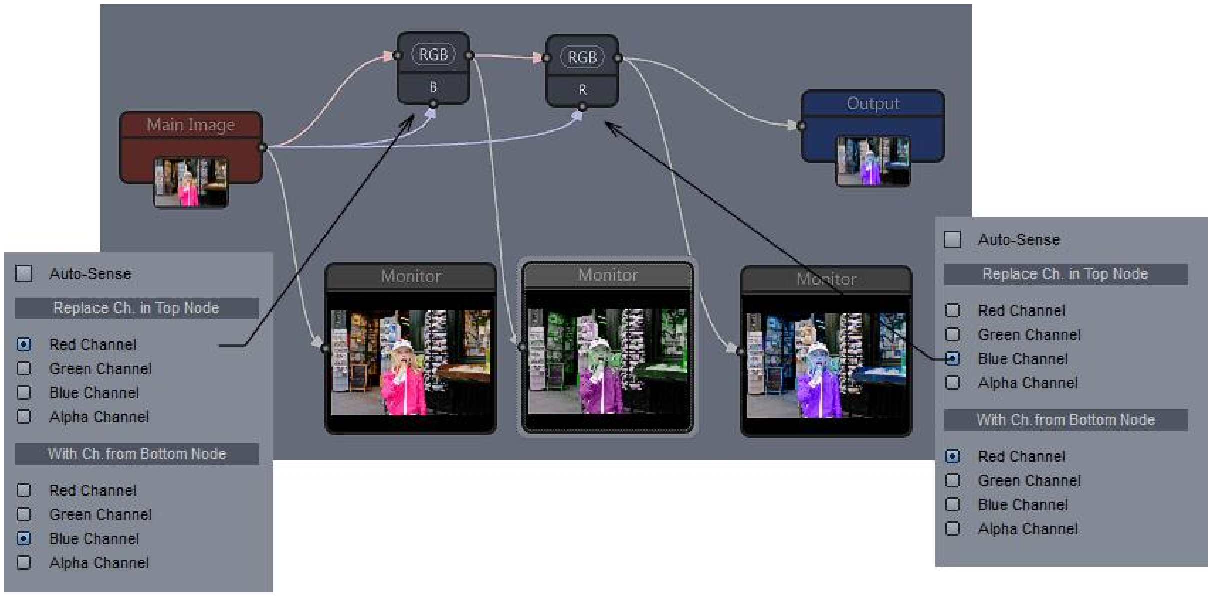
[✓ Red Channel]
☐ Green Channel
☐ Blue Channel
☐ Alpha Channel

With Ch.from Bottom Node

[✓ Red Channel]
☐ Green Channel
☐ Blue Channel
☐ Alpha Channel

The auto-sense can automatically merge the correct channels. If for example the incoming channel on the bottom node is alpha, then it will automatically replace the alpha in the top node with that one. In our case of blurring red channel it automatically replaces the red channel of the input node.

If we don't use auto-sense we can the mix channels as we want and use the merge object for various tricks.



☐ Auto-Sense

Replace Ch. in Top Node

[✓ Red Channel]
☐ Green Channel
☐ Blue Channel
☐ Alpha Channel

With Ch.from Bottom Node

☐ Red Channel
☐ Green Channel
[✓ Blue Channel]
☐ Alpha Channel

☐ Auto-Sense

Replace Ch. in Top Node

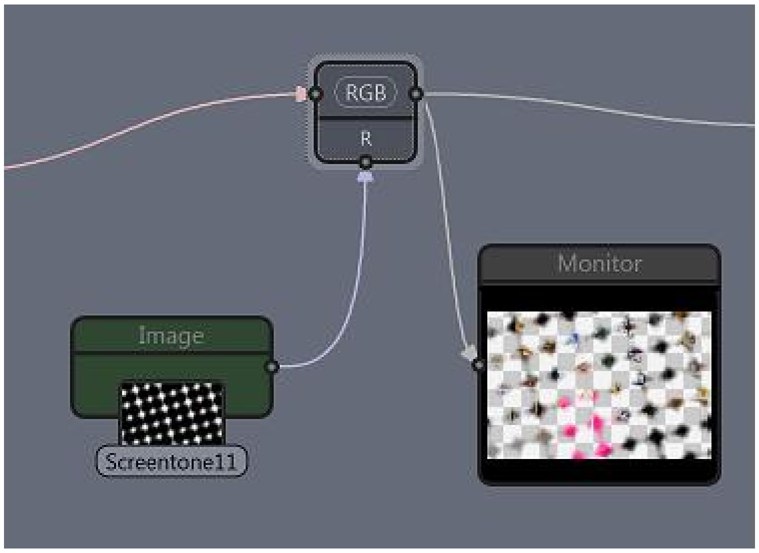
☐ Red Channel
☐ Green Channel
[✓ Blue Channel]
☐ Alpha Channel

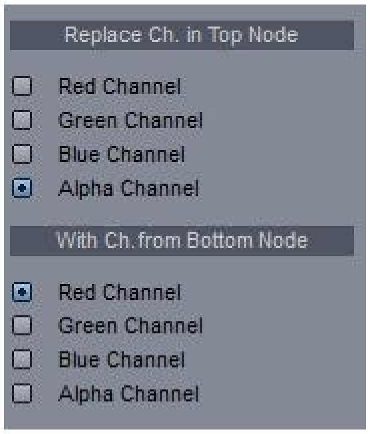
With Ch.from Bottom Node

[✓ Red Channel]
☐ Green Channel
☐ Blue Channel
☐ Alpha Channel

In this puzzling example we swapped Blue and Red channels with two merge objects. In the first object we replaced the Red channel with the Blue channel (all other channels stays as they were) and in the second object we replaced the Blue channel with the original red channel. (we cannot use the output node from the first object because the red channel has been replaced by blue channel - remember?).
It may get a bit to digest the above sample, but you have to remember the object doesn't expect that the second node is a single channel - it can be as in our case full RGB from input and we will then cherry-pick what channel we want to use in the properties.

In the next example we use a pattern and turn it into alpha channel.




Replace Ch. in Top Node

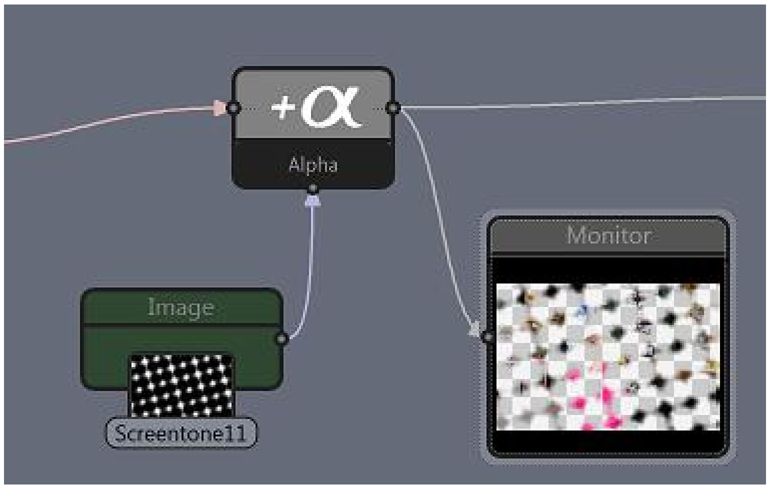
- [ ] Red Channel
- [ ] Green Channel
- [ ] Blue Channel
- [✓] Alpha Channel

With Ch. from Bottom Node

- [✓] Red Channel
- [ ] Green Channel
- [ ] Blue Channel
- [ ] Alpha Channel

It doesn't matter what channel we use from the pattern image since it is grayscale, all R,G or B are the same (we used Red channel here) and we are merging it as an alpha channel.

There is an equivalent object to the above example and it is called Alpha Merge.

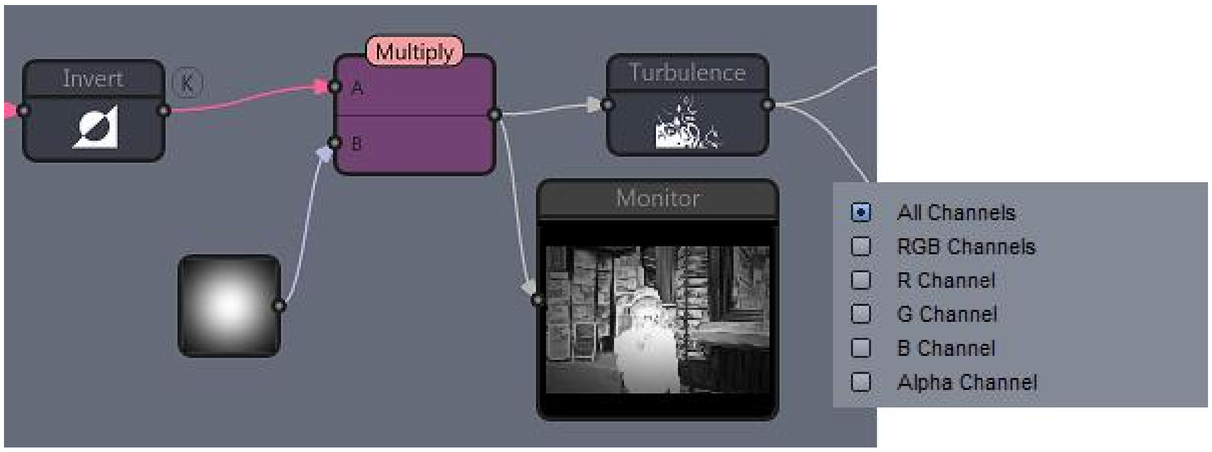


Alpha Merge doesn't let us choose what channel we going to replace - it is always ALPHA

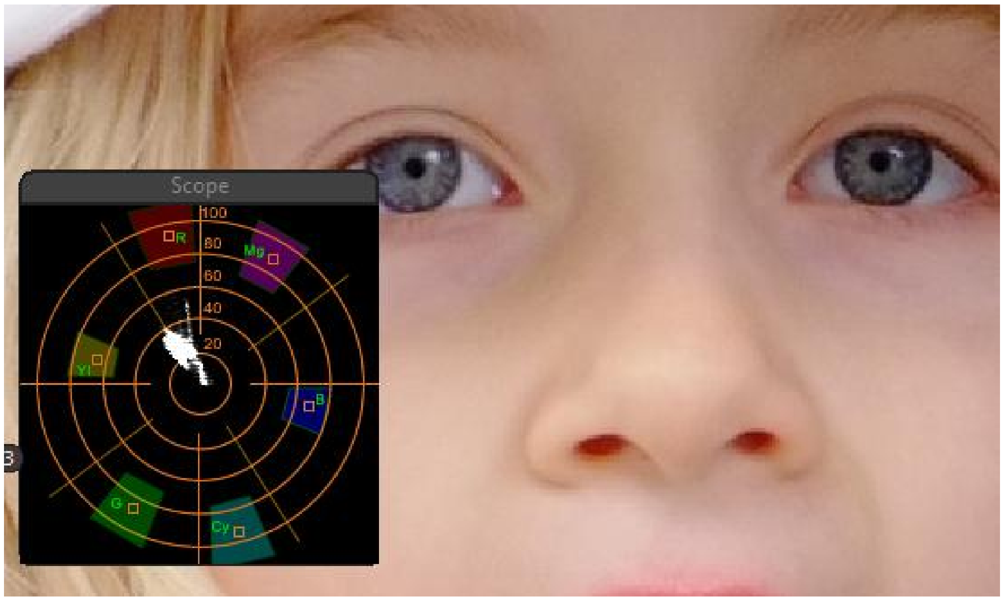
4.3 Monitor and Scopes

Flying by numbers.
Two objects the Monitor and Scope can be used to place a special device on the workspace to see how the data change at any particular points and do a special measurements.

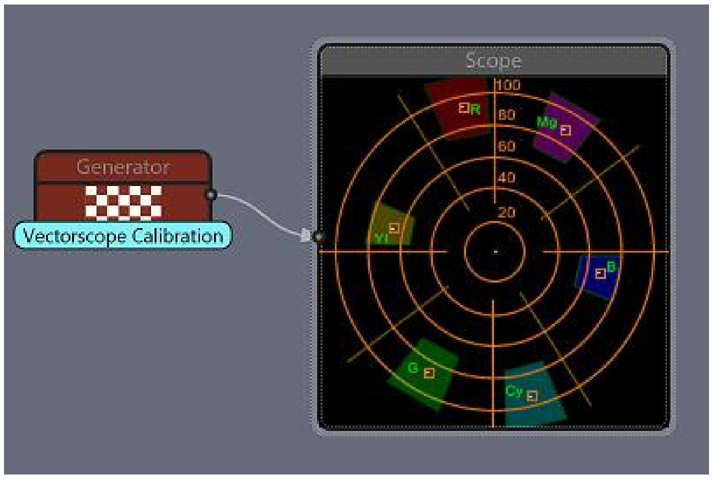
Monitor
Monitor displays data at particular point and we can select what channels we want to see. Double-clicking the monitor can enlarge it to bigger size.

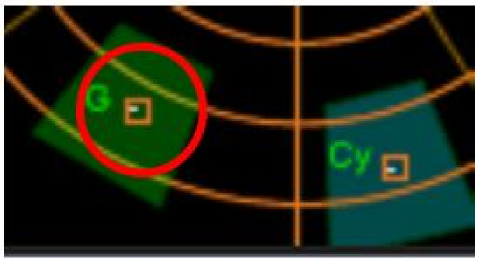


Vectorscope
Vectorscope is a special scope that shows saturation and hue of the pixels. It measures data from the current preview, that means **we can zoom** on some part of the image to get the measurements **just** from there.
It is often used to evaluate color of the skin. As we can see from the scope image below majority of the pixels falls around the 11 o'clock line which is called **flesh line**. The other thinner line that goes towards red (left of 12 o'clock) are likely the color of the lips. From the vectorscope it shows that the image is well graded and correctly balanced for skin tones. (Fuji X100, prized for its accurate skin tones)
Tip: if you can't see the angle clearly, you may click Multiply (Cr, Cb) 2x. This will "oversaturate" the colors so they show more further from the center and allow us to see the angle better.



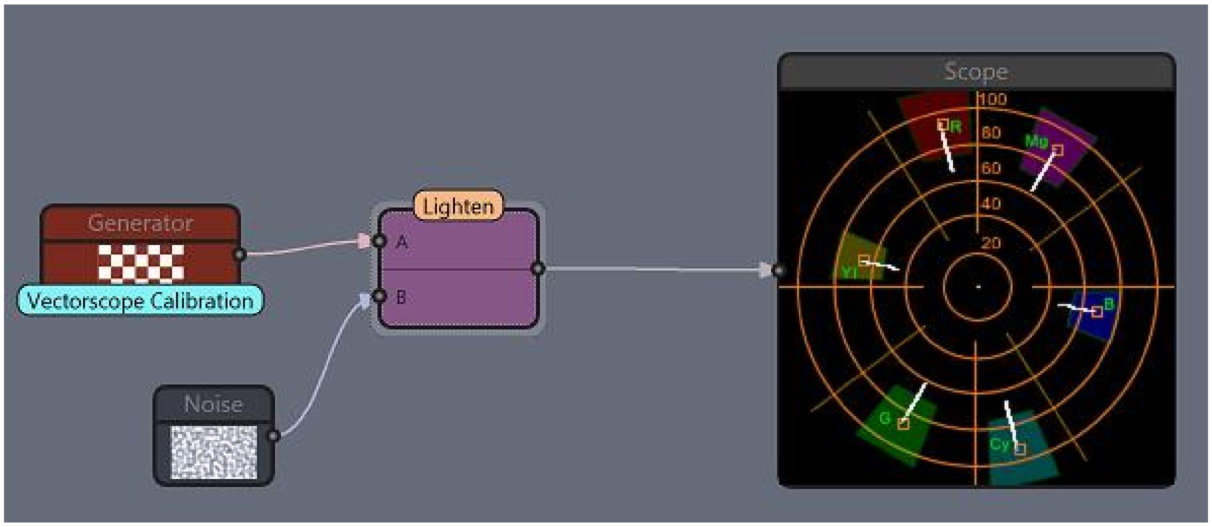
Vectorcope calibration
By connecting vectorscope and generator while setting it to **Vectorscope Calibration**, (also works in Test Bars, but they have few more color patches) we can observe the calibrations points. The vertical bars appear as a **tiny dots** and they have to fall withing the small rectangles of the color patches on the vectorscope.



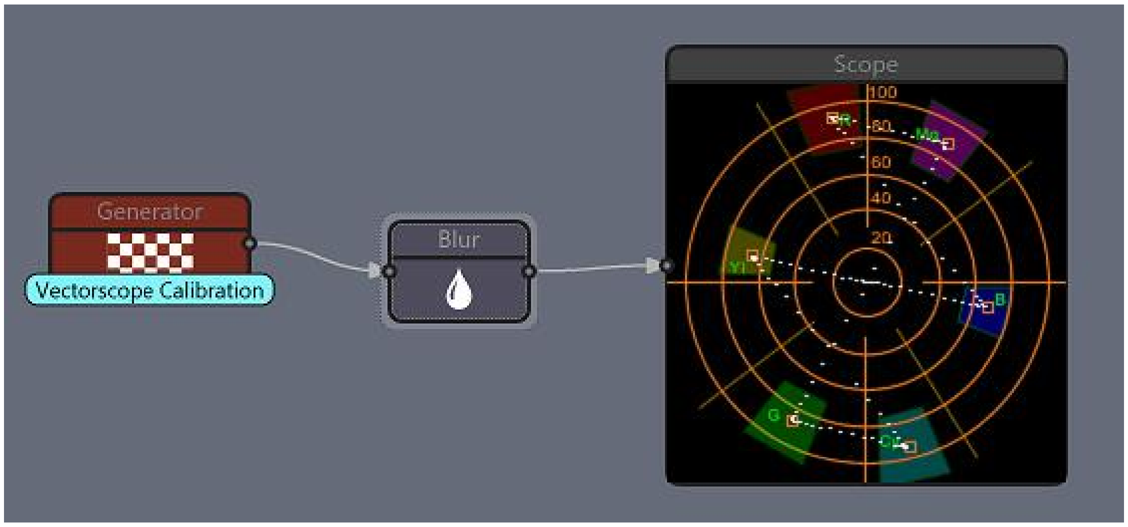


The dots are tiny because they represent a very exact color. If the color of the bars vary or there is some

addition of the noise the dots would appear bigger or as line.
In this test we add a noise component which affect the saturation. (we toned it down with the opacity slider to have just very little noise added) See how the dots changed to lines. The noise did not change the hue of the bars (the dots would appear thicker).

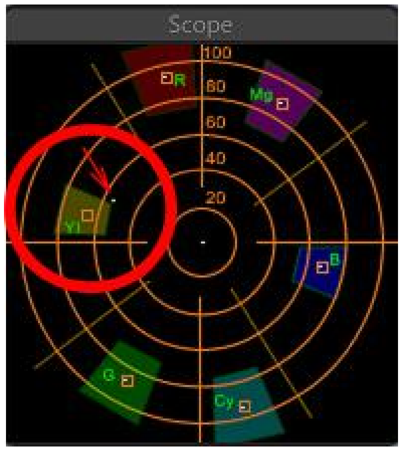


In the next test we add blur. Notice how addition of a large blur changes the image. The lines we see are the blurred transitions between one bar to another bar.

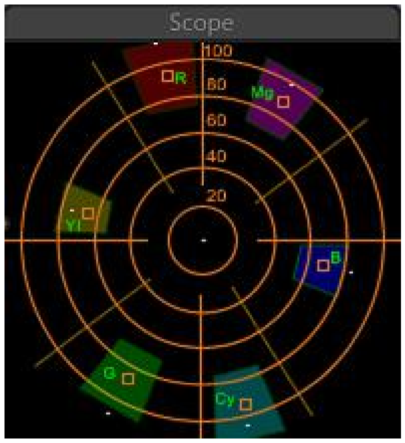


Note: This way we can test how our flow changes the colors by plugging generator instead of the input and monitoring the vectorscope at the output. We will see potential problems (for example overly saturating red and yellows which would mean red faces when used on portraits)

Example: By observing the vectroscope image below we may see that there is some non-linear hue shift in our flow. While all points fall on their targets, the Yellow on the left side (9 o'clock) is shifted towards reds and bit desaturated. This gives us very precise answer as what the flow does in regards of hue.



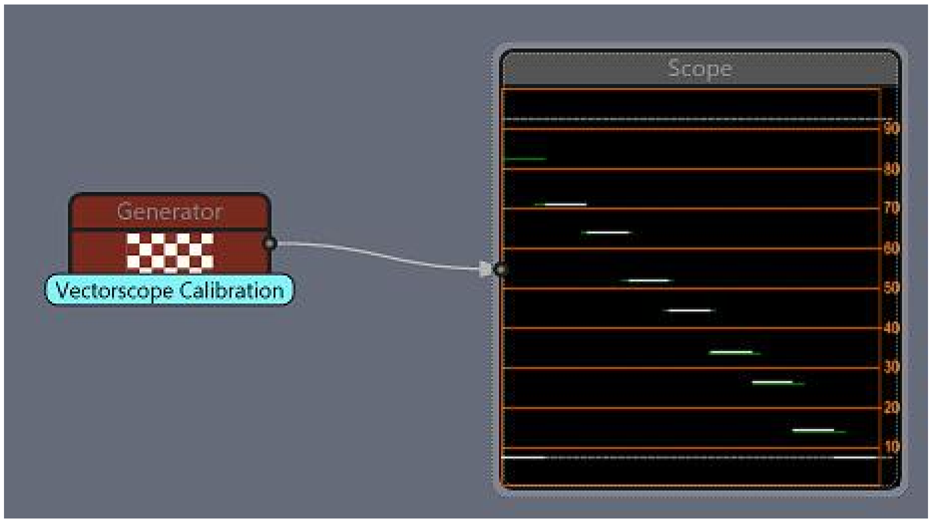
Example 2: here we see that while angle of the test bars are correct everywhere they are further from the center then should. That means they are over saturated. We can also see that the greens and reds are further than blue, cyan, magenta or yellows giving us yet more information.



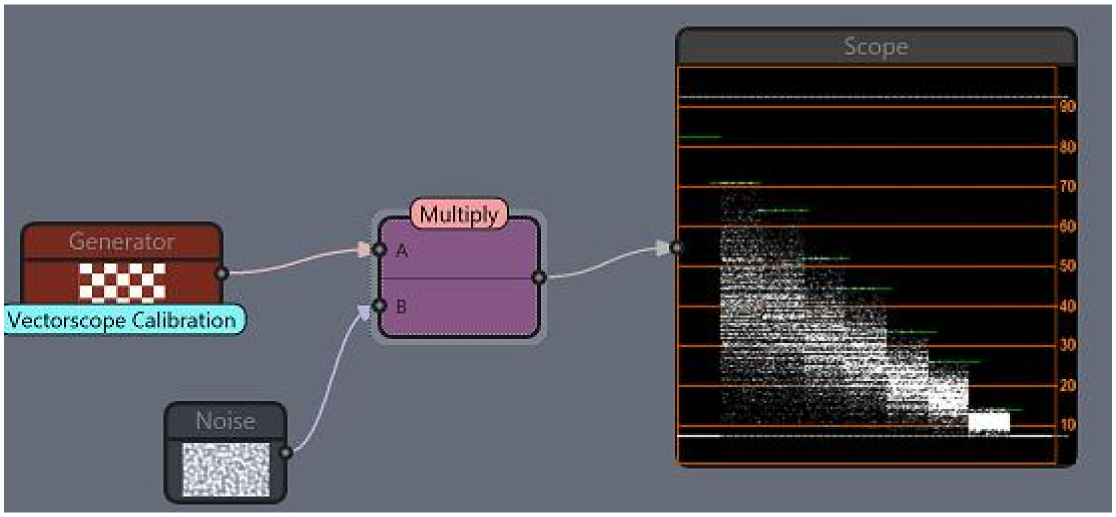
In both cases we can see that our effect changes the hue of the image and we can take some compensation measurements if needed. (for example in the second case, we may desaturate the results to get the targets closer.

