Photo-Reactor SDK

By Andy Dansby
ver 6-28-13

Index

Introduction

Photo-Reactor is a new generation of image editing created and written by Media Chance available at www.mediachance.com . It includes many image processing effects built in and is a powerful way to edit and enhance your images with Nodal editing. However, one of the most powerful features built into this new editor is a built in SDK for filter creation. This document aims to assist with filter creation as well as an introduction to image processing.

The goal of this document is to be a overall comprehensive guide, but there are no promises on being a complete guide.

This document makes some assumptions.

1) That you have a copy of Photo-Reactor and the SDK.
2) You have a C or C++ compiler.
3) Have some basic programming knowledge in C or C++ or are willing to learn
4) An Idea

The first is rather easy to accomplish, simply visit the Media Chance website www.mediachance.com and download and install Photo-Reactor.

The second is rather easy as well, Visual Studio Express is a C and C++ compiler available free from Microsoft. At the time of this writing it is available from http://www.microsoft.com/visualstudio . Locate the Express version of Visual Studio and download it and install.

Which leads to the third item and certainly the most complex, knowing or willing to learn C or C++. Fortunately, with Photo-Reactor, you do not have to worry about many things, as most of the interface work is done in the main program itself. All you have to worry about is the manipulation of the image data itself.

The final item An Idea, well why program at all unless you have this in the first place. Not to worry, I will for this time, provide an idea for a plug-in already written, Luminance. The example source code provided by Media Chance comes with it's own example idea, “desaturate”. Luminance is not far removed from the Media Chance example.

On the following pages we will go into some of the details of this entire process as well as showing some examples and source code.

What is an image

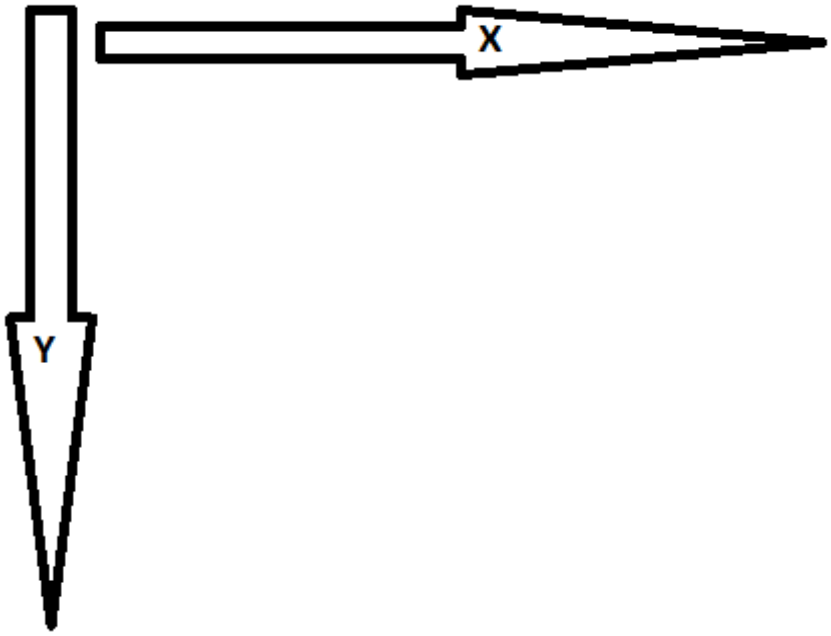
This may seem like a odd question, but before we program, we need to know what we are working with. Yes, an image, is what you see on your screen after you open an image file. However, what is an image to a computer? We need to know what we are working with so we can work on it.

Well, an image is an array. It has height, width and if it is a color image, depth.

Let's picture a color 640x480 image.

Image a spreadsheet 640 cells wide and 480 cells tall. The spreadsheet also has a depth of 3 stacks.

The width is known as X and the height is known as Y in Cartesian coordinates. Each of the 3 stacks represent a color basic, Red, Green Blue.