Learning how to read vectorscope is very useful for color grading images.

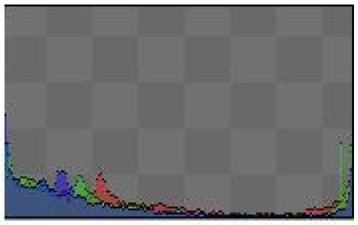
Waveform
Waveform displays IRE intensity of the image along the x axis. Bright objects will produce waveforms towards the top of the monitor and dark objects will produce waveforms toward the bottom. Notice how the test bars are specially chosen so they each produce equally less intensity from left to right.

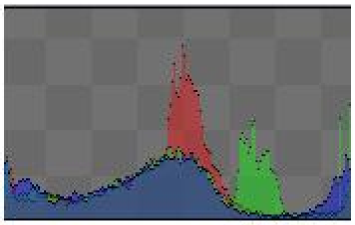


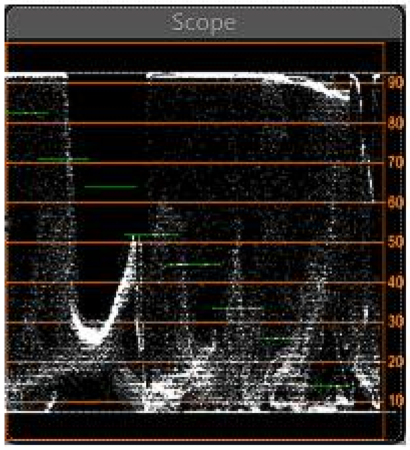
Notice how addition of the noise will spread the otherwise sharp waveform of the calibration pattern. Try changing the noise type and you will see different pattern.

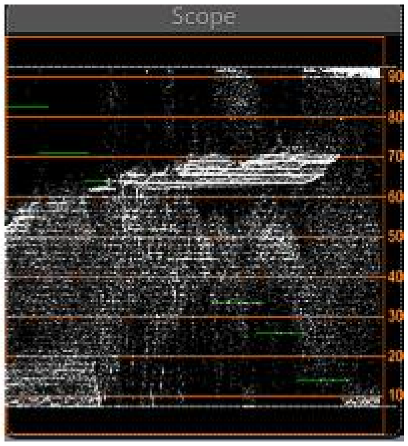


How to use Waveform?
Waveform can greatly help us to evaluate amount of clipping in the image with much better accuracy than histogram. From the waveform below on the left we can see large clipping lines on the top part of the monitor and also accumulation and clipping of pixels on the dark side (bottom part of the monitor) while the right sample looks much more evenly spread, with only a little shadow and highlight clipping. However on the histograms they both look like they may have the same amount of clipping from both dark and light side!



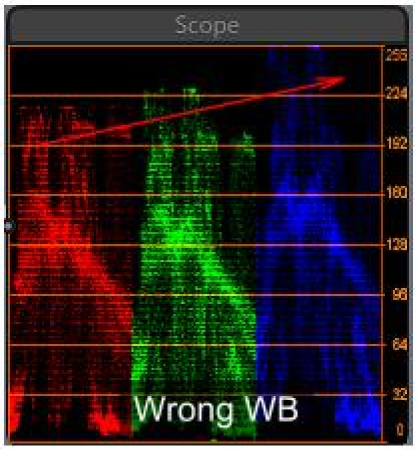


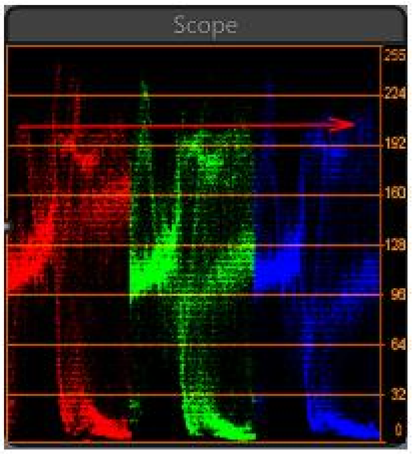




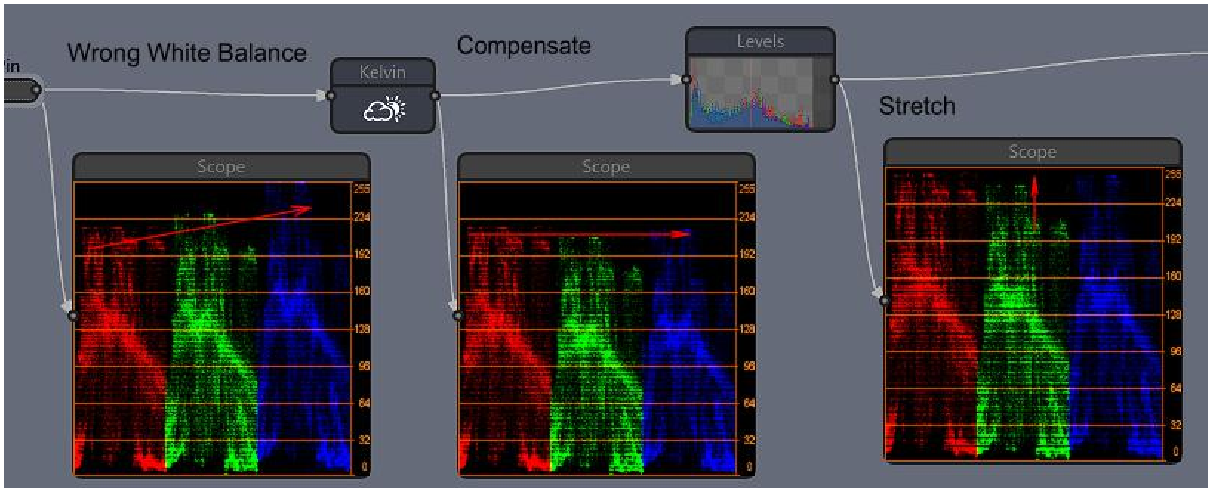
RGB Parade
Displays levels of red, green, blue channel one after another. Similar to the waveform, but allows us to evaluate channels in relation to each other.

RGB parade can be a **fantastic help** in determining the white balance. On the sample below, on the left we see that the levels of R,G,B channels are not equal, the blue is higher than both red and green and the green is higher than the red. This is consistent with the **wrong color balance** giving us a "blue cast". The image on the right has the RGB parade formation even (we are looking at the whole pattern shape) giving us a good confidence in its white balance.



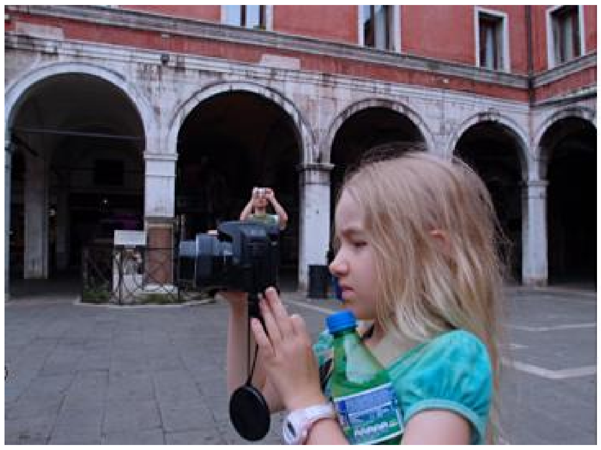


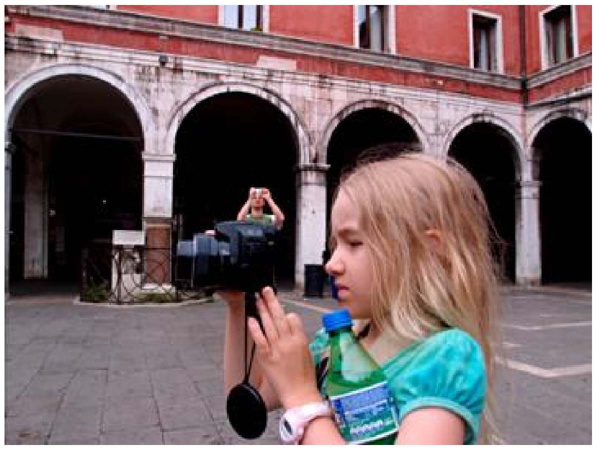
To correct the white balance, we can use the Temperature Correction (Kelvin) object. In this case we would need to choose lower K number than the default 6500K to compensate for the existing blue cast. Also doing so we can see that the levels of all three channels equalized but they don't stretch the whole height anymore, which means we should add Levels object at the end to stretch the parade to its full height moving the highlights slider lower.



See; we didn't even had to look at the image itself, we completely evaluated and fixed the image by looking at the gauges.

Difference between image with a WB cast and the corrected image:

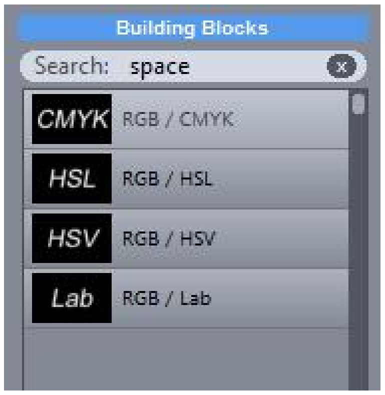




4.4 Color Space

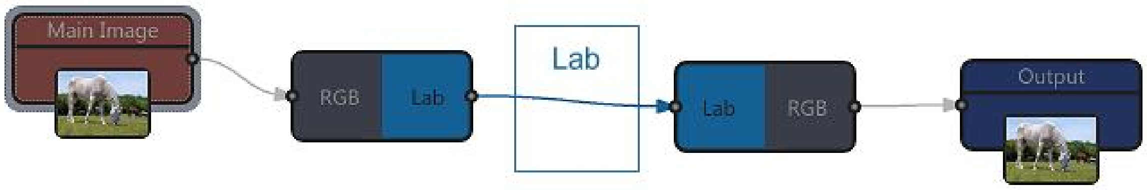
Working in different color space may give us different way of adjusting the image.

To start working in different color space we need to drop in one of the color space conversion object



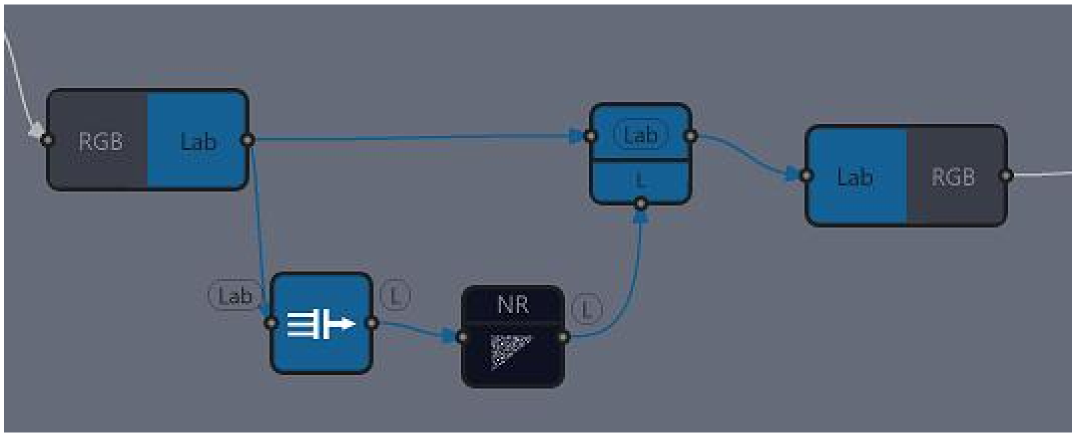
The color space object are auto-sensing and would orientate themselves to correctly do the RGB to color space conversion.
In general we would drop one of the space conversion object (RGB / Lab for example) on the line, then drop another to flip it back to RGB.
What we have in between (within the blue line) is the Lab color space.

We can use any of the objects on the blue line same way we would do it for RGB, however...

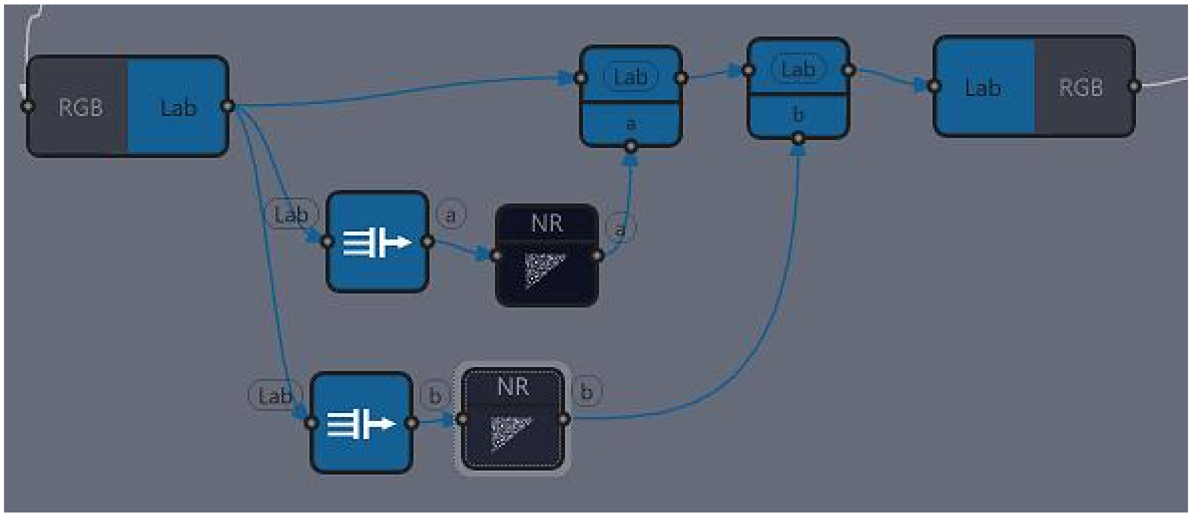


...it is not always straightforward to know what an effect would do when applied on all the channels at once in nonRGB space. For example try to imagine dropping Desaturate inside the Lab line where one channel of Lab is lightness and two channels are color info. It won't desaturate the image, a m least not in the sense we know it does for RGB image. It doesn't make it very useful to work on different space this way.

The channel split and merge functions are therefore very important as we can separate the exact channel we want to change and process each separately.
In the following example we are de-noising the Lightness channel by turning the RGB to Lab colors, separating the L channel, de-noising it and then merging back with the Lab main line. After that we are turning it back to RGB.



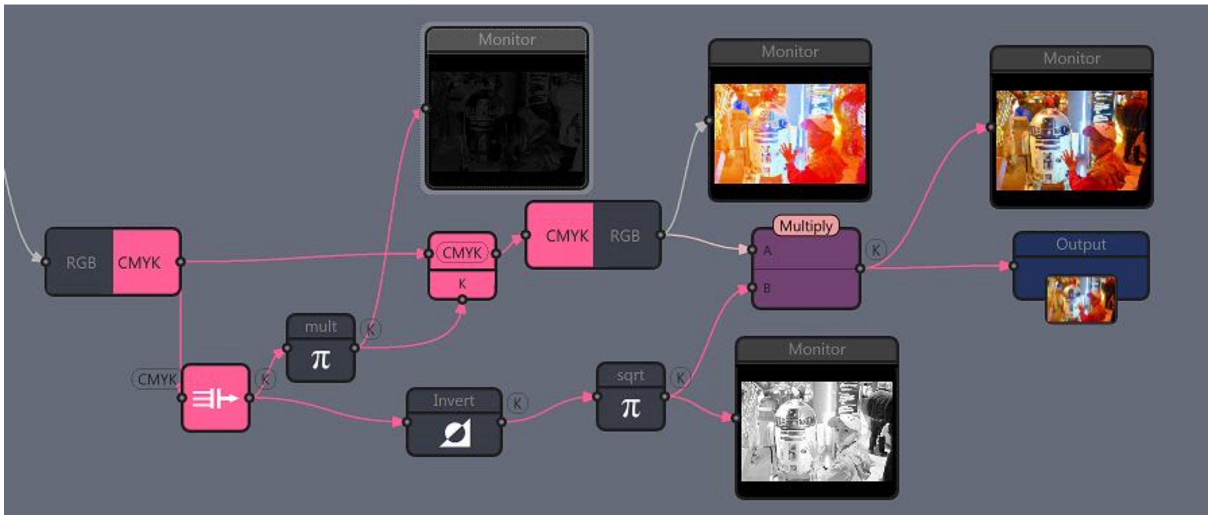
In the previous example we only reduced luminance noise and we didn't touch the color.
Similarly we may de-noise only the color and leave the luminance intact (for example if our image has chroma noise but very little luma noise)



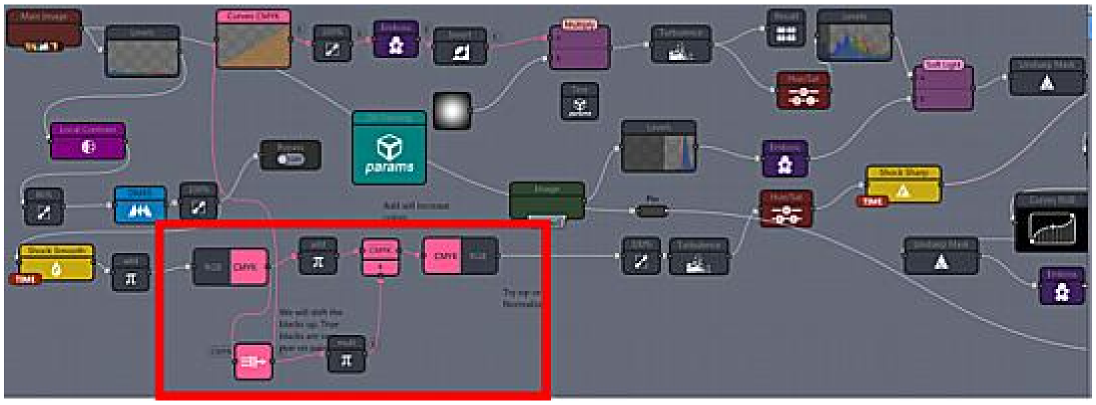
There is nothing preventing us to mix various color spaces together, although the usefulness of it is is questionable if it is just an random act. But who knows, it may produce some cool effect.

Also there is no rule that we have to always turn all of the channels back to RGB space - if we know what we are doing.

In the example below I did remove blacks from CMYK (using multiply and dragging it nearly to zero) yet adding the K channel later after CMYK-RGB conversion to the RGB image using Blend layer.


Monitor
Monitor
Monitor
RGB
CMYK
CMYK
K
CMYK
RGB
Multiply
A
B
K
Output
mult
π
K
CMYK
K
Invert
K
sqrt
π
K
Monitor

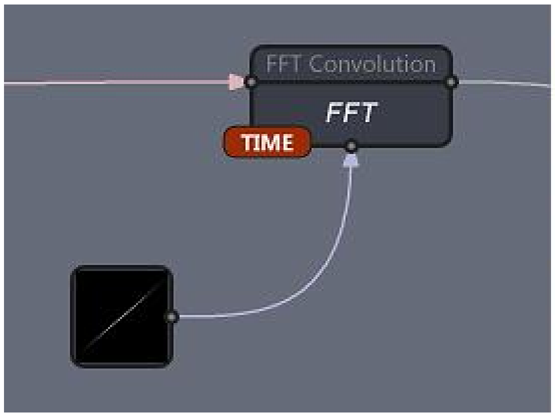
The example above doesn't seems that useful, because we got basically the image we started with, but it is in fact a part of much larger effect where I would process the color-minus-black image very different way than the K channel then merging them later in the flow getting an interesting effect.



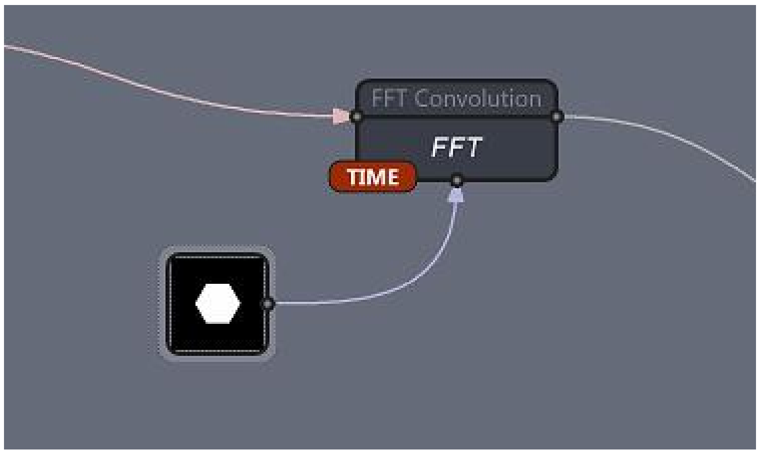
The CMY-K separation was used as a part of much larger flow to produce this effect:

4.5 FFT Convolution

FFT convolution is used to affect an image using a kernel. In practical terms this refer to blurring.
The kernel can be any bw image but for a normal purpose it is always a determined shape. For example blurring image with a thin and long ellipse kernel will produce something similar to a motion blur.

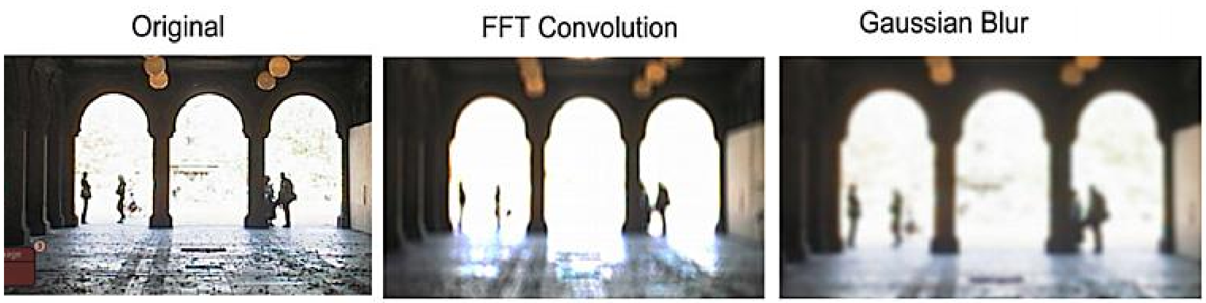


Using a kernel that resembles the lens aperture we can blur the image similar way a lens would de-focus a photography.

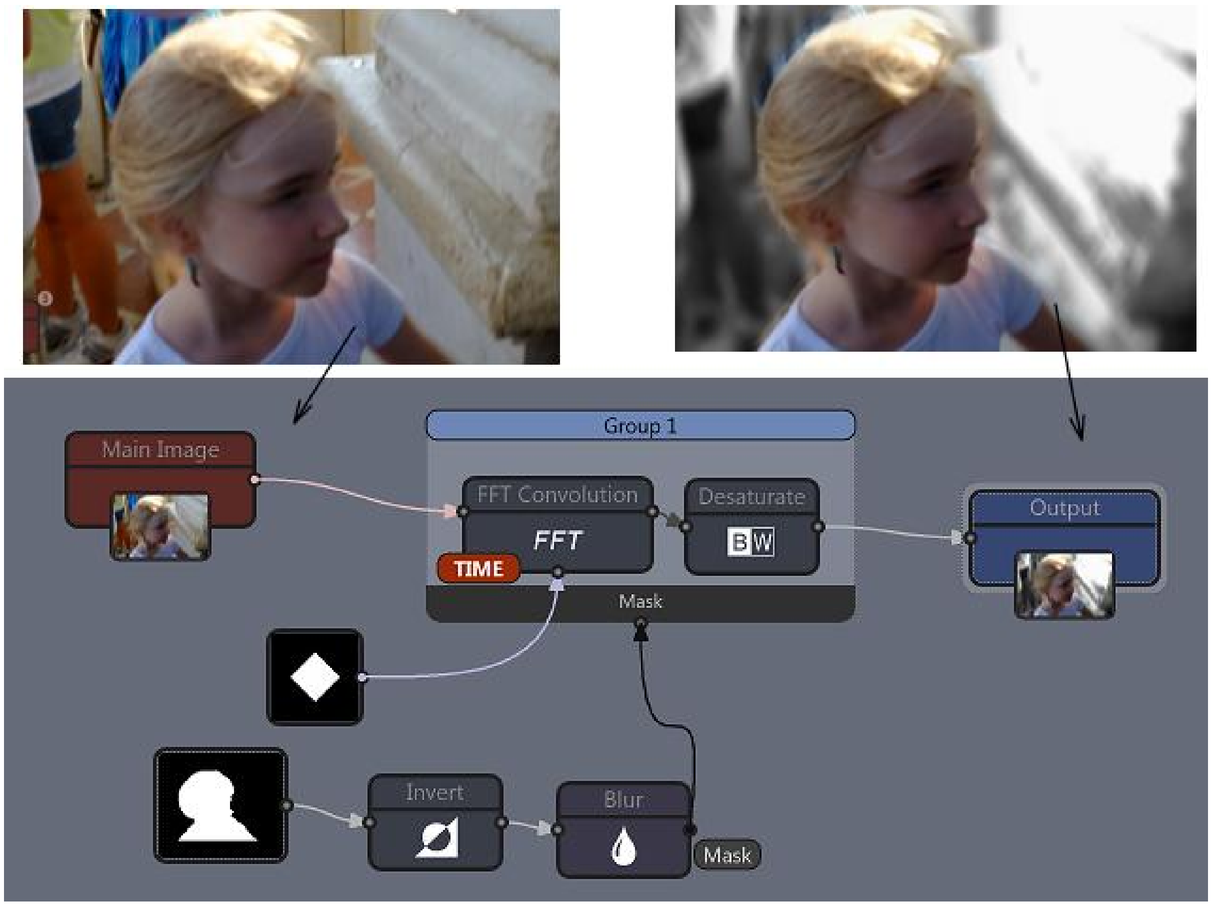


To simulate blooming the FFT convolution has also blooming parameter and threshold.

You can see the difference between de-focusing using FFT Convolution and Gaussian Blur. The Gaussian Blur simply makes the image soft while FFT convolution makes it de-focused.

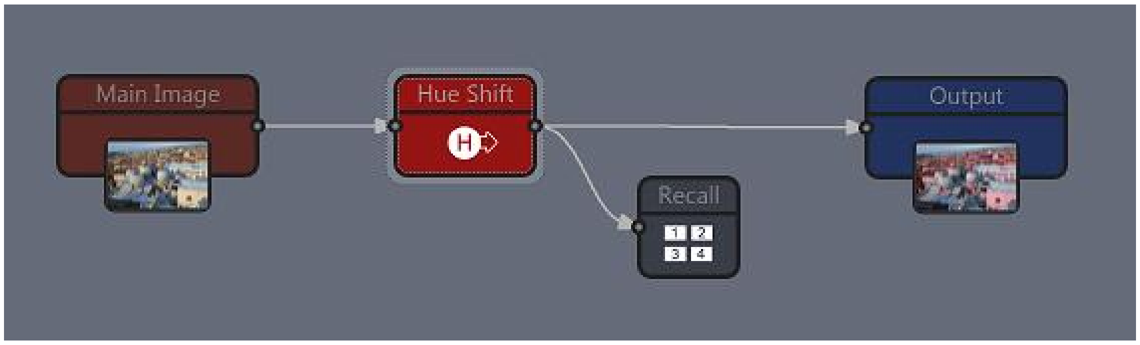


Here is an example of FFT Convolution with blooming while using Mask to de-focus (and de-saturate) the background. For Masking refer to the proper chapter.



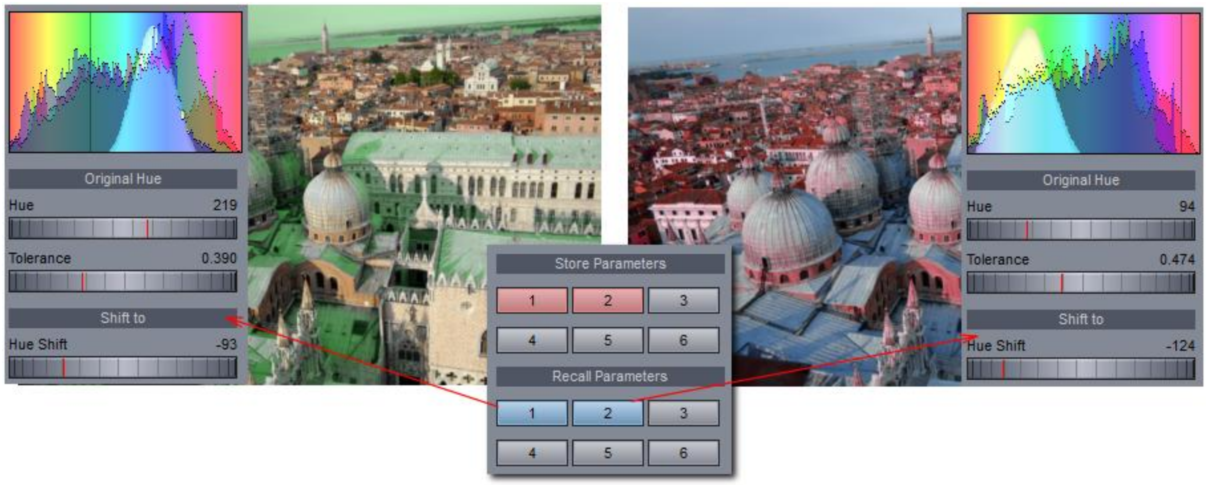
4.6 Recall Parameters

Recall is a helper object that can learn all parameters from an object and then recall them later. This is used to create presets for an object.



In the flow above we have Hue Shift which uses couple of parameters. If we want to store some settings and switch between them, we can connect the Recall object.

Now we will set the Hue Shift to the parameters we like and then press **button 1** in the Store parameters section of the Recall. Then we can change the Hue Shift parameters to different value and store them under **button 2**. We created two presets that can be quickly changed.
Any time we want to switch between those presets, we will use the blue buttons 1 or 2.

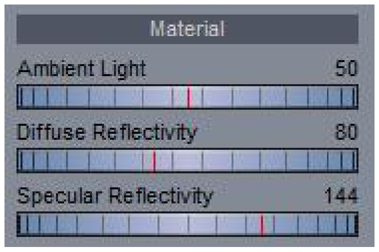


Tip: We can use the recall to record settings of one object, but recall it into another similar object.

4.7 Light Studio

Light studio allows you to add multiple colored light to the scene.

Light studio follows the naming convention of 3D software.
The top part is global material - this is how the material (the image) reflects the light.



Material

Ambient Light 50

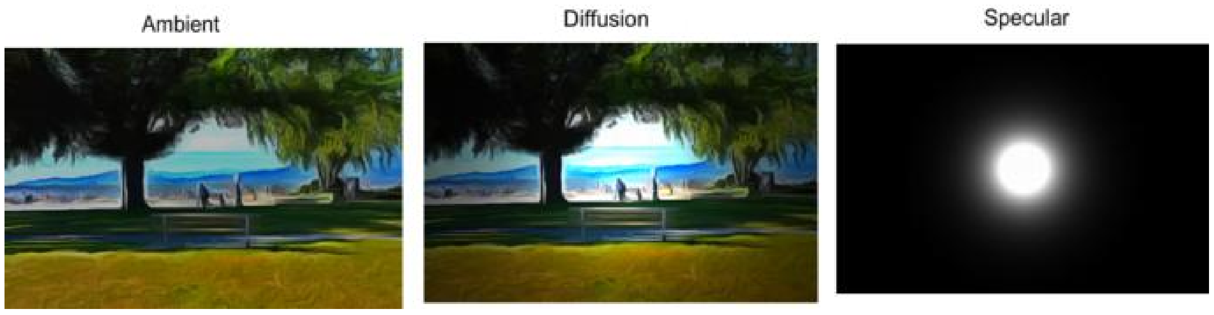
Diffuse Reflectivity 80

Specular Reflectivity 144

Ambient light
is a light present if no other light source exist. Ambient light has no direction and no position, it sets the dark point. Setting Ambient Light to maximum and Diffuse and Specular to zero will make the object look like sane as the input.

Diffuse reflection
is the reflection of light from a surface that reflects (scatters) at many angles. All matte materials have this type of reflections - the light is diffused. Plastic materials reflect more diffuse light than metal. Note that reflected light is the **main** carrier of color information. We see the color of an object because of the kind of light that the material reflects.

Specular reflection
is the mirror-like reflection of light from a surface where the light reflects in single direction. This is the light that creates light hotspots on shiny surfaces. Higher values create more sharper light reflection. Plastic materials reflect less specular light than metal. Specular reflection doesn't carry color information. If we set the Ambient Light and Diffusion reflection to zero, then all we would see is a light, but no original image.



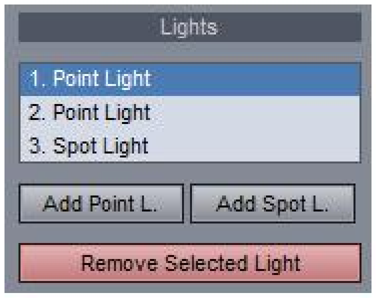
This is how various material properties affect the image.

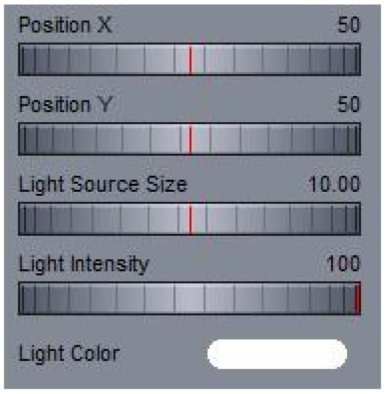
We can add many lights, both point and spot lights.

Point Light
Spot Light

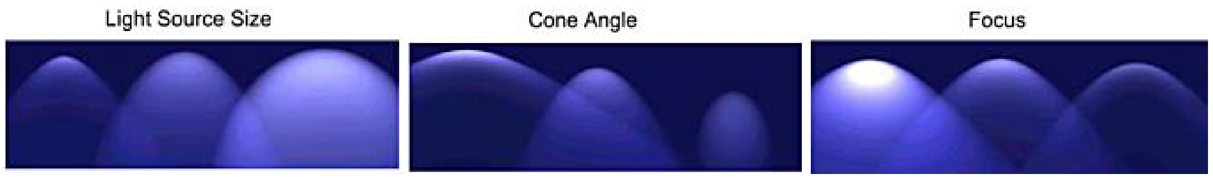
Point light is a very dynamic light and has less parameters to adjust. You may imagine this light as a light bulb or a reflector pointed straight at the object.

Spot Light is more complex light - it has position, direction, cone angle and focus. You may imagine this light as a reflector which can cast a light cone at the object.



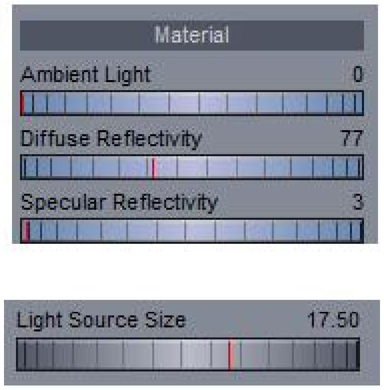


Position X 50
Position Y 50
Light Source Size 10.00
Light Intensity 100
Light Color



Note: There is difference between Light Studio and Light and Material object. Light and Material uses only one light but works with bump maps to add texture height. It can also use environmental mapping.

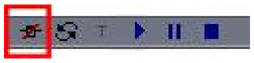
Tip: With some Diffused Reflectivity and both Ambient Light and Specular down while enlarging the light with Light Source Size you can make a soft light that falls towards the edges, almost like an vignette that both lights the center and darkens the edges (There is also a vignette object in reactor that is all subtractive)



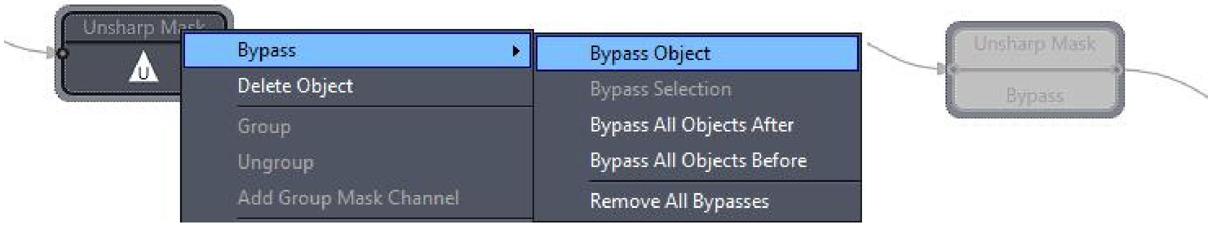
4.8 Bypass

Sometimes we would like to disable an object or a group for while.
Bypass does exactly that. The object will simply pass the input data to output without changing them.

The simplest way to bypass a selected object is to use the Bypass button on the bottom of object properties.

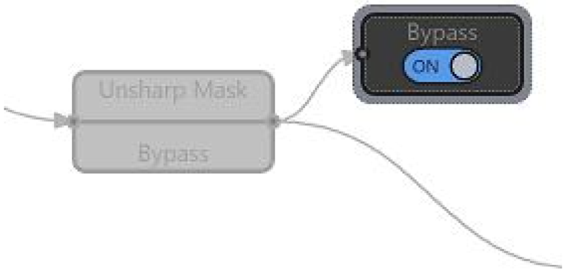


Also you can right click on any object and use Bypass from the menu.

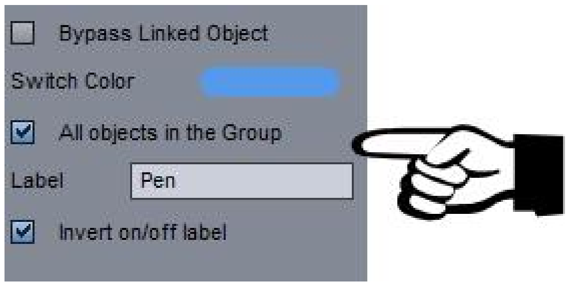


There are also options to Bypass objects that are calculated before or After the current object.

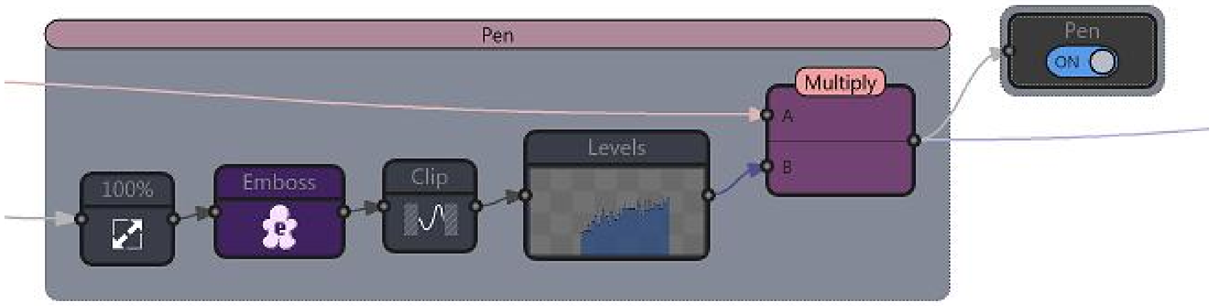
If we would like to bypass the object often we can use **Switch Bypass**. (Building blocks)
This is a special switch object that controls bypass of another objects or whole groups.



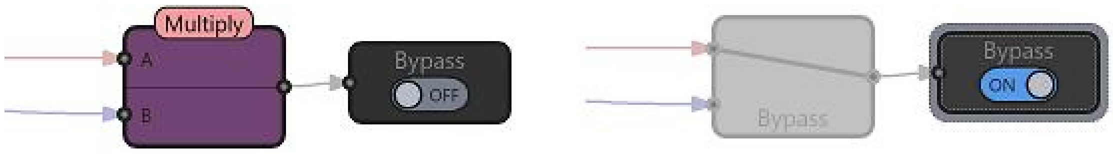
Group Bypass
A *whole group* can be set to bypass with the switch. Simply connect it to any member of the group and use *All objects in the Group*


☐ Bypass Linked Object
Switch Color
☑ All objects in the Group
Label: Pen
☑ Invert on/off label

You can also rename the bypass switch label and make it it inverse (that is it will show ON when not bypassed)
This can be used to create switch that adds some functionality to the flow, for example in this case the bypass switch controls adding outline to the image.
If we set the bypass (the label will read off because it is inverted) all the objects in the group will be bypassed and no outline will be added.



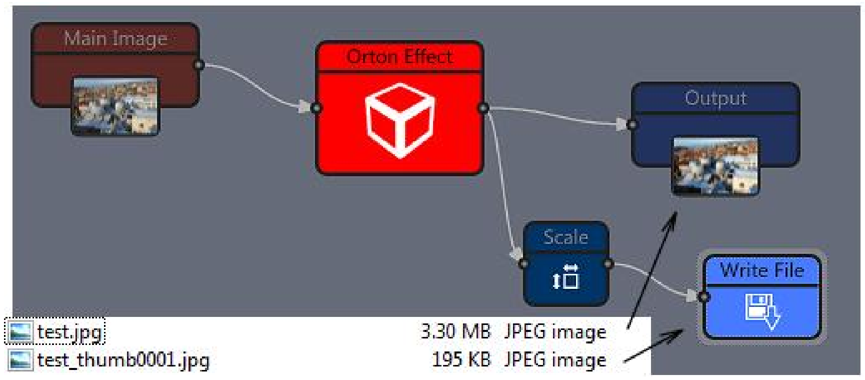
Note: Bypassing 2 input objects such as Blend Layer will always connect the Input 1 (top node marked A) with the Output.



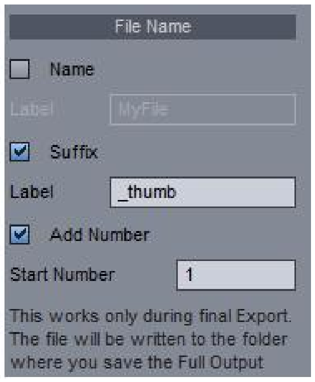
4.9 Write File

The Output Image will always save the file when we use Export Full Image, but we can save **additional** files at any place in the flow.

The name of the additional file will be generated according to the settings we choose in the Write File. In the flow below we are saving the image and also an additional thumbnail created by Scale object.



test.jpg 3.30 MB JPEG image
test_thumb0001.jpg 195 KB JPEG image


File Name

☐ Name

Label MyFile

☑ Suffix

Label _thumb

☑ Add Number

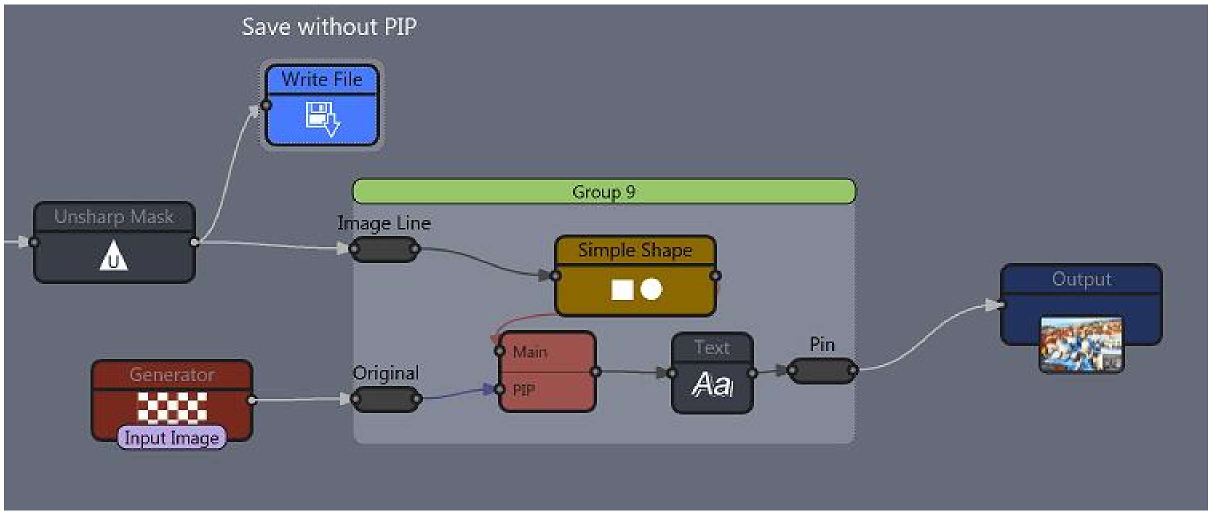
Start Number 1

This works only during final Export. The file will be written to the folder where you save the Full Output

If we don't use the Name, then the name of the output file will be used as the base name.
Then a suffix can be added to the base name and also a number that will increase each time we export a new set.

In our example if we export the flow once more, the test.jpg file will be overwritten but additional thumbnail image test_thumb0002.jpg will be created besides the previous one.

The Write File objects can be placed at any point in the flow. In the example below we are saving the image before a Picture in Picture is added to the final Output.