To picture the spreadsheet analogy better, I've created a sample.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	135	135	136	136	137	137	134	133	133	134	133	132	132
2	136	136	136	137	136	134	132	132	131	134	132	130	131
3	135	135	135	134	133	133	131	129	130	131	130	129	130
4	133	133	133	133	133	132	130	130	128	131	129	128	129
5	131	131	131	131	132	132	131	130	129	131	128	127	128
6	130	130	130	130	131	131	131	130	130	131	129	127	129
7	131	130	130	130	130	130	130	129	130	131	129	128	129
8	131	131	130	130	130	130	129	128	129	131	129	128	130
9	131	130	130	130	131	131	131	130	130	131	130	129	127
10	129	130	129	130	131	131	131	131	132	133	131	129	127
11	129	129	129	130	132	132	132	132	133	133	132	129	128
12	129	129	129	130	132	133	133	133	133	133	131	129	129
13	128	128	128	130	131	132	133	133	132	131	130	129	130
14	130	129	129	131	132	133	134	133	132	131	131	131	133
15	130	130	130	132	133	133	133	132	133	133	133	134	136
16	132	131	132	132	133	133	133	133	135	135	135	138	139
17	132	131	130	129	131	133	133	134	136	136	136	139	142
18	133	131	131	130	132	135	137	137	138	138	140	142	145
19	135	133	132	133	134	136	139	139	139	140	143	144	144
20	138	137	134	134	134	136	137	138	140	142	144	144	143
21	137	137	135	135	134	136	137	139	143	143	144	142	140
22	135	135	134	134	135	137	139	142	147	145	142	139	136
23	135	135	134	135	136	138	141	144	148	143	138	132	128
24	136	136	137	135	136	137	141	143	145	140	131	125	122
25	138	137	136	136	137	139	144	147	144	137	128	123	118
26	135	134	134	134	137	141	145	148	141	133	123	118	115
27	133	132	133	135	139	144	146	146	137	128	118	112	108
28	134	134	134	138	142	146	144	141	132	124	114	105	99
29	135	135	137	142	145	147	142	137	127	120	108	97	90
30	135	136	139	144	148	147	139	134	123	113	99	86	80

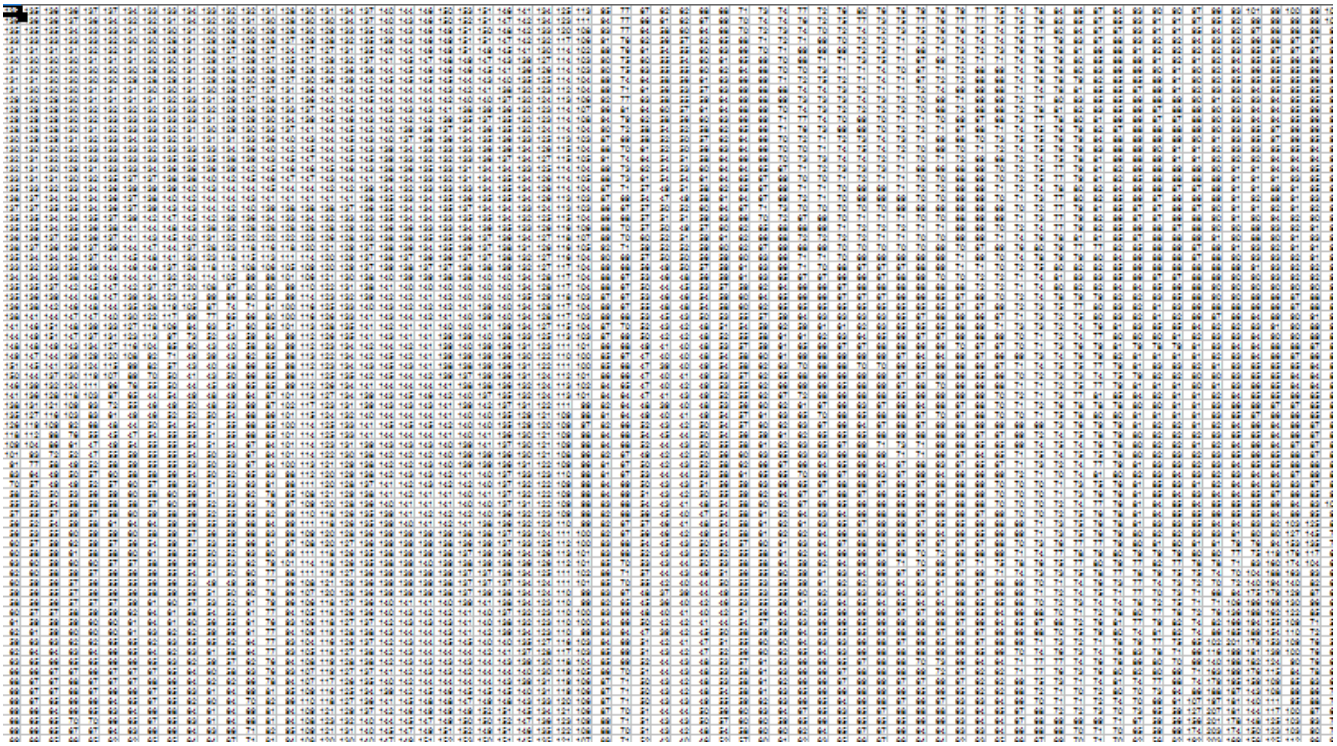
Red Green Blue

Here's a series of numbers, I've named each of the layer or stacks with a color name.