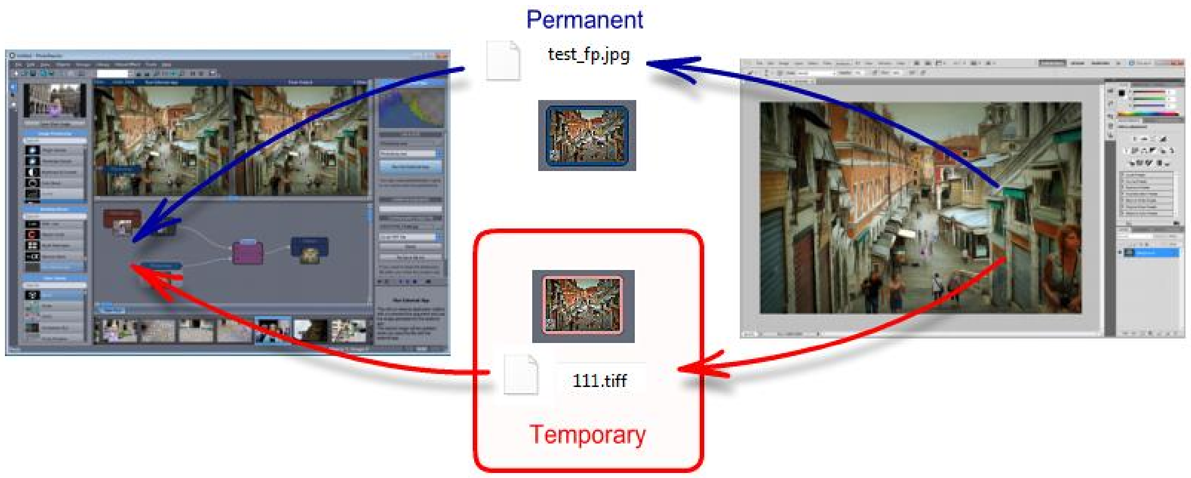


4.10 Run External App

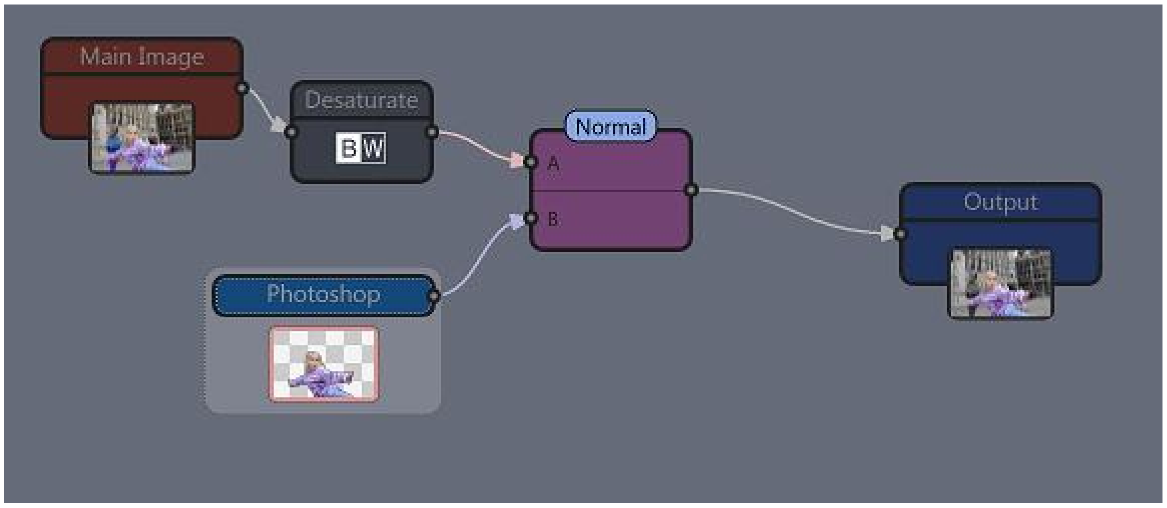
Run External App is an object that will run external application and then grab the image created by that external application.

It has only output and no input. The reason for **not having input** is that the external applications are not plugins, reactor can't control the external application to update input image.

The following graph shows how the applications communicate. It is done through either temporary or permanent file. A temporary file exists only until the object exist. Once we close the project, delete the object or load new project the temporary files will be deleted.



The Run External App calls the external application with a file as an argument and it is up to the external application to deal with it.
As soon as the temporary file is saved back (to the same file) by the external application, reactor will update the object.

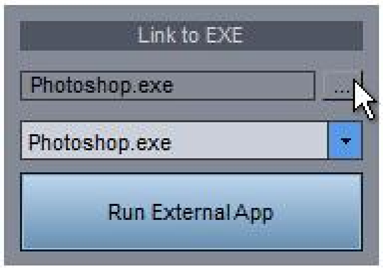


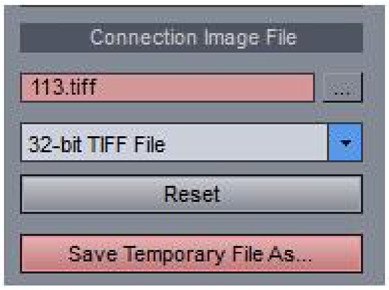
By default a temporary file is created from the full size Main Image. This is also done to setup the correct size and ratio.
However as soon as we delete the object or close the project, the temporary file will be also deleted. We

can choose to keep the file for later by Saving it as a permanent file.

First we have to find external application, using the Browse button in the **Link to EXE** section. The selected application(s) will be stored in the combo box below for a future quick access.

In the **Connection Image File** section we can choose what type of file is going to be used to communicate with the external app. As on the image below, I selected 32-bit TIFF because I want to modify alpha channel and the 32-bit TIFF works great with my external application (photoshop). The other choices are 24-bit BMP or JPG file if I don't need to carry the alpha channel. There is also 32-bit PNG for applications that don't support TIFF, but saving PNG file takes usually much longer. The object will create the temporary image automatically. JPG file loads and saves fast, but it is not losseless format.





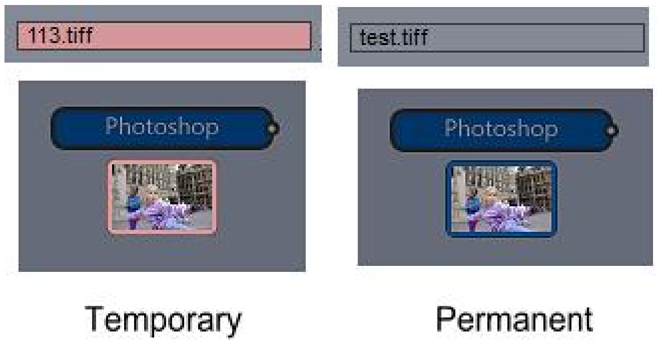
Run External App - External app with command line arguments
To start the external application, press the large blue button **Run External App**. Many external applications will check if another instances are running and will then open or update the image in the already running application. (Photoshop does it this way so it is very easy to use with Reactor). Other applications may not do that, and could open the image as a new file or in a new application window each time you press Run External App. Some applications cannot open image from command line argument.
It is important to realize that once the connection image file is chosen or created the communication is only one sided - The external application will update the image and the Reactor will watch and load changes, then update the flow. It isn't the other way around.

Reset
The reset button will re-create a temporary connection image from the Main Image. It will cancel all changes that were made by the external application. This is used if we for example load different Main Image - the connection image file is still the old one from previous section until we press Reset button.

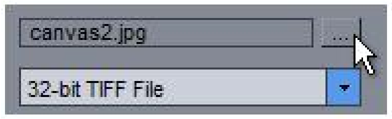
Temporary vs. permanent
Temporary files are stored momentarily in the windows temporary folder. The benefit is that they ensure we are not using an original file as the connection image for experiments but a copy file placed in temporary folder. The temporary files are automatically deleted when not needed. A project that uses temporary files will not load the same way back because those linked files will no longer exist. The object using temporary file can be easily changed into a using a permanent file with the Save Temporary File As button.
Permanent file is any other image file placed on a visible part of the drive. It is up to the user to make sure he/she is not using some valuable and irreplaceable photo for transferring data between the applications. Permanent files will stay there and so a project that refers to permanent files can be loaded later.

Save Temporary File As
As mentioned before, the temporary files have short life-span - until the object exist. This button will let

us to re-save the temporary file to a permanent file somewhere on your HD so it can be used later and it is not going to be deleted.
We can see if the connection image file is temporary (and will be deleted) or permanent by the different color of the object.



Load permanent connection file
We can use any other input image instead of the temporary file for the communication between the applications. Loading a connection file is done by using the browse button in the Connection Image File section.

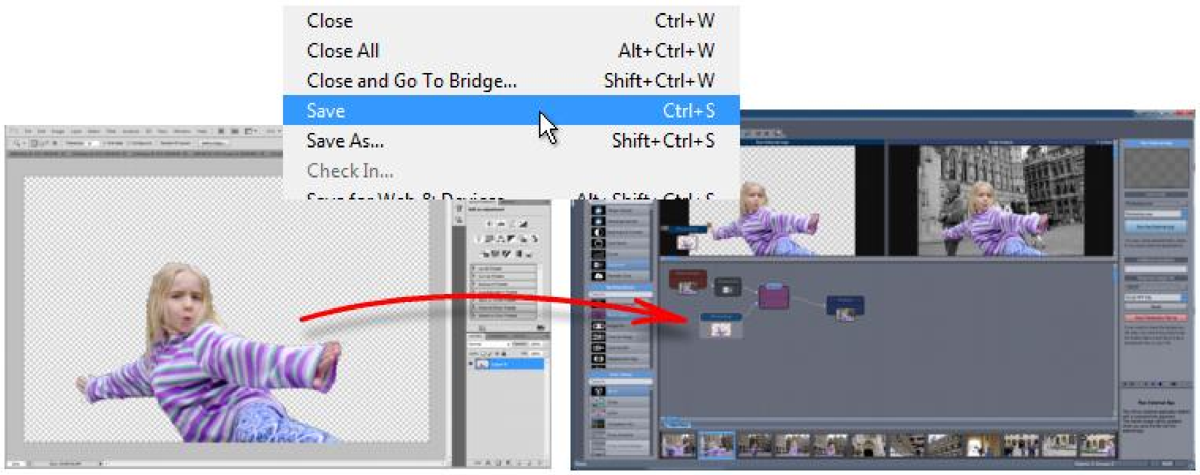


User has to be sure he/she is not mistakenly linking an irreplaceable original photo then using it for communicating between applications.

Note: On the image above I selected an existing jpg file as my connection image. In this case we don't have to worry about what is selected as the type (BMP, TIFF...) of the file.
Important: Loading and changing **existing** image in the external app will (of course) permanently modify it - beware of that. If you don't want to modify the original image, load custom connection image and immediately Re-Save it as another file with the"Save Copy As" button that replaces the "Save Temporary File As". Also you can turn it to temporary file by selecting the file type (BMP, TiFF...) in the combo.

Create Temporary image copy from an existing image
You can create a temporary copy from an existing image by dropping existing image file from Quick Access bar or from explorer over the object. Unlike the browse button, this will immediately create a **temporary** copy of the image (red ring around the image) so we can open it in external application, change and save it without changing the original image.

Update the image to Reactor.
The edited image is updated back to Reactor as soon as you save it in the external application.
It has to be saved to **the same file**. If an application doesn't allow you to save to the same file (perhaps it can read, but not save the format), you can save it to a different permanent file, then go back to reactor and choose that file as a custom connection file.

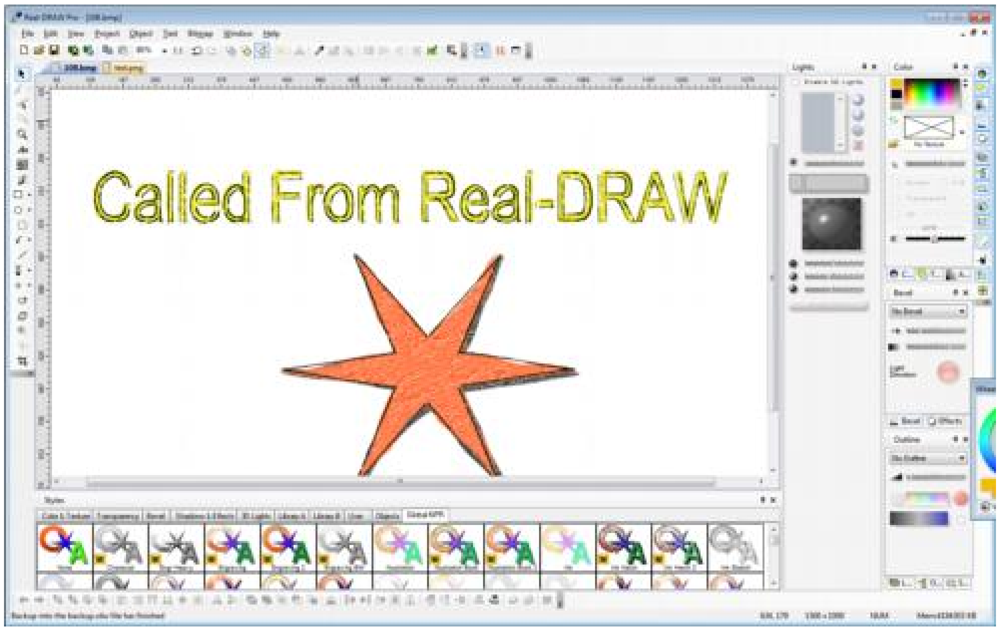


Note: Photoshop is a registered trademark of Adobe.

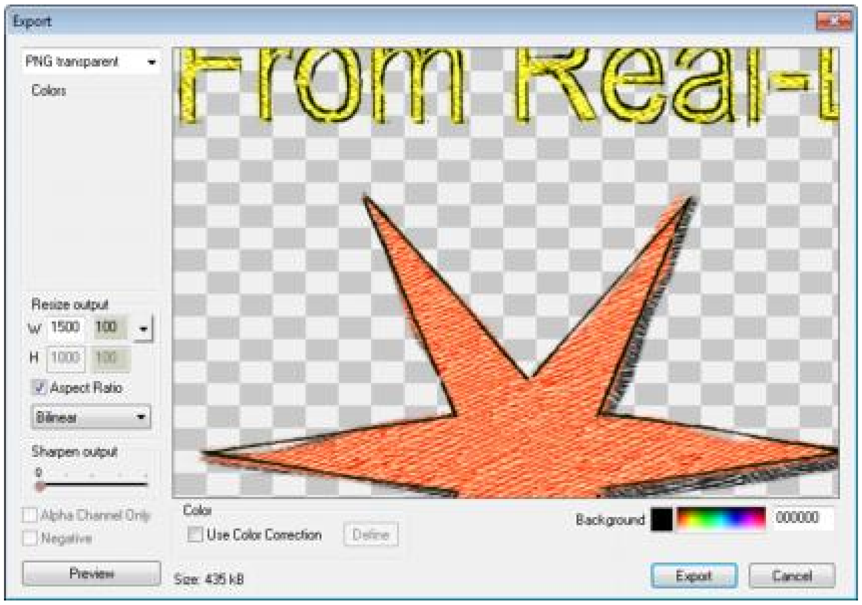
Example: Using Application without command line
Adding text from Mediachance **Real-DRAW**
Because real-draw is a vector-like application, we will not use Run External App button because that would open just a bitmap in Real-DRAW and we want to use vector objects instead. In fact we don't even need to load the realdraw.exe in the object "Link to EXE".
The link to exe could be empty or can have any other application set there. If we are not going to use the blue button, it doesn't make any difference.

Similarly this would work same way with other vector graphics (non-bitmap) applications.

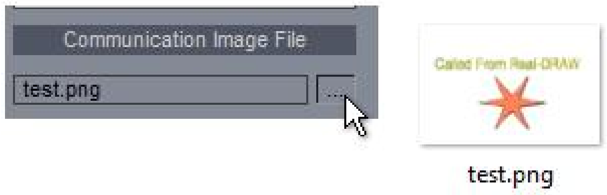
1. First run Real-Draw normally from Windows, **not** from reactor.
2. Set the Project to 1500x1000 - we don't need to use the same size as the image - it will be stretched.
3. Draw Something in Real-DRAW



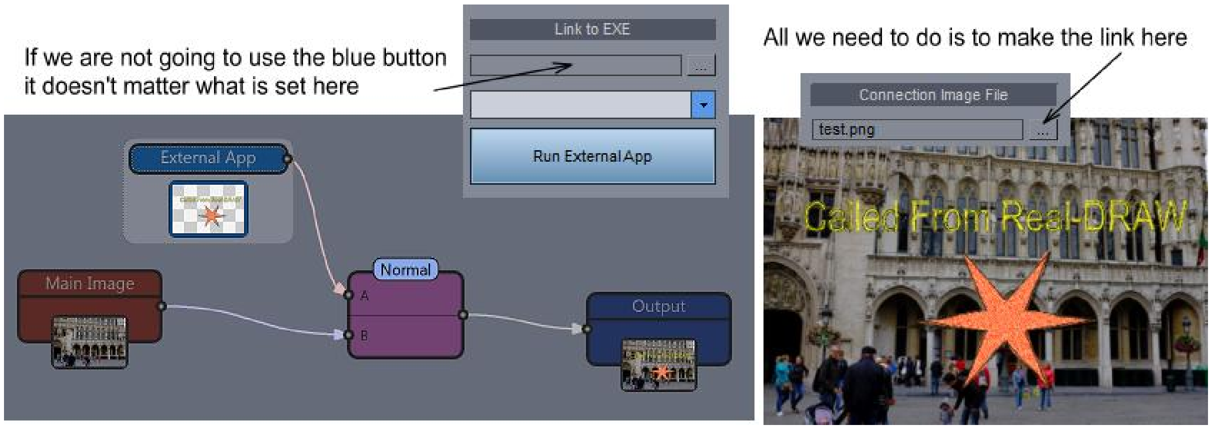
4. Now we will export the drawing as a transparent PNG, keep real-draw running.



5. In the Reactor, add the **Run External Object** and just load the Connection image - the png file you just exported. The link is established.



Mix the image with the Main image using Blend Layers.


If we are not going to use the blue button it doesn't matter what is set here
Link to EXE
Run External App
All we need to do is to make the link here
Connection Image File
test.png
External App
Main Image
Normal
A
B
Output
Called From Real-DRAW

Any time we want to **update the image** in the reactor, all we need to do is to Export the project in the Real-Draw to the same png file and reactor will automatically pick-up the changes.

Note: The link to permanent files is **persistent** across save. Any time we load back such project and change the linked file with **any** external application, reactor will update the changes.

5 Virtual Effects

This section is written with a lot of language terms explained in the previous paragraphs. If you try reading it without being familiar with the reactor work-flow lot of the sentences would sound completely gobbledygook, but that's the thing with a complex software.

So let's start.

Groups and Devices are more of a visual aid for us humans - to make structuring of the flow easier - but really they don't change anything in the flow, only how it is displayed.

However there is one special object that truly change the way reactor see the flow.

It is called **Virtual Effect**.

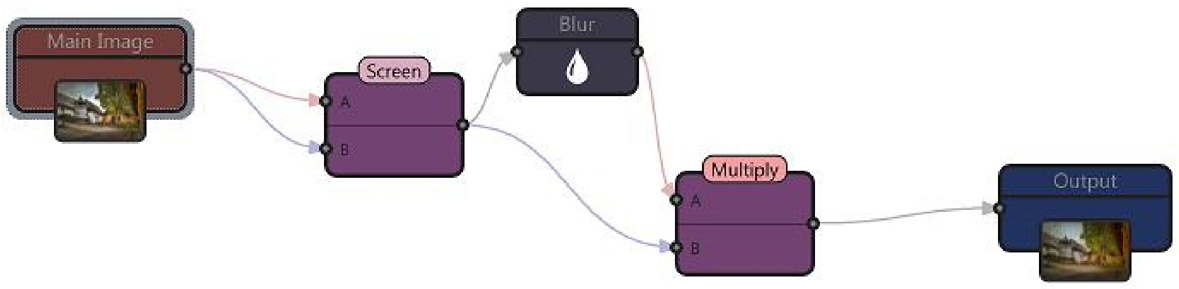
5.1 Basic Idea

Unlike the Groups, objects encapsulated inside a virtual effect **no longer exist** in the reactor flow.
If we create Virtual Effect from 10 objects, the reactor will at the end really see only 1 object - the Virtual Effect.
So unlike a Group, Virtual Effect truly encapsulate the objects inside and creates a **single virtual effect**.

Why we would want that?
Besides a full encapsulation of complex effect into a single object, the other big reason for developing Virtual Effect was that it can be later used in other compatible programs (that presumably will be future mediachance programs) as a sort of drop-in, or insert effect type that also carries its own parameters.
You may even think of it as a plug-in of some sort and Photo-Reactor as a developer tool.
A Virtual Effect is more or less a definitive version of some complex effect flow with specified selection of what parameters user can adjust and their default values.

Virtual Effect Parameters
The Virtual Effect parameters can be specified as a subset of any of the object parameters locked inside the Virtual Effect.

The best is to illustrate it on an example.
We will start with our old good Orton effect.

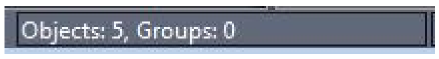


Object Crumbs:

ID88, 5, 1, 1, 192, 96, 0, P, 100.000, 0.000, 0.000, 7.000
ID89, 12, 88, 0, 304, 58, 0, P, 35.692, 1.000
ID90, 5, 89, 88, 397, 154, 0, P, 100.000, 0.000, 0.000, 3.000

For virtual effect we don't need any Binding Objects as they will ineffective inside the Virtual Effect. (But it is not necessary to delete them either if you have them)

Note in the status bar, the system shows 5 objects and this is indeed how many we see on the screen.


Objects: 5, Groups: 0

Note: The Virtual effect is always created from the whole flow, not from a selection. This way the Virtual Effect has automatically defined what is its single input (Main Image output node) and a single output (Output Object input node) and those are all the nodes it has - one input and one output, no more, no less.

External Files
You may add other images (such as textures) inside the virtual effect, but all image files will be carried as a references.
That means, they will work on your computer (if you don't move the files) but would not be found on others computers which may limit the effect transferability.
The exceptions are the image files in the photo reactor /textures/ folder. If you use any **factory supplied** textures from there, they will be correctly loaded in other computers as well.

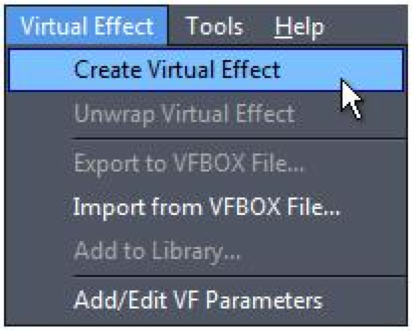
Note: When Photo-Reactor runs into a file that doesn't exist on a target computer it will look in the textures folder if it is there. That means if you want portability of the Virtual Effect and it requires an external image, the end user needs to copy that image to /textures/ folder of his/hers Photo-Reactor (or compatible application).

Main Image settings such as resize are ignored - those settings will have no effect on the final Virtual Effect. Any other resize objects inside the Virtual Effect are of course fully functional.

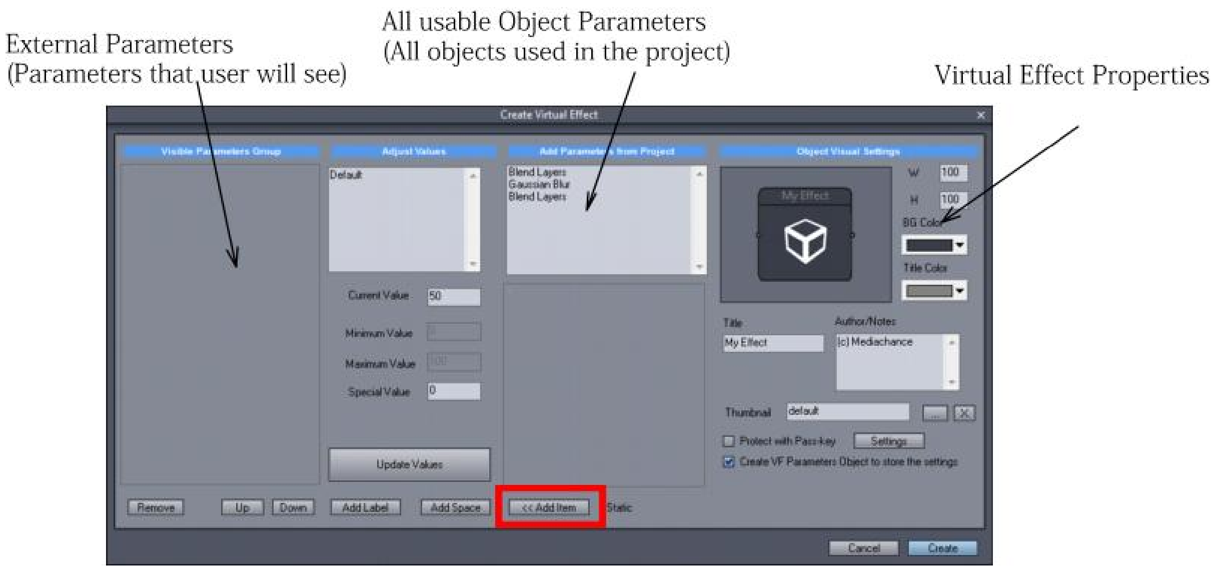
Let's turn the flow into a shiny Virtual Effect.

5.2 Create Virtual Effect

To create Virtual Effect, use menu **Virtual Effect - Create Virtual Effect**.



We will get into an interesting dialog box. A major part is to select the parameters from the project we want to expose as Virtual Effect Parameters where smaller part is for other properties.



Default

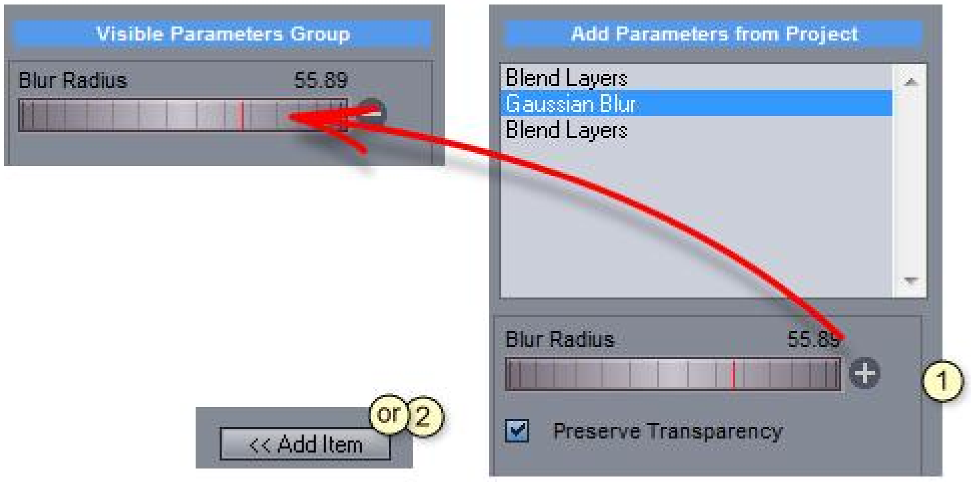
Blend Layers
Gaussian Blur
Blend Layers

Current Value 50
Minimum Value 0
Maximum Value 100
Special Value 0

W 100
H 100
BG Color
Title Color
Title My Effect
Author/Notes (c) Mediachance
Thumbnail default
☐ Protect with Pass-key Settings
☑ Create VF Parameters Object to store the settings

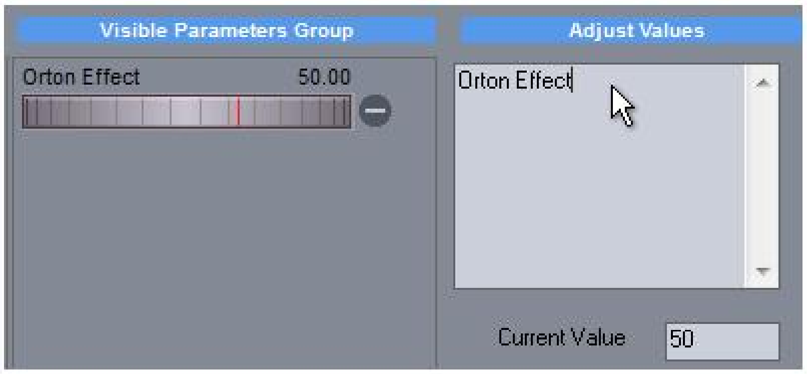
The left most column is a list of **all parameters that will be exposed to the outside world**. And as you can see there is nothing yet. Such Virtual Effect box will simply have no adjustable parameters. In our case what we really want user to adjust is the Blur Radius of the Gaussian Blur object.

Select the Gaussian Blur in the list of parameters and all of the gaussian blur parameters will appear in the window beneath it.



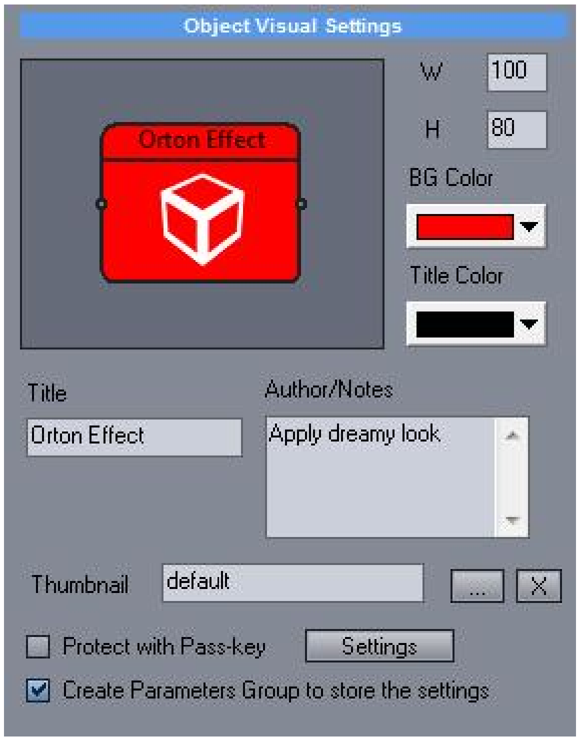
We can either click on the desired control and press the **plus** sign on the right (1) or use Add Item button (2). In both cases the control will be added to our Visible Parameters list.

The title *Blur Radius* may be too general and in an effect where there may be many of them it will be also too confusing, so we will change that.
Select the control in the left list and type in the Adjust Values.



We can also change the default value (you would need to use the large **Update** button to reflect the changes in the control)

On the right side there are general parameters about our Virtual Effect, name and the look, so let's change them.



W 100
H 80
BG Color
Title Color
Title: Orton Effect
Author/Notes: Apply dreamy look
Thumbnail: default
☐ Protect with Pass-key
Settings
☑ Create Parameters Group to store the settings

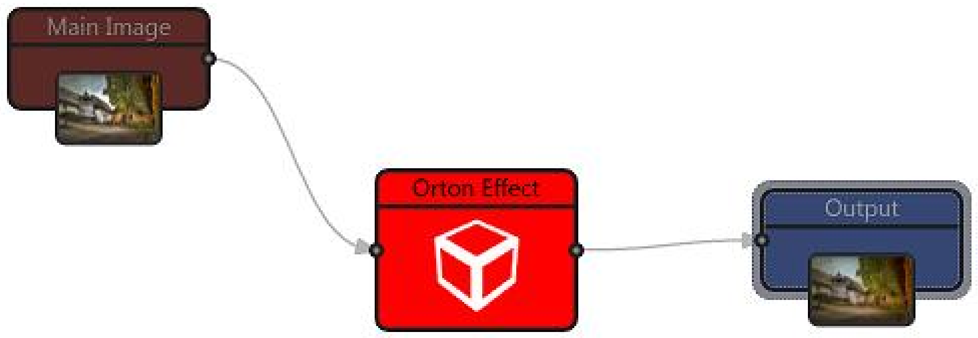
Note the **Create Parameters Group** settings and leave it ON. This will add a Parameters Group object inside the Virtual Effect and write a copy of the selected parameters and the general settings. From the point of Virtual Effect this is redundant, however if we need to unwrap the effect, the Parameters Group object will become very handy, because we will not need to repeat the steps we just did. Read more about that later.

Note: The currently set values of all parameters in the project will became a default values of the Virtual Effect.

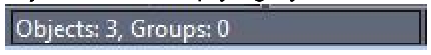
Now press **Create** button

5.3 Virtual Effect Usage

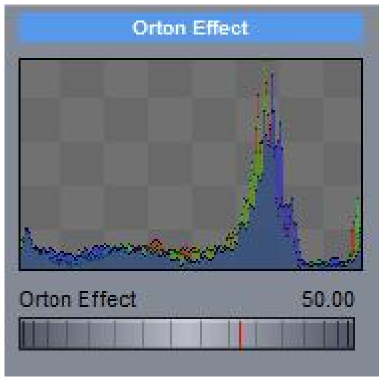
In the previous chapter we created the Virtual Effect by using Virtual Effect menu - Create Virtual Effect. Now all the objects disappeared and we have a single object between Main Image and Output.



If you look in the status bar, now we have 3 objects in our flow, so unlike the groups that only hide the objects from our prying eyes, Virtual Effect truly encapsulate the objects.


Objects: 3, Groups: 0

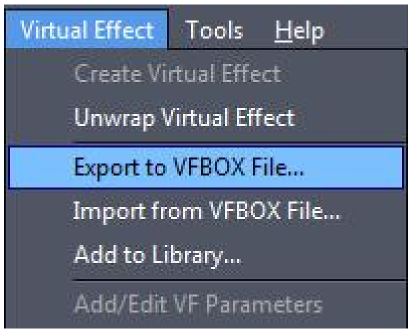
And since we set the visible parameters, the Orton Effect slider indeed appears in the Virtual effect properties.


Orton Effect

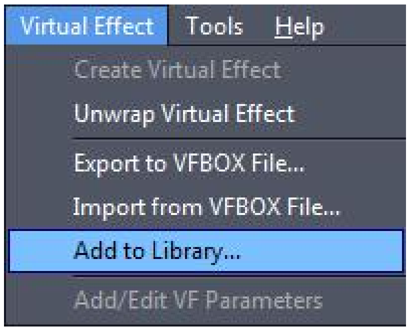
Orton Effect 50.00

What to do with it?
It is a full encapsulation of an effect. The new object behaves exactly as any other object in reactor regardless how many items it has inside.
Also as mentioned before the "bigger picture" was to create an exchangeable "plug-in -like" format that can be then used in other compatible software (it is safe to assume that during writing this tutorial there are no such programs) and it can be exchanged with other users.
You can use it in this form in your flows and add it to library if you wish.

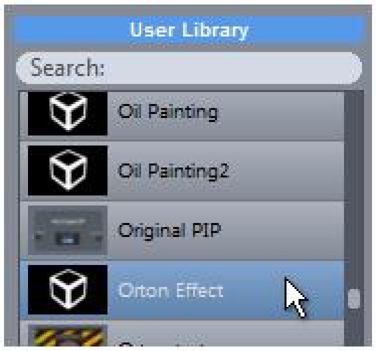
Export for external Compatible applications
The file-exchange format has extension *.vfbox and it can be created from menu Virtual Effect.
First select the Virtual Effect in the workspace then use Export to VFBOX File.



Add To Library
To add the Virtual Effect to your library use Add To Library command.

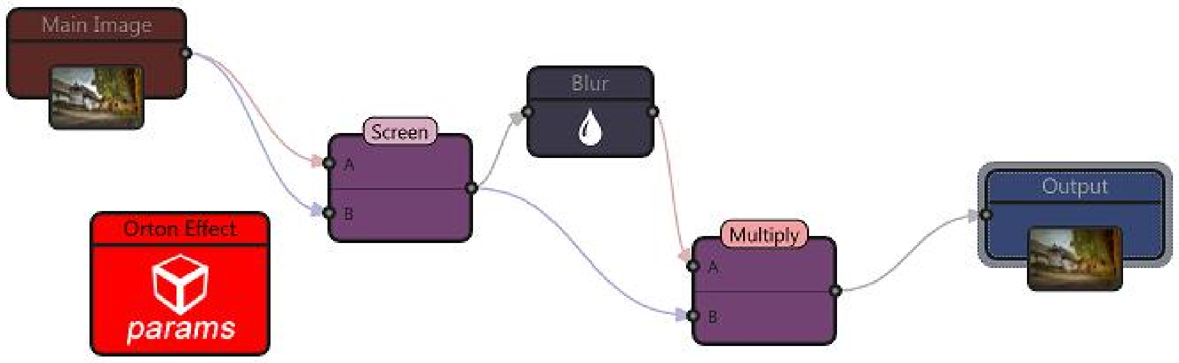


You can name it (the name that will appear in the library) and choose icon (the icon that will appear in the library)
And there you have it:



5.4 Unwrapping Virtual Effect

Unwrapping virtual effect means expanding Virtual Effect back to the original flow.



As we unwrapped the effect there is one more object inside! It is the Parameters Group box mentioned in the Create Virtual Effect.
If you double-click on the **params** object you will see the dialog is very much like the Create Virtual Effect Dialog.
If you going to create the virtual effect the settings will be copied from the Parameters Group.

5.5 Parameters Group

Parameters Group object does not process any data. (it has no input or output)
It only groups parameters of all other objects that can be then used from a single place.

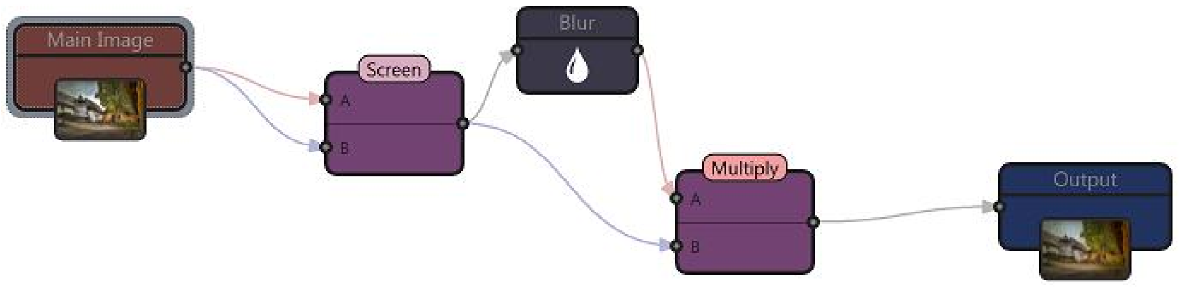
For Virtual Effect the Parameters Group holds a copy of the Virtual Effect settings that would be lost if we unwrap the Virtual Effect.

However while this may seems a primary usage, Parameters Group can be used also for a very nifty tasks.

Group important parameters in a single place without running wires to it!
If you create a very complex flow with a lot of parameters then you know that adding more sliders and knobs may not be ideal solution - it will only increase complexity and make the data scattered over the whole workspace with lot of extra wires.

Parameters Group may help you with this. You can collect all important parameter settings, sliders or checkboxes from the whole flow in one object (you guessed it: Parameters Group), then use that object as an alias for direct access to these settings.
Note: To structure your work-flow, you can create more than one Parameters Group, each with different controls.

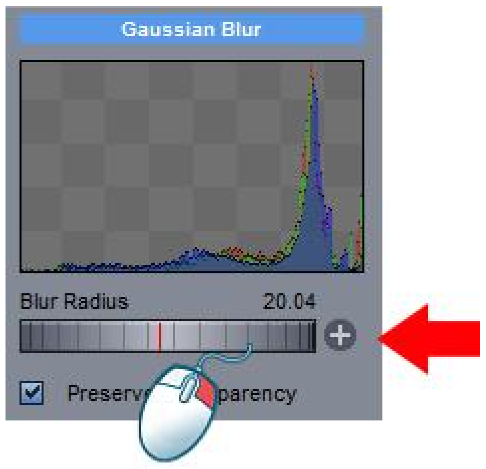
Let's just go with our example.
If you have Parameters Group, delete it, so we are starting from a clean Orton project.



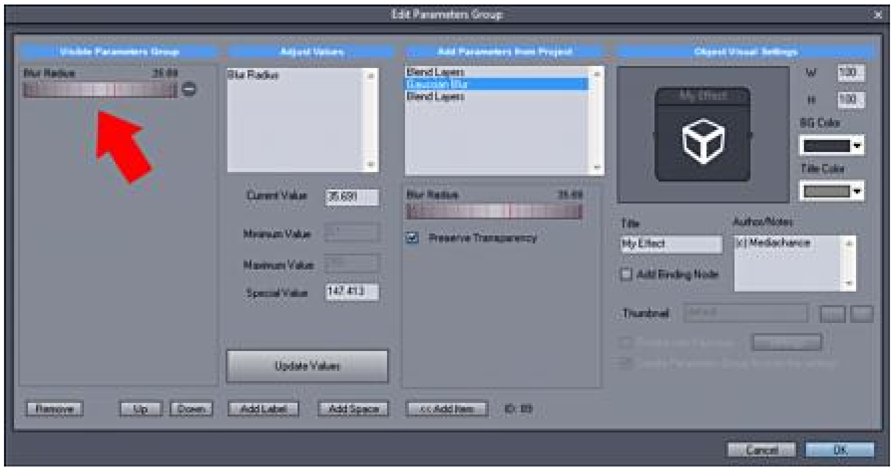
As previously we would like to pinpoint the Blur settings, but this time, let us also use the Opacity slider in the Screen blend layer so we have more settings.

Select the blur object and look in the properties panel.

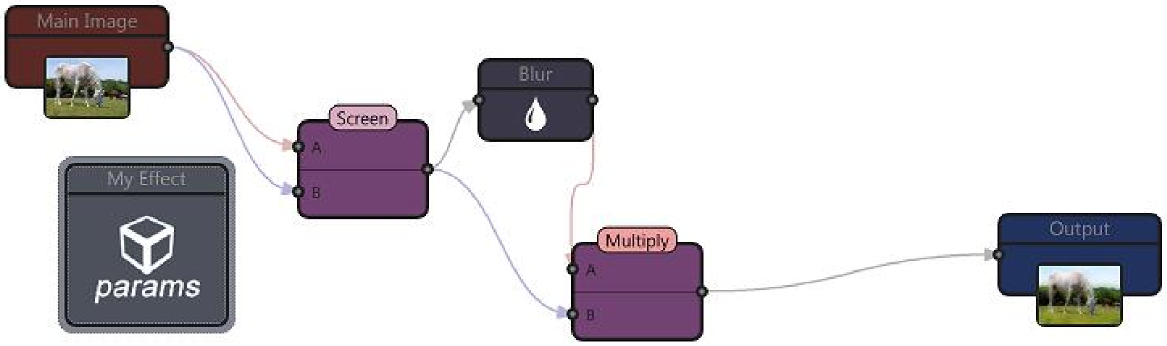
To grab anything from the properties window to be stored in the Parameters Group, here is a trick: **Right Click** on the control:



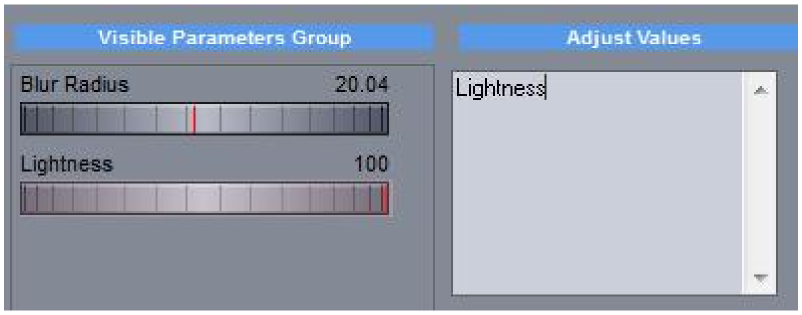
As you right-click on any control a + sign will appear beside it. Right click on the Blur Radius Slider. Click on the + sign.
The familiar dialog will open with the Blur Radius already pre-selected.



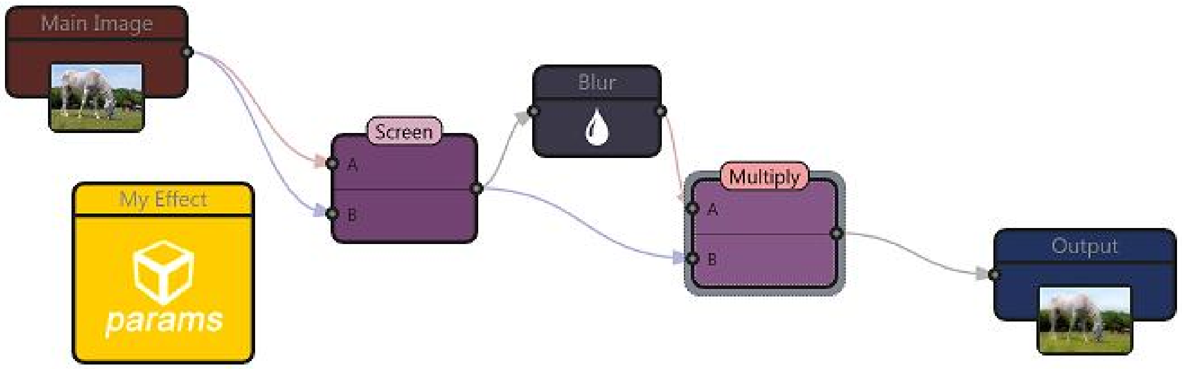
IF you press OK a Parameters Group object will be created.



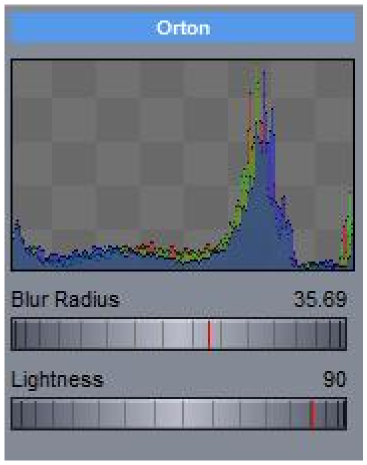
Let's repeat this with the Screen Opacity slider.
- Select the Screen Blend Box, - Right click on the Opacity Slider in Properties, - Click on the plus sign.
The Slider will be added to the Parameters Group.
The Opacity name doesn't say much, but in the effect we can control Lightness with it. So let's rename it so and change some properties, like name and color.



Now We have Parameter box that has shortcuts to object parameters scattered around the workspace - not here, this is trivial project - but in General.