Lets zoom out a bit.

135	135	136	136	137	137	134	133	133	134	133	132	132	131	128	130	131	134	137	140	144	146	150	153	151	146	141	134
136	136	136	137	136	134	132	132	131	134	132	130	131	130	128	130	130	134	137	140	143	146	150	152	151	147	142	134
135	135	135	134	133	133	131	129	130	131	130	129	130	129	128	130	129	133	135	140	143	146	149	151	150	148	142	133
133	133	133	133	133	132	130	130	128	131	129	128	129	128	127	129	128	132	135	139	143	146	149	151	151	147	142	132
131	131	131	131	132	132	131	130	129	131	128	127	128	127	124	127	127	131	135	140	144	146	149	151	149	145	141	130
130	130	130	130	131	131	131	130	130	131	129	127	129	127	125	127	129	132	137	141	145	147	148	149	147	143	139	127
131	130	130	130	130	130	130	129	130	131	129	128	129	128	126	129	132	136	139	144	145	146	146	146	145	141	136	126
131	131	130	130	130	130	129	128	129	131	129	128	130	129	127	130	136	139	142	145	145	145	145	144	143	140	135	125
131	130	130	130	131	131	131	130	130	131	130	129	127	127	131	136	141	143	145	144	144	144	143	142	141	138	132	123
129	130	129	130	131	131	131	131	132	133	131	129	127	128	131	136	142	144	145	144	144	144	142	140	140	137	132	124
129	129	129	130	132	132	132	132	133	133	132	129	128	129	133	137	144	145	144	143	143	143	141	138	139	136	132	123
129	129	129	130	132	133	133	133	133	133	131	129	129	130	134	139	145	146	143	142	142	142	139	135	137	135	132	123
128	128	128	130	131	132	133	133	132	131	130	129	130	133	137	141	144	145	142	140	139	139	137	134	136	135	132	124
130	129	129	131	132	133	134	133	132	131	131	131	133	136	140	143	144	145	143	140	137	136	136	134	135	136	133	125
130	130	130	132	133	133	133	132	133	133	133	134	136	140	142	145	144	145	143	139	134	133	133	133	136	136	134	126
132	131	132	132	133	133	133	133	135	135	135	138	139	143	145	147	144	145	145	139	133	132	132	133	136	137	134	127
132	131	130	129	131	133	133	134	136	136	136	139	142	145	146	146	145	146	143	138	136	135	133	131	134	134	133	125
133	131	131	130	132	135	137	137	138	138	140	142	145	146	147	147	143	144	141	136	134	133	132	131	134	135	134	126
135	133	132	133	134	136	139	139	139	140	143	144	144	145	144	144	142	142	139	134	132	133	133	133	134	135	134	126
138	137	134	134	134	136	137	138	140	142	144	144	143	141	141	141	141	141	138	135	133	134	135	136	135	135	134	126
137	137	135	135	134	136	137	139	143	143	144	142	140	139	138	138	138	137	136	135	134	134	135	137	134	134	133	124
135	135	134	134	135	137	139	142	147	145	142	139	136	134	133	134	132	133	133	135	134	134	133	135	135	134	132	125
135	135	134	135	136	138	141	144	148	143	138	132	128	128	128	128	128	129	132	136	136	133	133	134	136	135	133	126
136	136	137	135	136	137	141	143	145	140	131	125	122	122	122	123	126	128	133	138	138	135	135	136	137	137	134	127
138	137	136	136	137	139	144	147	144	137	128	123	118	116	118	120	121	129	137	138	136	134	135	136	137	137	131	126
135	134	134	134	137	141	145	148	141	133	123	118	115	113	111	114	120	129	137	138	137	136	136	137	137	138	132	127
133	132	133	135	139	144	146	146	137	128	118	112	108	106	105	109	120	129	137	139	138	137	137	138	138	139	132	127
134	134	134	138	142	146	144	141	132	124	114	105	99	98	101	106	121	130	138	140	139	139	138	138	140	140	134	128
135	135	137	142	145	147	142	137	127	120	108	97	90	90	99	110	122	131	138	141	140	140	140	140	140	140	134	128
135	136	139	144	148	147	139	134	123	113	99	86	80	85	99	114	123	132	139	142	142	141	142	140	140	140	135	129
136	138	142	146	148	144	135	128	119	105	87	74	71	81	100	116	125	133	140	143	142	142	142	141	139	140	134	128
138	141	144	147	147	140	130	122	117	99	77	65	66	80	100	116	126	133	141	143	144	142	141	140	139	140	134	128
141	146	151	148	139	133	127	118	106	84	63	51	60	85	101	113	128	135	141	142	141	141	140	140	141	139	134	127
144	149	151	147	137	131	123	113	97	73	52	43	59	84	99	112	126	135	141	143	141	141	139	139	140	138	133	125
148	148	149	143	134	127	116	104	85	60	43	40	58	83	99	112	123	134	142	144	142	141	139	140	139	136	131	123
149	147	144	139	129	120	108	92	71	49	39	43	62	85	98	113	122	134	142	145	142	141	139	139	139	136	130	122
151	145	141	133	124	115	99	82	57	43	40	48	66	85	98	112	123	134	143	145	143	141	138	138	139	136	131	122
150	144	137	130	119	107	89	70	50	41	43	50	66	85	98	111	125	135	142	145	144	142	139	137	139	136	131	124
146	139	132	124	111	96	76	55	50	44	45	49	65	85	99	112	126	134	141	144	144	144	141	138	138	136	132	124