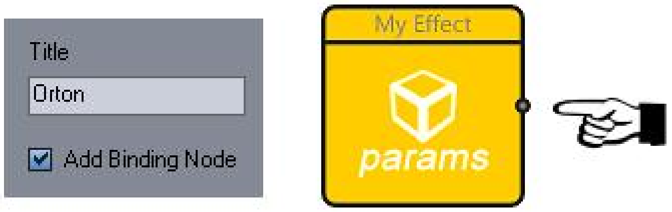
If you select the Parameters Group, in its properties you will see the very same controls and you can use them to directly change the *invisibly* linked original parameters.



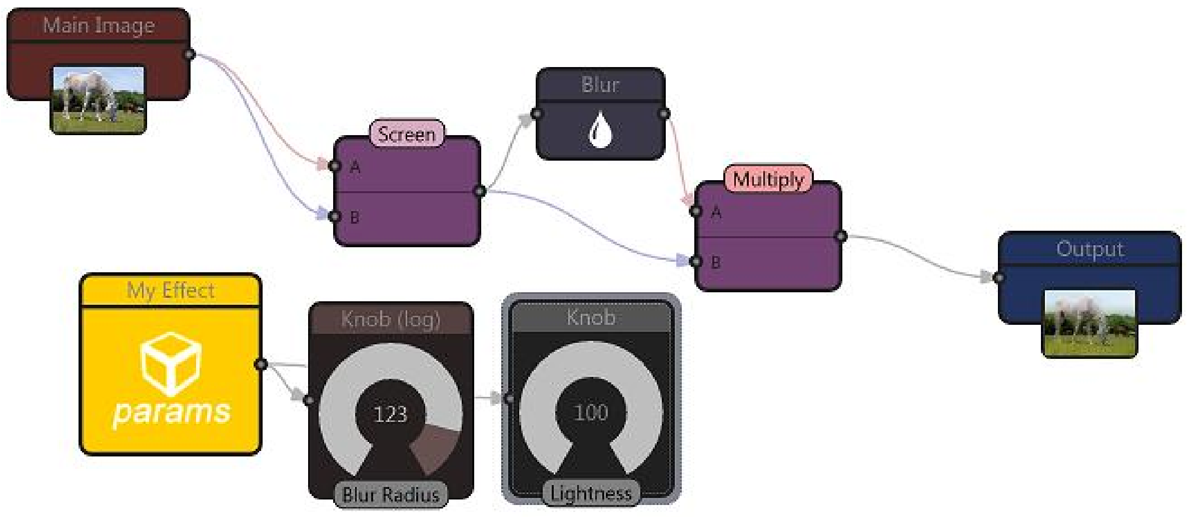
Note: You can use the Parameters Group exactly like described above before you use Create Virtual Effect. This way everything will be prepared for you in the Create Virtual Effect dialog.

If you Double-Click on the **Parameters** Group you may spot there is one extra parameter in the dialog.

Add Binding Node



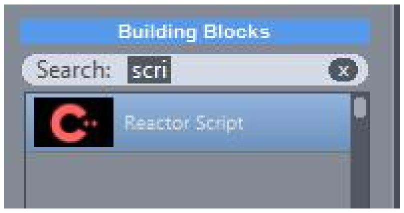
You can add output node to the Parameters Group. It will not output any meaningful data, the whole purpose is so we can add binding object to it!



This way we can fully separate the parameters from the flow itself and dedicate one corner of our workspace just for parameters (without the need of more spaghetti going across the whole project).

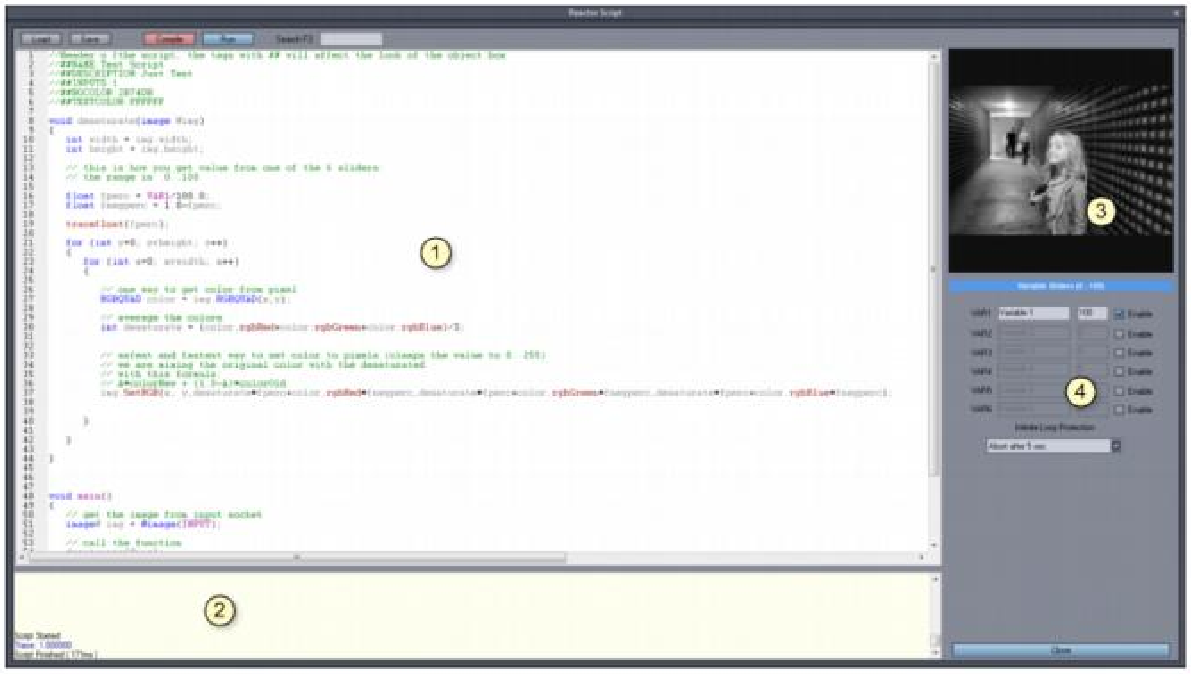
6 Reactor Script Reference

Reactor script is one of those hard-core feature that may not have immediate appeal to wide audience, but it offers another layer of unlimited potential.
Translation: if you are not a programmer or simply not interested, skip the whole block.



Reactor Script object is a fully scriptable objects with a C++ syntax. The code is object oriented, supports functions and classes with inheritance and polymorphism.

Script Compiler
Reactor Script is a bytecode compiler much like a java. It offers many advantages while maintaining reasonable run-time speed.
To start editing, double click on placed object or click Edit Script button in the properties.



The scripting dialog:
1 - source code
2 - debug/output window
3 - processed image
4 - external sliders

Compile
Compiles the script without running it.

Run
Runs the script and if needed it also compiles it.

Search
Type search phrase and press F3
Alternatively, select a word in the source code editor and press CTRL+F3 (Similar to MS VS)

Sliders
There are 6 sliders that will appear in the properties page. The sliders are in range of 0..100 and are exposed to variables VAR1 ... VAR6

6.1 Features

Reactor script is a high level language that uses close c++ syntax with only a minor differences (there are no pointers, instead we use object handles).

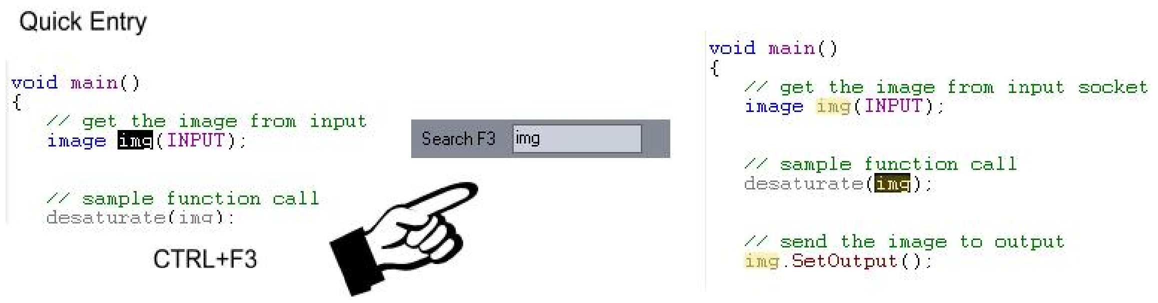
- Unlike most scripting languages that are dynamically typed Reactor Script uses the same static types as C++ which means lots of bugs can be caught during compile time.
- Reactor Script is fully object oriented where you can declare classes with single inheritance and polymorphism.
- Reactor Script supports arrays and multi-dimensional arrays with build-in search and sorting functions.
- We provide you with few classes to access and work with image data

The goal of this guide is not to teach you c++. It is written more like a reference and it assumes the reader knows programming and c++.
So if you are new to this, let's just say the following sections may not be that thrilling.

Working with the script window.
Type script and press Compile to compile the script. Alternatively p[ress Run which will first compile then run the script.
Output window will show erreos and warnings.

Search
Enter search string in the search box and press F3. All occurrences will be highlighted and you can cycle through them with each pressing of F3.

Search - **Quick Entry**
Selecting a text and pressing CTRL+F3 will enter the string to the search box and highlight all occurrences. Pressing F3 will cycle through them.



```
void main()
{
   // get the image from input
   image img(INPUT);

   // sample function call
   desaturate(img);
```

```
void main()
{
   // get the image from input socket
   image img(INPUT);

   // sample function call
   desaturate(img);

   // send the image to output
   img.SetOutput();
```

6.2 Types

Primitive types

Integers

type	min value	max value
int8	-128	127
int16	-32,768	32,767
int	-2,147,483,648	2,147,483,647
int64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint8	0	255
uint16	0	65,535

uint	0	4,294,967,295
uint64	0	18,446,744,073,709,551,615

Real

type	range of values	smallest positive value	maximum digits
float	+/- 3.402823466e+38	1.175494351e-38	6
double	+/- 1.7976931348623158e+308	2.2250738585072014e-308	15

bool
bool is a boolean type with only two possible values: true or false

Usage of typedef
Typedef can be used to create alias for primitive types
```
typedef int8 BYTE;
```

Enums
Enums are integer constants that are used throughout the script as named literals instead of numeric constants to improve readability of code

```
enum MyColors
{
  red = 1,
  green,
  blue
}
```

6.3 main function

In Reactor Script the point of entry is ALWAYS void main() function

```
void main()
{
      // get the image from input socket
      image img(INPUT);

      img.Blur(4.5);

      // send the image to output
      img.SetOutput();

}
```

Factory global variables and functions

`INPUT`	constant, has value 1
`INPUT1`	constant, has value 1
`INPUT2`	constant, has value 2
`ZOOM`	float number, currently used zoom
`FULLWIDTH`	integer, full width of Main Image (it is not the same as preview width or the width obtained from INPUT image!)
`FULLHEIGHT`	integer, full height of Main Image (it is not the same as preview height or the height obtained from INPUT image!)
`VAR1...VAR6`	float in range 0..100, reflects the value of interface sliders
`traceint(int value)`	displays integer number in the output window: traceint (myvariable);
`tracefloat(float value)`	displays float number in the output window: tracefloat (myvariable);

6.4 image class

The most important class that connects script with the reactor object is *image* class

image class holds pixel values of uint8 for r,g,b,a in the range 0...255

`image( uint width, uint height )`	creates empty image of width and height
`image( uint inputnode)`	receives image from input node, valid arguments are 1 or 2 constants INPUT, INPUT1 and INPUT2 are provided for readability purposes (INPUT=INPUT1)
`image( image& img)`	creates copy of image img.
`uint width`	object variable, width of the image
`uint height`	object variable, height of the image
`void SetOutput( )`	set the image to the output node
`void SetRGB (uint x, uint y, int r, int g, int b)`	set the r,g,b values at the pixel (x,y), values are clamped to 0..255
`void SetR(uint x, uint y, int value)`	sets value of Red for the pixel (x,y), values are clamped to 0..255
`void SetG(uint x, uint y, int value)`	sets value of Green
`void SetB(uint x, uint y, int value)`	sets value of Blue
`void SetA(uint x, uint y, int value)`	sets value of Alpha
`void Copy(image imgIn);`	create copy of imgIn data
`uint8 Red(uint x, uint y)`	returns or sets the red value at pixel (x,y) the value is not clamped
`uint8 Green(uint x, uint y)`	returns or sets the green value at pixel (x,y) the value is not clamped
`uint8 Blue(uint x, uint y)`	returns or sets the blue value at pixel (x,y) the value is not clamped
`uint8 Alpha(uint x, uint y)`	returns or sets the alpha value at pixel (x,y) the value is not clamped
`uint8 Channel (uint channel, uint x, uint y)`	returns channel value (channel is 0,1,2,3 in Blue,Green,Red, Alpha order)
`pixel Pixel(uint x, uint y)`	returns or sets the pixel structure at the pixel (x,y)
`RGBQUAD RGBQUAD(uint x, uint y)`	returns or sets RGBQUAD structure at the pixel (x,y)
`void Resize(uint x, uint y)`	resize the image to new size x, y
`void Blur(float radius)`	blurs the image with radius
`void Sharpen(int nAmount)`	Sharpensd the image with amount 0..100
`uint8 Luma(uint x, uint y)`	returns luminance at pixel (x,y)
`void Fill(int r, int g, int b, int a)`	fill whole image with the r,g,b colors and the alpha a
`void RGB2YUV()`	convert internally the whole image from rgb to YUV, You will still access the data as if they are rgb where Y = r, U = g, V = b, a is unchanged

`void YUV2RGB()`	convert internally the whole image from YUV back to rgb

Note:
The image class does not provide assign (=) operator to avoid the mistake between assigning object handles and objects which would be costly during processing of large images. If you try to assign image object instead of image object handle you will get compiler error.
To physically copy data from one image object to another use Copy method.

Helper structures

Please note both pixel and RBQUAD have the same functionality, they differ only in object attribute names (the later is similar to windows convention).

pixel
pixel (uint8 R, uint8 G, uint8 B)
pixel (uint8 R, uint8 G, uint8 B, uint8 A)

uint8 r
uint8 g;
uint8 b;
uint8 a;

RBQUAD attributes

uint8 rgbBlue
uint8 rgbGreen;
uint8 rgbRed;
uint8 rgbAlpha;

Usage Examples

Receive image from input node 1
```
image img(INPUT); // receives object with image from INPUT node
```
using handles (aka safe pointers):
```
image@ img = @image(INPUT);  //receives object handle to the image from INPUT node

Note: INPUT and INPUT1 are interchangeable and are for convenience
```

Sets the image to output node
```
img.SetOutput();
```

Get width and height of image
```
int width = img.width;
int height = img.height;
```

Copy image (physically creates copy of the img data)
```
image img2;
img2.Copy(img);
```

or use copy constructor
```
image img2(img);
```

Handle assignment (does not create copy of the data, img2 points to the same data of img)
```
image@ img2 = @img;
```

Gets value from x,y
```
pixel color = img.Pixel(x,y);
```
alternative
```
RBQUAD color = img.RBQUAD(x,y);
```

gets value of red
```
int red = img.Red(x,y);
```

sets value of red to 126
```
img.Red(x,y) = 126;
```

sets r,g,b values (this is **preferred** function because it clamps values to 0..255)
```
img.SetRGB(x,y, red, green, blue);
```

sets r,g,b using pixel structure
```
img.Pixel(x,y) = pixel(red,green,blue);
```

blur image with radius 4.5
```
img.Blur(4.5);
```

6.5 float image class

Similar to the image class, *fimage* class holds image pixels as float values
fimage class holds only subset of methods of *image* class.
Note: it is preferred to use image class. float image class require 4 x more memory to hold the pixel data.

fimage class holds pixel values of float for r,g,b,a in the range 0.0...1.0

`@fimage( uint width, uint height )`	creates empty fimage of width and height
`@fimage( uint inputnode)`	receives fimage from input node, valid arguments are 1 or 2 constants INPUT, INPUT1 and INPUT2 are provided for readability purposes (INPUT=INPUT1)
`uint width`	object variable, width of the image
`uint height`	object variable, height of the image
`void SetOutput( )`	set the image to the output node
`void SetRGB (uint x, uint y, float r, float g, float b)`	set the r,g,b values at the pixel (x,y)
`void Copy(fimage imgIn);`	create copy of imgIn data
`float Red(uint x, uint y)`	returns or sets the red value at pixel (x,y) the value is not clamped
`float Green(uint x, uint y)`	returns or sets the green value at pixel (x,y) the value is not clamped
`float Blue(uint x, uint y)`	returns or sets the blue value at pixel (x,y) the value is not clamped
`float Alpha(uint x, uint y)`	returns or sets the alpha value at pixel (x,y) the value is not clamped
`float Channel (uint channel, uint x, uint y)`	returns channel value (channel is 0,1,2,3 in Blue,Green,Red, Alpha order)
`fpixel Pixel(uint x, uint y)`	returns or sets the fpixel structure at the pixel (x,y)
`void Resize(uint x, uint y)`	resize the image to new size x, y
`float Luma(uint x, uint y)`	returns luminance at pixel (x,y) in the range 0..1

Note:
The fimage class does not provide assign (=) operator to avoid the mistake between assigning object handles and objects which would be costly during processing of large images. If you try to assign image object instead of image object handle you will get compiler error.
To physically copy data from one image object to another use Copy method.

Helper structure

fpixel
fpixel (uint8 R, uint8 G, uint8 B)
fpixel (uint8 R, uint8 G, uint8 B, uint8 A)

float r
float g;
float b;
float a;

Usage
See image class.

6.6 rndgen class

Simple generator of pseudo-random numbers

`void Randomize()`	set the random generator seed from the time
`void SetSeed(uint seed)`	set the random generator seed. A same seed will produce the same sequence of pseudo-random numbers
`float GetFloat()`	returns pseudo-random number in the range 0...1.0

Example:

```
rndgen rnd;

rnd.Randomize();

int n = 0;

while( n++ < 100 )
{
    float randomnum = rnd.GetFloat();
    tracefloat(randomnum);

     n++;

}
```

6.7 Arrays

Arrays can be created with any primitive type.

There are two basic ways to allocate an array:

```
int[] a(length);
array<int> a(length);
```

To access the array:

a[10] = 200;

Initialization
When declaring arrays it is possible to define the initial size of the array by passing the length as a parameter to the constructor. The elements can also be individually initialized by specifying an initialization list.

```
array<int> a;           // A zero-length array of integers
array<int> b(3);        // An array of integers with 3 elements
array<int> c(3, 1);     // An array of integers with 3 elements, all set to 1 by defau
array<int> d = {,3,4,}; // An array of integers with 4 elements, where the second and
```

Multidimensional arrays

```
array<array<int>> a;                     // An empty array of arrays of integers
array<array<int>> b = {{1,2},{3,4}}      // A 2 by 2 array with initialized values
array<array<int>> c(10, array<int>(10)); // A 10 by 10 array of integers with uninitia
```

Example:

```
int[] lut(256);

lut[0] = 0;
for (int n = 1; n< 256; n++)
{
      lut[n] = n+lut[n-1];
}

for (int n = 0; n< 256; n++)
{
      traceint(lut[n]);
}
```

Operators

```
= assignment
[] index operator
==, != equality
```

Methods

```
uint length() const;
void resize(uint);
void reverse();
void insertAt(uint index, const T& in);
void insertLast(const T& in);
void removeAt(uint index);
void removeLast();
void sortAsc();
void sortAsc(uint index, uint count);
void sortDesc();
void sortDesc(uint index, uint count);
int find(const T& in);
int find(uint index, const T& in);
The T represents the type of the array elements.
```

Example:

```
int main()
{
  array<int> arr = {1,2,3}; // 1,2,3
  arr.insertLast(0);        // 1,2,3,0
  arr.insertAt(2,4);        // 1,2,4,3,0
  arr.removeAt(1);          // 1,4,3,0
  arr.sortAsc();            // 0,1,3,4
  int sum = 0;
  for( uint n = 0; n < arr.length(); n++ )
    sum += arr[n];

  traceint(sum);
}
```

6.8 math

List of standard math functions supported by the script

float cos(float)
float sin(float)
float tan(float)

float acos(float)
float asin(float)
float atan(float)
float atan2(float,float)

Hyperbolic
float cosh(float)
float sinh(float)
float tanh(float)

float log(float)
float log10(float)

float pow(float, float)
float sqrt(float)

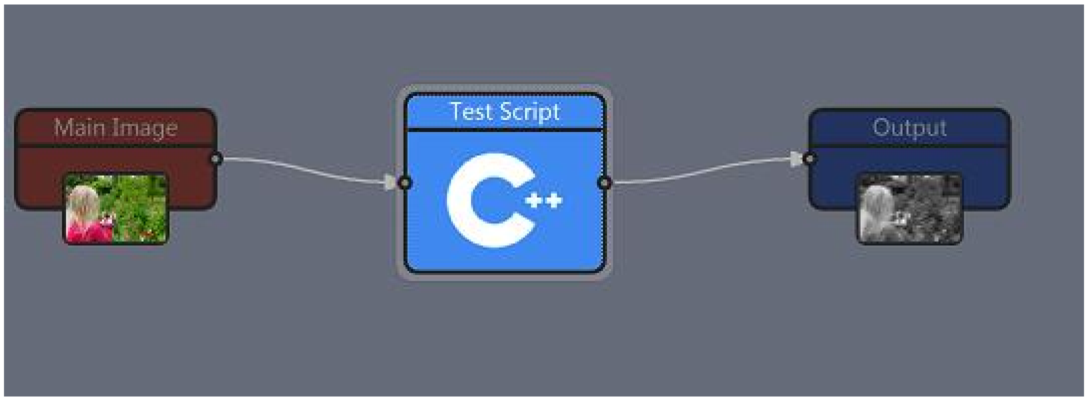
float ceil(float)
float abs(float)
float floor(float)
float fraction(float)

Min, Max and Clamp
float min(float, float)
float max(float,float)
float clamp(float x, float low, float high)
uint clamp255(uint)

6.9 Script header and UI

Script header can specify visual settings for the the object in the comment section. This header is not required nor all settings have to be specified

```
//##NAME:Test Script
//##DESCRIPTION:Just Test
//##INPUTS:1
//##BGCOLOR:2B74DB
//##TEXTCOLOR:FFFFFF
```



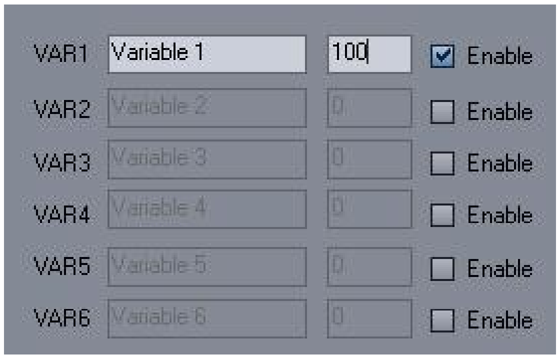
Colors are in HEX format. A quick example of few colors

```
yellow      00DDFF
red         0000FF
orange      0066FF
blue        FF0000
light green 50FFC5
dark green  004912
light blue  FFE25F
purple      FF2897
black       000000
white       FFFFFF
```

Please note the header is in the comment section, it is not processed by the script, but it is parsed by the dialog after each compilation. Do not remove the comment tags.

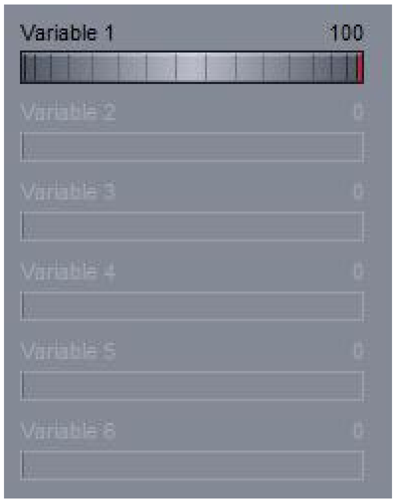
UI Variables
There are 6 free UI variables that can be called from the script.
They all are in the range 0..100 and return float value

You can define the name and the default values in the script interface:



VAR1	Variable 1	100	✓ Enable
VAR2	Variable 2	0	☐ Enable
VAR3	Variable 3	0	☐ Enable
VAR4	Variable 4	0	☐ Enable
VAR5	Variable 5	0	☐ Enable
VAR6	Variable 6	0	☐ Enable

This will then appear in UI properties as:



You can use the values from the sliders as global variables with names VAR1...VAR6

Example:

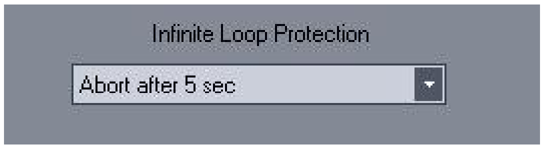
```
float opacity = VAR1/100.0;
```

the opacity will be in range 0...1.0

Define the default values in the script header comment section (optional).
If you want to post the script as a text on the web, you may want to define the default values and names in the script header so they will be set automatically.
After the compilation the comment section will be parsed and the default values set in the appropriate boxes. The syntax example is

```
//##VAR1:50
//##VAR1_NAME:Opacity
```

Infinite Loop Protection
To make sure the script will not lock down the software with an infinite or very long loop there is a protection feature that will terminate the script if it runs longer than the predefined time.
This is valid only during the preview phase (in reactor interface) and debugging phase (in script window).
The protection is removed during final export (because the export on full image may be potentially much longer).



In any case if the script takes more than 5-10 sec to calculate the preview, it is advisable that you think about writing a native plug-in for reactor.

Speed
The script itself runs very fast but it is still an interpreter. The ratio is about 15:1 compared to a native code. (as most interpreters go, that is actually very fast)
That means a looping through image pixels that would take 30ms using native code will take about

450ms using the script.

6.10 Appendix: Object Handles

Reactor Script uses c++ syntax. However for security reasons there is **one** notable difference.

There are No Pointers
Reactor script doesn't use pointers, Instead it uses **object handles**.

What is object handle?
In c++ a pointer would look like this:

```
image* img = new image(200,200);
```

In Reactor Script we use Object Handle for similar purpose:

```
image@ img = @image(200,200);
```

Note the difference:
- instead of * we use @
- there is no arrow ' ->' operator with object handles, just dot '.' operator like with normal object. (ex: `img.Blur(4.5);`)
- we use convention of writing @ at both left and right side of assignment
- there is no new or delete allocation/de-allocation, the script objects are automatically reference counted

Reactor Script uses automatic reference counting which means once the object handle is assigned, the script will keep the referenced object alive.

Object Assignment vs Handle Assignment
object handle assignment:
`myclass@ p2 = p1;`
This creates an object handle p2 that references the object p1. Any further using p2 or p1 is interchangeable, they both access the very same data, we still have only one copy of the data in memory

and assignment operator (if provided by the myclass):
`myclass p2 = p1;`
This physically COPY the data from p1 to p2 so we end up having two copies of the same data

Note:
For this reason the ***image*** class in Reactor Script does not provide assign (=) operator to avoid the mistake between assigning object handles and copying object data which would be costly during processing of large images. If you try to assign image object instead of object handle you will get compiler error.

Assigning object handles:

```
CMyClass object; // we created object
CMyClass @handle = object; // handle to the object
CMyClass object2 = object // a new copy

object.method();
handle.method(); // calls the method on the same data as object
object2.method();  /// we work on a copy of the data
```

6.11 Appendix: Parameter references

The object handles are described as a "safe" pointers that can be passed by functions. But we can as easily use the c++ parameter references in functions call.

Calling function using object handles (aka pointers):

```
void function_handle(image @img)
{
....do something to the img
}

image@ img = @image(INPUT);

function_handle(@img);
```

Using c++ references (Note: function parameter uses & instead @)

```
void function_reference(image &img)
{
....do something to the img
}

image img(INPUT);

function_reference(img);
```

In both cases the result will be the same.

When to use object handles and references?
As with C++ it may come down to anybody preference or need.

In general you should use reference where you can and object handle where you must.
For example if a function creates an object inside, the only way to safely return that object for further usage is by using object handle. (similar to the way a pointer will be used in c++)

```
image@ newimage()
{
      image@ img = @image(INPUT);
      return @img;
}

void main()
{
      // get the image
      image@ myimage = newimage();

      // sample function call
      desaturate(myimage);

      // send the image to output
      myimage.SetOutput();
}
```

6.12 Examples

Here are few quick and simple examples

6.12.1 Simple

Simple example (desaturates image)

```
void main()
{
      // get the image from input socket
      image img(INPUT);


      int width = img.width;
      int height = img.height;

      for (int y=0; y<height; y++)
      {
            for (int x=0; x<width; x++)
            {

                  // one way to get color from pixel
                  RGBQUAD color = img.RGBQUAD(x,y);

                  // average the colors
                  int desaturate = (color.rgbRed+color.rgbGreen+color.rgbBlue)/3;


                  // safest and fastest way to set color to pixels
                  //(clamps the value to 0..255)

                  img.SetRGB(x, y,desaturate,desaturate,desaturate);


            }

      }


      // send the image to output
      img.SetOutput();

}
```

6.12.2 Global Function

Example of calling a global function, same as simple image except the processing is in a global function

```
void desaturate(image &img)
{
      int width = img.width;
      int height = img.height;

      for (int y=0; y<height; y++)
      {
            for (int x=0; x<width; x++)
            {

                  // one way to get color from pixel
                  RGBQUAD color = img.RGBQUAD(x,y);

                  // average the colors
                  int desaturate = (color.rgbRed+color.rgbGreen+color.rgbBlue)/3;


                  // safest and fastest way to set color to pixels
                  //(clamps the value to 0..255)

                  img.SetRGB(x, y,desaturate,desaturate,desaturate);


            }

      }

}



void main()
{
      // get the image from input socket
      image img(INPUT);

      // call the function
      desaturate(img);


      // send the image to output
      img.SetOutput();

}
```

6.12.3 Class

Using a class and UI sliders. The first commented lines are the header that gets parsed and sets the default values for the sliders. You need to include them.

```
//##INPUTS:1
//##VAR1:21
//##VAR2:71
//##VAR3:7
//##VAR1_NAME:Red Influence
//##VAR2_NAME:Green Influence
//##VAR3_NAME:Blue Influence

class MyLuminance
{
      // constructor
      MyLuminance()
      {
            m_fR = 0.2126;
            m_fG = 0.7152;
            m_fB = 0.0722;
      }
      // A class method
      void Process(image &img)
      {
            int width = img.width;
            int height = img.height;


            for (int y=0; y<height; y++)
            {
                  for (int x=0; x<width; x++)
                  {

                        // one way to get color from pixel
                        RGBQUAD color = img.RGBQUAD(x,y);

                        // average the colors
                        int luminance = color.rgbRed*m_fR
                                          +color.rgbGreen*m_fG
                                          +color.rgbBlue*m_fB;


                        img.SetRGB(x, y,luminance,luminance,luminance);

                  }

            }
      }
      void SetCoeffs(float r, float g, float b)
      {
            // together the sum of coefficients have to be 1.0
            float sum =r+g+b;

            //avoid 0 or we will receive division by zero excepion
            if (sum==0.0)
            {
                  sum = 1.0;
            }

            float scale = 1.0/sum;


            m_fR = r*scale;
            m_fG = g*scale;
            m_fB = b*scale;

      }

       // Class properties
   float m_fR;
   float m_fG;
   float m_fB;
}


void main()
{

      MyLuminance lum;

      // get the image from input socket
      image img(INPUT);


      // get coefficients from the sliders
      lum.SetCoeffs(VAR1/100.0, VAR2/100.0, VAR3/100.0);

      lum.Process(img);

      // send the image to output
      img.SetOutput();

}
```

6.12.4 Two inputs

This examples emulates Blend Layers multiply
To get two node input object, we have to specify ##INPUTS:2 in the header, compile the script, close the script window and connect the second node

This sample uses both references and object handle as a return from the multiply function. Please note we are changing img1 in the function and returning object hanle (aka safe pointer) to it. imgout is in fact an object handle of img1. We could as easily use `img1.SetOutput();` in the main function.
You have to include the first two commented lines so the object switches to 2 node input.

```
//##NAME:Multiply Images
//##INPUTS:2

image@ multiply(image &img1, image &img2)
{
      int width = img1.width;
      int height = img1.height;

      //both images are the same size


      for (int y=0; y<height; y++)
      {
            for (int x=0; x<width; x++)
            {


                  pixel color1 = img1.Pixel(x,y);
                  pixel color2 = img2.Pixel(x,y);


                  // Multiply colors
                  color1.r = color1.r*color2.r/255;
                  color1.g = color1.g*color2.g/255;
                  color1.b = color1.b*color2.b/255;

                  img1.Pixel(x, y) = color1;
                  // if you need to clamp the values, use SetRGB instead


            }

      }

      return img1;

}
```

```
void main()
{
      // get the image from input socket
      image img1(INPUT1);
      image img2(INPUT2);

      // call the function
      image@ imgout = multiply(img1,img2);


      // send the image to output
      imgout.SetOutput();

}
```

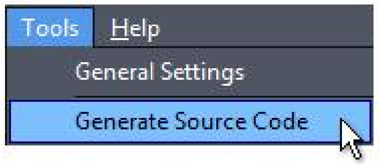
7 Plugin SDK

Plug-in SDK is a way how to create native external plug-ins for Photo-Reactor.
Unlike Script, plug-ins run on native code and the author has to provide both 32 and 64 bit versions.

First you need to download the SDK. There is nothing to install, the zip package has a few samples and modifying those samples user can create their own plug-ins.
Download:
http://www.mediachance.com/reactor/pluginSDK.zip

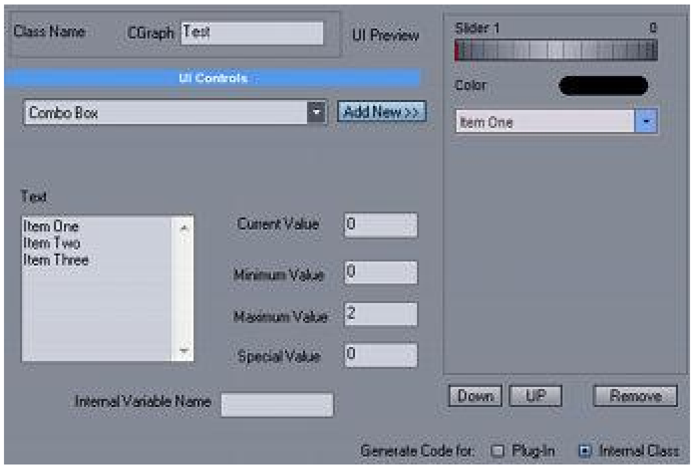
plugin - a basic plugin that desaturate the image with a slider for strength and check box for inversion. (Similar to the Desaturate effect in Reactor)
pluginbind - example how to create a binding plug-in with a simple on-the-workspace slider object that can control value of other objects (Similar to the Slider object in Reactor)
pluginrect - an example of drawing semi-transparent rectangle on the image and the calculation necessary for preview cropping (Similar to the Simple Shape object in Reactor)

Source code generator
To make this process super easy we build-in a source code generator that generates the main plugin cpp file.



The Photo-Reactor can generate the necessary Source code for the plug-in UI and class which will set up the interface and settings. You will receive generated plugin.cpp file that can be simply substituted in the example projects - that's all.
The steps are:
- Copy "plugin" folder in the examples a new folder.
- Go to Photo-Reactor menu Tools - Generate Source code
- Set up the interface for your parameters (you can also manually add/change them later, but it is easier from the interface) and set up the global settings what kind of plug in it is (simple plugin has usually nothing set)
- Generate Code - creates plugin.cpp
- Replace the plugin.cpp in your copied "plugin" folder
- Change identifier string in the plugin.cpp to be unique (reverse domain is usual practice)
- Put your code in Process_Data that gives you pointer to input and output buffers and the parameters from interface
- that's it, compile, put the dll in plugins folder and use it

The UI building part let's you add various controls and set their limits and values.
The controls will appear in the right panel when the object is selected. A corresponding code will be generated where necessary.



Class Name CGraph Test

Combo Box

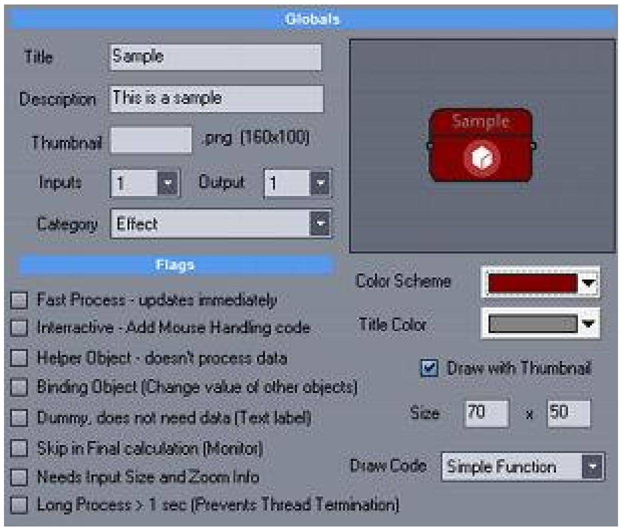
Text
Item One
Item Two
Item Three

Current Value 0
Minimum Value 0
Maximum Value 2
Special Value 0
Internal Variable Name

Slider 1 0
Color
Item One

Generate Code for: ☐ Plug-In [✓ Internal Class]

Global section will let you set the properties and flags and the look of the object on the workspace. The flags set other important properties about plug-in and are fully described in the plugin.cpp file.

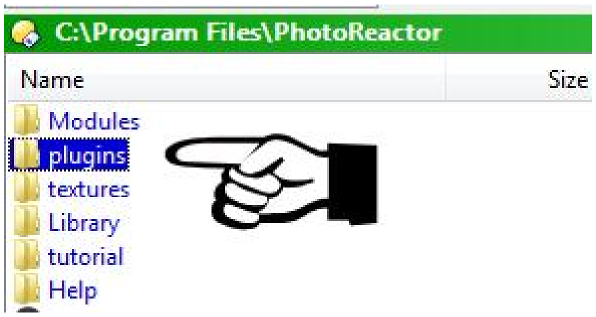


Title Sample
Description This is a sample
Thumbnail .png (160x100)
Inputs 1 Output 1
Category Effect

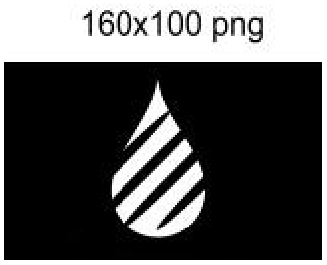
☐ Fast Process - updates immediately
☐ Interractive - Add Mouse Handling code
☐ Helper Object - doesn't process data
☐ Binding Object (Change value of other objects)
☐ Dummy, does not need data (Text label)
☐ Skip in Final calculation (Monitor)
☐ Needs Input Size and Zoom Info
☐ Long Process > 1 sec (Prevents Thread Termination)

Color Scheme
Title Color
☑ Draw with Thumbnail
Size 70 x 50
Draw Code Simple Function

Compiled plugin dll's go to the *plugins* folder in the Reactor application folder.



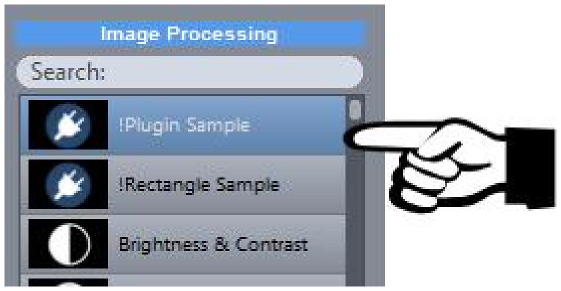
Icon
If you want customized icon you can create PNG file 160x100 that will be named same as the plugin dll and placed in the same plugins folder.



Unique string identifier
Every plugin needs an unique string identifier set in the GetPluginID() function which is in form of your reverse domain such as: "com.mediachance.myfirstplugin" (put your own domain there or something that identifies you)
This way the string will be unique and the project can easily find the correct plugin when saved and loaded regardless of its name or dll name.

```
// This MUST be unique string for each plugin so we can save the data
extern "C" __declspec(dllexport) char* GetPluginID()
{
      return "com.mediachance.testplugin";
}
```

Plug-in name
This name will appear in the object list

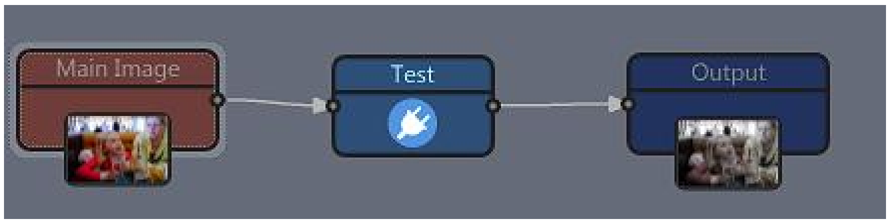


```
// this is the name that will appear in the object library
extern "C" __declspec(dllexport) char* GetPluginName()
{
      return "!Plugin Sample";
}
```

Category
CATEGORY_EFFECT
CATEGORY_BUILDING_BLOCK

```
// category of plugin, for now the EFFECT go to top library box, everything else goes
extern "C" __declspec(dllexport) int GetCategory()
{
      return CATEGORY_EFFECT;
}
```

Title and description
Title will appear on the box in the workspace



```
//this is the title of the box in workspace. it has to be short
const char* GetTitle () const
{
      return "Test";
}

// this will appear in the help pane, you can put your credits and short info
const char* GetDescription () const
{
      return "This is my first plugin";
}
```

Flags
The UI interface builder (Generate Source Code) will fill those flags according to the checkboxes

see the interface builder
ex: nFlag = FLAG_FAST_PROCESS | FLAG_HELPER;

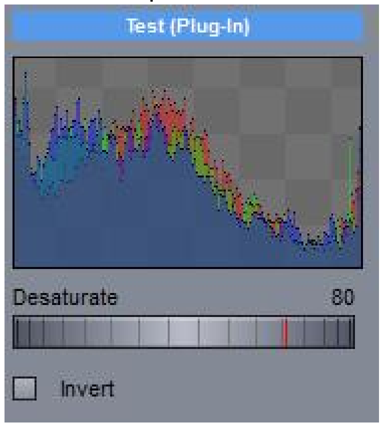
FLAG_NONE same as zero   Default, no other flags set
FLAG_UPDATE_IMMEDIATELY It is very fast process that can update immediately. When user turns the sliders on UI the left display will update
Use Update Immediately only for fast and single loop processes, for example Desaturate, Levels.
FLAG_HELPER   It is an helper object. Helper objects will remain visible in Devices and they can react to mouse messages. Example: Knob, Monitor, Bridge Pin
FLAG_BINDING   Binding object, attach to other objects and can change its binding value. It never goes to Process_Data functions. Example: Knob, Switch, Slider
FLAG_DUMMY   It is only for interface but never process any data. Never goes to Process_Data functions. Example: Text note
FLAG_SKIPFINAL   Process data only during designing, doesn't process during final export. Example: Monitor, Vectorscope
FLAG_LONGPROCESS   Process that takes > 1s to finish. Long Process will display the Progress dialog and will prevent user from changing values during the process.
FLAG_NEEDSIZEDATA   Process need to know size of original image, the zoom and what part of image is visible in the preview. When set the plugin will receive SetSizeData
FLAG_NEEDMOUSE   Process will receive Mouse respond data from the workplace. This is only if your object is interactive, for example Knob, Slider

```
int GetFlags ()
{
      // it is fast process
      int nFlag = FLAG_UPDATE_IMMEDIATELY;
      return nFlag;
}
```

UI Parameters
UI parameters is a structure of the UI control. The **Generate Source code** will fill those UI parameters for you in the GetUIParameters function

```
int GetUIParameters (UIParameters* pParameters)
{
      // label, value, min, max, type_of_control, special_value
      // use the UI builder in the software to generate this
      AddParameter(0,"Desaturate",80.0, 0.0, 100, TYPE_SLIDER, 0 );
      AddParameter(1,"Invert",0.0, 0.0, 1, TYPE_CHECKBOX, 0 );
      return 2;
}
```

which corresponds to:



See definition of the UIParameters in IPlugin.h:
All controls always work with double values to simplify things (even checkboxes or index combo boxes)

```
typedef struct UIPARAMETERS
{
      // 0...100
      char m_sLabel[255];
      double m_dValue;
      double m_dMin;
      double m_dMax;
      double m_dSpecialValue;
      int m_nType;

} UIParameters;
```

Use the Parameters
Any function that you can change (and there are only few) in SDK will carry the current parameters. The most obvious function is the main process

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UI
```

There are few macros to get the value from the ID. so in our case to get desaturate value

```
double dEffect = GetValue(0)/100.0;
```

or using the verbal define
```
double dEffect = GetValue(PARAM_DESATURATE)/100.0;
```

What the macro GetValue does is simply use the structure
```
#define GetValue(N) (pParameters[N].m_dValue)
```

That means we can expand it like this and it would be valid:
```
double dEffect = pParameters[PARAM_DESATURATE].m_dValue
```

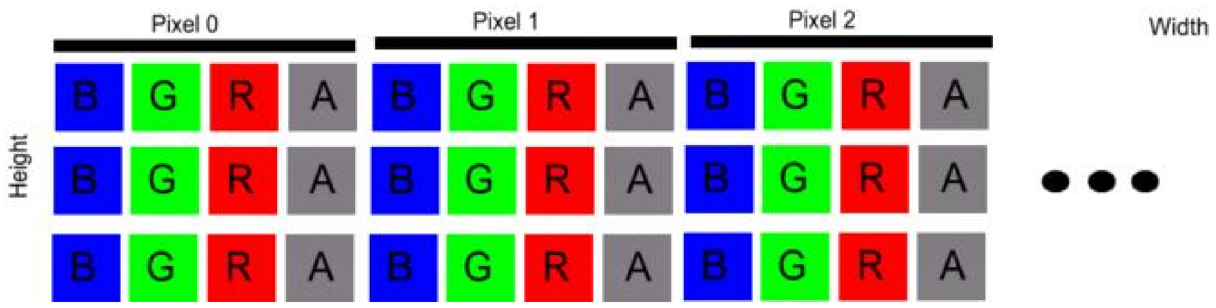
We can do the same for a checkbox, because when checkbox is on it returns maximum value (1.0) when it is off it returns minimum value (0.0) but casting double to bool is really ugly: BOOL bInvert = (BOOL) GetValue(1); would work, but it is potentially a bug source so there is another macro that simply compare the value with the maximum value and return BOOL
```
BOOL bInvert =  GetBOOLValue(1);
```

7.1 Main Processing

The processing function is `Process_Data and Process_Data2 for plugins with 2 inputs.`

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UI
```

The data of the input and output buffers are 8-bit per channel (BYTE or unsigned char) in the sequence B,G,R,A



Each of the little box can have value 0...255 and it is unsigned char.