A bit more



All the way out

Starting to look like a face?

Kinda like this.

So an image is an array, which works like a spreadsheet. It's a series of numbers with a position in an array. The numbers for a standard 8 bit image, the numbers run between 0 and 255, for a 16 bit image the numbers run between 0 and 65535.

The point of the spreadsheet illustration was to demonstrate that we have data to work with, however the data is limited. We know that each pixel has a value in each Red, Green and Blue (even if the value is 0), We know that there is an X position and a Y position.

That's about it.

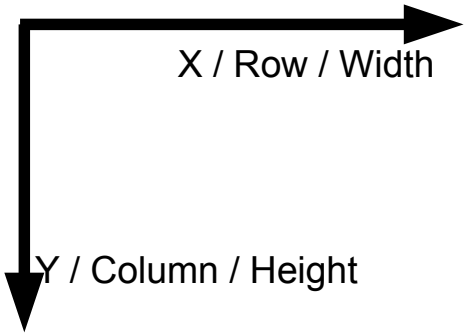
So we have a limited set of values to work with. However we can still do an awful lot with that information.

In software, we can find what the neighboring pixels are to a single pixel. We can find the mean, median, standard deviation of the image.

Kinda all sound like a lot of math. Well it is and that's the what image processing is, using math to alter your images.

Walking through an image

If you recall earlier, we talked about the Cartesian coordinates.



It was also mentioned that each color was a stack

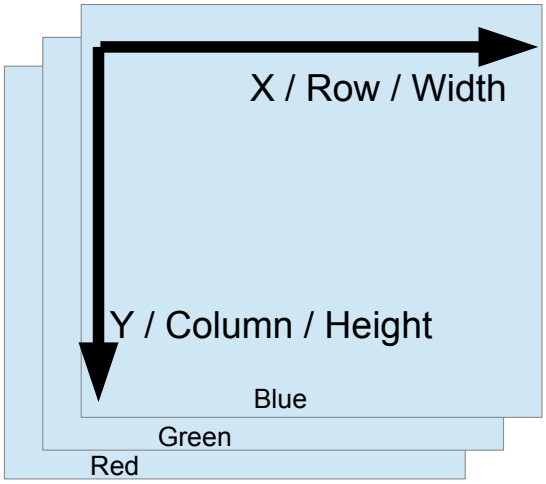
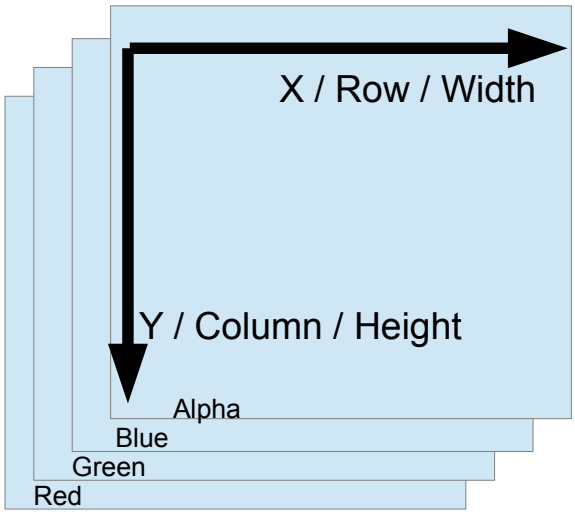


Photo-Reactor also has an Alpha layer, which serves as an Opacity layer for effect blending strength, so the complete picture looks like.