So a line of pixels in the buffer is is **width * 4** of unsigned chars also called **rowstride**
The number of unsigned chars in the image is **width*height *4**

Incidentally the image is also upside down which usually doesn't play a role in a normal processing.

As the example shows, simple desaturating image by looping through height and width:

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in,
                          int nWidth, int nHeight,
                          UIParameters* pParameters)
{

      for (int y = 0; y< nHeight; y++)
      {
            for (int x = 0; x< nWidth; x++)
            {

                  int nIdx = x*4+y*4*nWidth;

                  int nR = pBGRA_in[nIdx+CHANNEL_R];
                  int nG = pBGRA_in[nIdx+CHANNEL_G];
                  int nB = pBGRA_in[nIdx+CHANNEL_B];


                  int nA = (nR+nG+nB)/3;

                  pBGRA_out[nIdx+CHANNEL_R] = nA;
                  pBGRA_out[nIdx+CHANNEL_G] = nA;
```

```
                    pBGRA_out[nIdx+CHANNEL_B] = nA;

                }

            }
        }
```

Note we use defines for the channels to make it more readable. CHANNEL_R is for example value of 2
Most typical image processing can be handled this way. In some cases you would need to know also the zoom of the preview window (for example a blur radius needs to be multiplied by the zoom to always show correct amount of the blur regardless of the zoom). You can get the zoom value from the SetSizeData function. (See below)

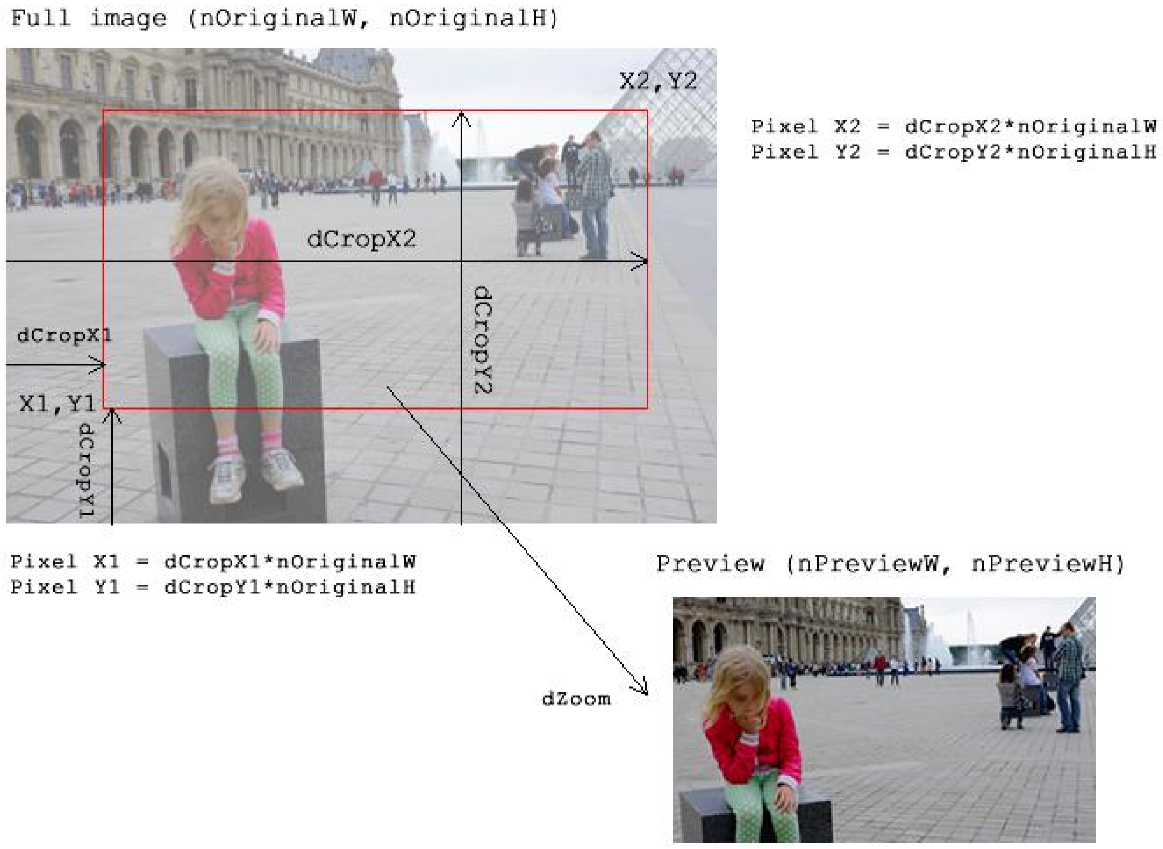
Frame Position (more complicated)
If your processing depends on a position in the frame (for example drawing a rectangle at a certain position) it become more complicated as you would need to know the position (crop) of the preview in the final frame and the zoom value. If we don't take that into account then the rectangle will be always drawn at the same position in the preview image regardless of what part of the image the preview displays.

The preview image (what we receive in the Process_Data) can be a crop from the original image and it is displayed with a zoom.
For that we have SetSizeData function that receives information about the full size of the image, the preview size, relative crop positions and zoom.

```
void SetSizeData(int nOriginalW, int nOriginalH, int nPreviewW, int nPreviewH,
                double dCropX1, double dCropY1, double dCropX2, double dCropY2,
                double dZoom)
```

This gives you info about the size and position of the current preview within the original image.
See **pluginrect** sample that draws a rectangle at certain position and fully takes the zoom and the crop into account..

Crop coefficients
The crop coefficients are relative in value 0...1 and they are meant to be multiplied by the width and height to get an absolute value.


Full image (nOriginalW, nOriginalH)
X2,Y2
Pixel X2 = dCropX2*nOriginalW
Pixel Y2 = dCropY2*nOriginalH
dCropX2
dCropY2
dCropX1
X1,Y1
dCropY1
Pixel X1 = dCropX1*nOriginalW
Pixel Y1 = dCropY1*nOriginalH
Preview (nPreviewW, nPreviewH)
dZoom

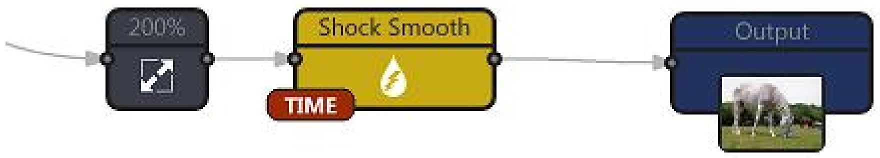
Example:
we want to know where is the preview left size relative to the main image:

```
nleft = dCropX1*nOriginalW*dZoom
```

8 FAQ

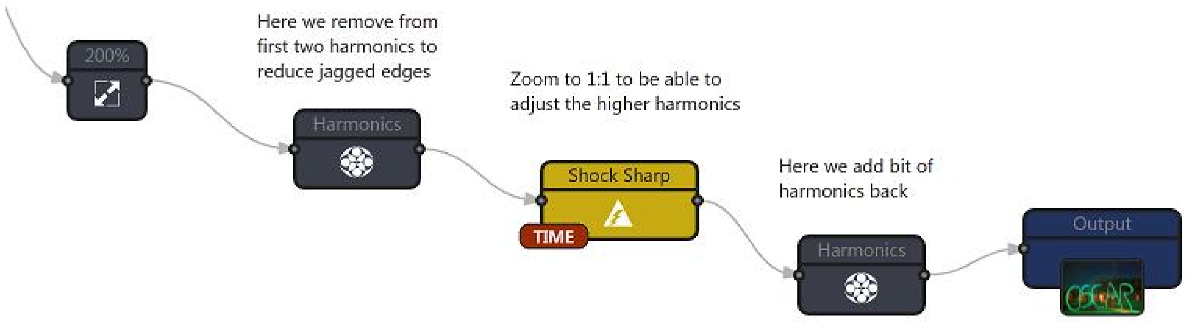
If things don't work properly it could be due various issues. Also some issues may simply be the way things work.

The result image doesn't look exactly like the preview
This is likely due to the nature of some object. Not all effects are perfectly scalable and if there is big difference between the size of the preview and result image the result may be off. A typical problematic effects are the ones that work localy with pixel groups, like Smoothing, Segmentation etc..
The resolution is to scale down the Main image or use Scale If object just after the Main Image. An example: image of size 4500x3200 pixels may render a complex effect not as desired. Resizing it to 35% or setting Scale if to 1500 would render the image much closer to desired output.
If a high pixel output size is desired the best approach would be to scale the image down, then scale it up before the Output. There are various scaling methods depending on the type of image. For a photo, you may try just simple resize, but choose interpolation that doesn't enhance pixel jaggines.
For paint type of effect, one can place a resize object using Lanczos interpolation followed by Shock Smoothing with a toned down settings.



Object Crumbs:
ID3, 41, 1, 0, 233, 37, 0, P, 200.000, 0.000, 0.000, 0.000, 1.000
ID4, 42, 3, 0, 376, 37, 0, P, 50.000, 0.410, 1.000, 0.000, 0.803, 80.000, 0.000, 1.000, 1.000

Art Resize (for graphics, not for gradients and photos)
Another trick up-sizing graphics image (like water color effect) and removing jagged pixel edges


200%
Here we remove from first two harmonics to reduce jagged edges
Harmonics
Zoom to 1:1 to be able to adjust the higher harmonics
Shock Sharp
TIME
Here we add bit of harmonics back
Harmonics
Output

Object Crumbs:
ID105, 41, 1, 0, 220, 148, 0, P, 200.000, 0.000, 0.000, 6.000, 1.000
ID107, 74, 105, 106, 331, 154, 0, P, -32.941, -23.529

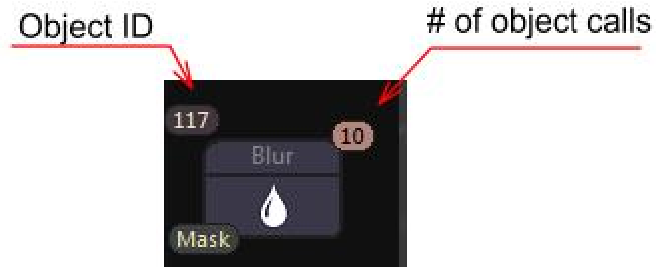
ID109, 43, 107, 0, 457, 156, 0, P, 1.000
ID110, 74, 109, 0, 601, 155, 0, P, 28.235, 11.765

The result image is very different than the preview
(for example colors are completely off, etc..)
Some of the objects had not been tested for every situations and can have bugs. Orr it can be an extreme case of the above issue.
It is best to visit www.mmbforums.com and post the Snapshot (File - Save Flow Snapshot) while explaining what you trying to do. The snapshot has also embedded flow, which other users can open and examine.
Also look around if the error had not been already posted.

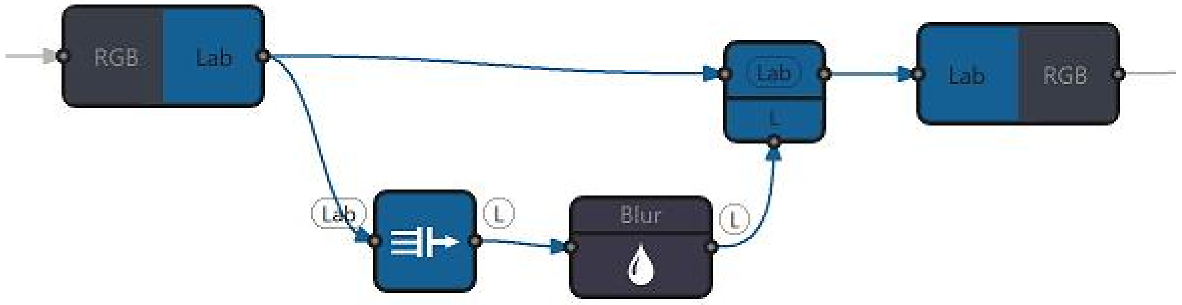
The preview is ok, but the result is empty (shows a chessboard)
It means the final processing failed at some point or some objects were freed before their final time. The best is to send us the flow.

The effect is not as desired
It is best to visit www.mmbforums.com and post the Snapshot (File - Save Flow Snapshot) while explaining what you trying to do. The snapshot has also embedded flow, which other users can open and examine.

What are the numbers on the right monitor around the object pictogram.
Those numbers can be used to debug the issues (but most likely for the author).
The Object ID is the issued object, also shows which object was added when (higher number means the object was added later)
Number of object calls. How many times the object is called during calculation. It should be equal or higher than number of outgoing nodes. Sometimes it may be much higher if the flow is complex.

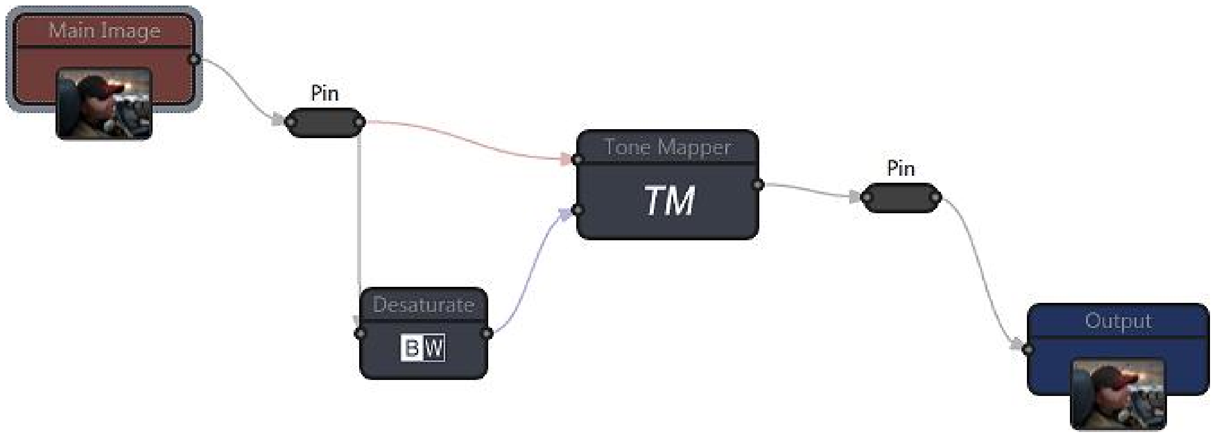


I want to save Lab colors so I added RGB-Lab object before Output but it didn't work as desired.
This is not how the space conversion objects are used. They are for internal conversion, for example we want to apply certain effects not in RGB but in different color space
Ex: apply blur to "a" in Lab color space: A typical configuration would be to put RGB/Lab, then Channel split to extract "a" channel, apply blur to it, then use Channel Merge to merge it back with the Lab. After that use another RGB/Lab object (in reverse) to go back to RGB.



Object Crumbs:
ID120, 30, 1, 0, 184, 90, 0, P, 1.000, 0.000
ID121, 30, 124, 0, 619, 97, 0, P, 1.000, 0.000
ID122, 31, 120, 0, 316, 174, 0, P, 0.000
ID123, 12, 122, 0, 413, 179, 0, P, 2.500, 1.000
ID124, 32, 120, 123, 506, 105, 0, P, 1.000

How does Tone Mapper object work with second input?
The optional second input is the Mask for the tone mapper - the mask is used to determine which areas of the main image (the image that goers to the the top input) are dark and which are light. While it seems trivial, the trick is that we can process the mask a different way (for example smooth it or change levels) and that will dramatically affect the output of the tone mapper. In all normal cases the mask is derived from the input image itself but how we process it is up to us.
In the following example we use desaturate object in the mask line - by changing the R,G,B color weights of the desaturation we can change the final output tone of the Tone Mapper.



Object Crumbs:
ID4, 108, 8, 6, 329, 80, 0, P, 2.54, 1.56, 2.68, 1.61, Y
ID6, 2, 8, 0, 208, 168, 0, P, 100.00, T, 0.444, 0.201, 0.621
ID8, 8, 1, 0, 168, 68, 0, P
ID9, 8, 4, 0, 489, 110, 0, P

9 Object Crumbs

Object Crumbs is a short text that represent sequence of objects, their parameters and also links between them. By pasting the sequence back you can quickly create all the objects with the parameters set and their internal links.

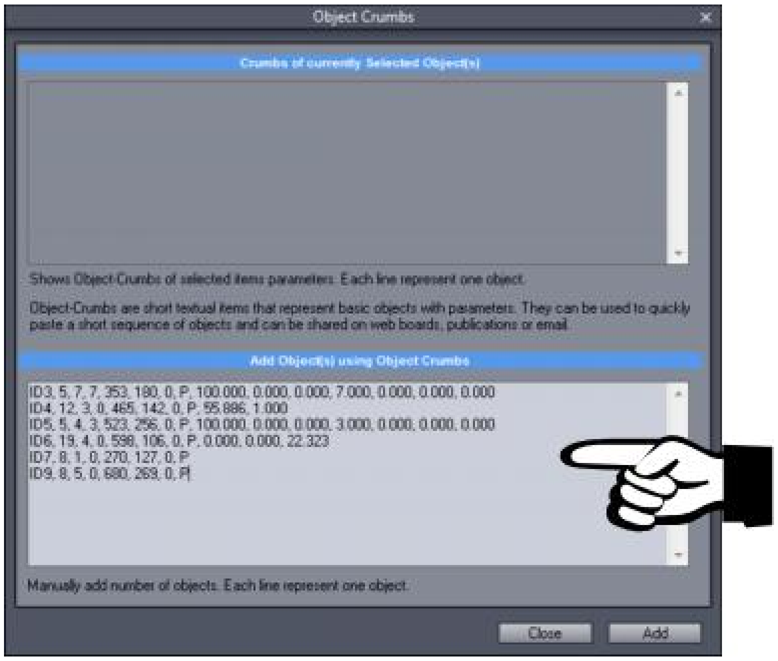
This is a great helper for web boards, email or even this manual where you can share some small part of the effect using a simple text.

Instead of explaining how to add such and such object then listing all the parameters of the object we can post only a short text that can be pasted to Object Crumbs window and would create the objects for us.

Let's just try it:

Open menu **Object - Object Crumbs**
and paste in the bottom window following text (just copy it from here CTRL+C and paste it in the window CTRL+V)

```
ID3, 5, 7, 7, 353, 180, 0, P, 100.000, 0.000, 0.000, 7.000
ID4, 12, 3, 0, 465, 142, 0, P, 55.886, 1.000
ID5, 5, 4, 3, 523, 256, 0, P, 100.000, 0.000, 0.000, 3.000
ID6, 19, 4, 0, 598, 106, 0, P, 0.000, 0.000, 22.323
ID7, 8, 1, 0, 270, 127, 0, P
ID9, 8, 5, 0, 680, 269, 0, P
```


Object Crumbs
Crumbs of currently Selected Object(s)
Add Object(s) using Object Crumbs
Close
Add

Shows Object-Crumbs of selected items parameters. Each line represent one object.

Object-Crumbs are short textual items that represent basic objects with parameters. They can be used to quickly paste a short sequence of objects and can be shared on web boards, publications or email.

```
ID3, 5, 7, 7, 353, 180, 0, P, 100.000, 0.000, 0.000, 7.000, 0.000, 0.000, 0.000
ID4, 12, 3, 0, 465, 142, 0, P, 55.886, 1.000
ID5, 5, 4, 3, 523, 256, 0, P, 100.000, 0.000, 0.000, 3.000, 0.000, 0.000, 0.000
ID6, 19, 4, 0, 598, 106, 0, P, 0.000, 0.000, 22.323
ID7, 8, 1, 0, 270, 127, 0, P
ID9, 8, 5, 0, 680, 269, 0, P
```

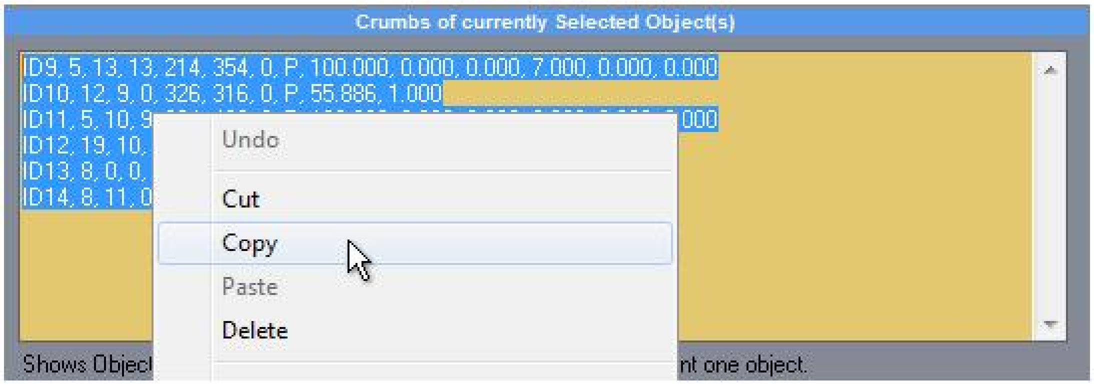
Manually add number of objects. Each line represent one object.

Then press Add button.

That's all!

How to get the Crumbs?
The top yellow window will display crumbs of currently selected object, multiple selection or group.
To get the Object Crumbs, select couple of objects and go to Object Crumbs window.
Then select the text in the top window and press Control+C to copy it to clipboard.

There you have it.


Crumbs of currently Selected Object(s)

```
ID9, 5, 13, 13, 214, 354, 0, P, 100.000, 0.000, 0.000, 7.000, 0.000, 0.000
ID10, 12, 9, 0, 326, 316, 0, P, 55.886, 1.000
ID11, 5, 10, 9                                                        000
ID12, 19, 10,
ID13, 8, 0, 0,
ID14, 8, 11, 0
```

Shows Object   nt one object.

Note:
The Object Crumbs carry only the basics parameters that help you to create the objects quickly. They don't carry any string information (such as customized labels), or any other additional data (mask, curves). Also few non-processing helper objects such as label or text will be ignored.
To capture content of a group, select the group. If you want to capture only a part of a group, go to **Edit Inside** (tab) then select the items.

The crumbs are not very user editable (well, they are in theory but it is better not to touch them)

Trick:
After the letter P there are parameter crumbs. If you don't care about the parameters, you may make the crumb shorter by removing everything after the P (but keep the P).
Example:
`ID5, 5, 4, 3, 523, 256, 0, P, 100.000, 0.000, 0.000, 3.000, 0.000, 0.000, 0.000`
`shorter`
`ID5, 5, 4, 3, 523, 256, 0, P`
this will create the same object but all parameters will be default

Tip:
You can use Object-Crumbs to create multiple numbers of the same object or block of objects (just paste the same crumbs few times in the window). The new objects will be placed neatly in a grid.
You can also copy a selection between two projects (without saving/loading from library) using the Crumbs.