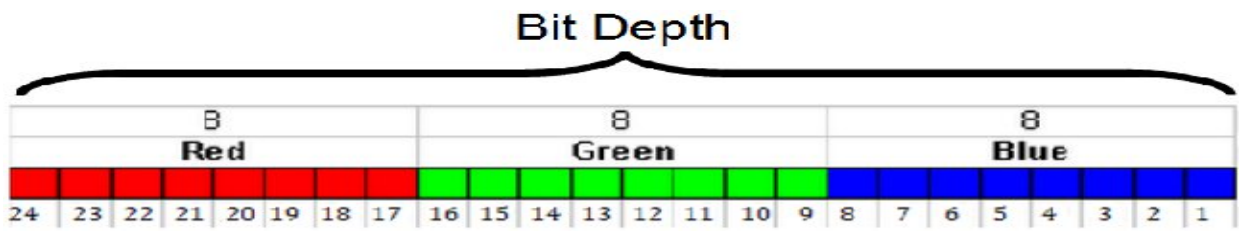
So in order to go through the image pixel by pixel, we need to create several loops, one loop to step through the Height of the image and one loop to step through the Width of the image.

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

We are also going to need the Bit Depth of the image so we can step through each pixel properly. This requires a little explanation.

Bit Depth / Color Space

Later in this document you will see references to Bit Depth or Color Space. An image can be refereed to an 8 bit image or 8 Bpp (8 Bits per pixel). The exact same image can be refereed to as a 24 bit image, both terms are correct.



If you decide to include Alpha (transparency) in the mix the image looks like

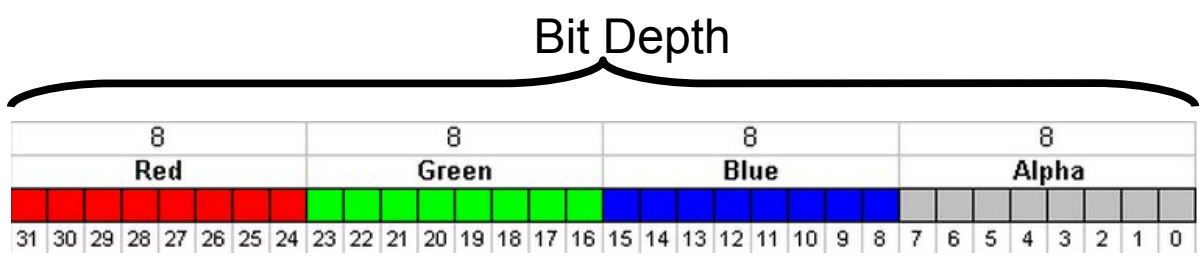


Photo-Reactor respects the Alpha Layer, so in the index calculation

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

The 4 represents each of the 4 layers, Red, Green, Blue and Alpha to skip to get to the next pixel.

In the SDK this is labeled nIdx and is displayed as

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

So the complete routine for stepping through the image would be

```
for (int y = 0; y< nHeight; y++)
{
        for (int x = 0; x< nWidth; x++)
        {
                int nIdx = x*4+y*4*nWidth;
        }
}
```

Acquiring the Compiler

Microsoft has since 2005 offered a free version of their C/C++ compiler, known as Visual Studio Express. There are several versions of Express and to complicate matters, there are a number of differences between each of the versions of Express. The versions available now are 2008, 2010 and 2012.

Express 2008 is being phased out by Microsoft, but there are still some avenues on acquiring it. 2008 works on Windows XP to Windows 7. Express 2010 replaced Express 2008 and it works on Windows Vista to Windows 7 in 32 bit or 64 Bit, but as of this writing did not work in 64 Bit mode on Windows 8. Express 2012 Desktop works on Windows 7 and Windows 8 and compiles in 32 and 64 Bit.

So your download version of Visual Studio Express depends on your operating system and preference.

For the purposes of this paper, we are using Express 2012 for Desktop. However your choice depends on your operating system and your preference. On versions prior to Express 2012, you will have to download the C/C++ specific compiler.

Download Visual Studio Express and install. You will most likely have to register your program, this is free of charge from Microsoft.

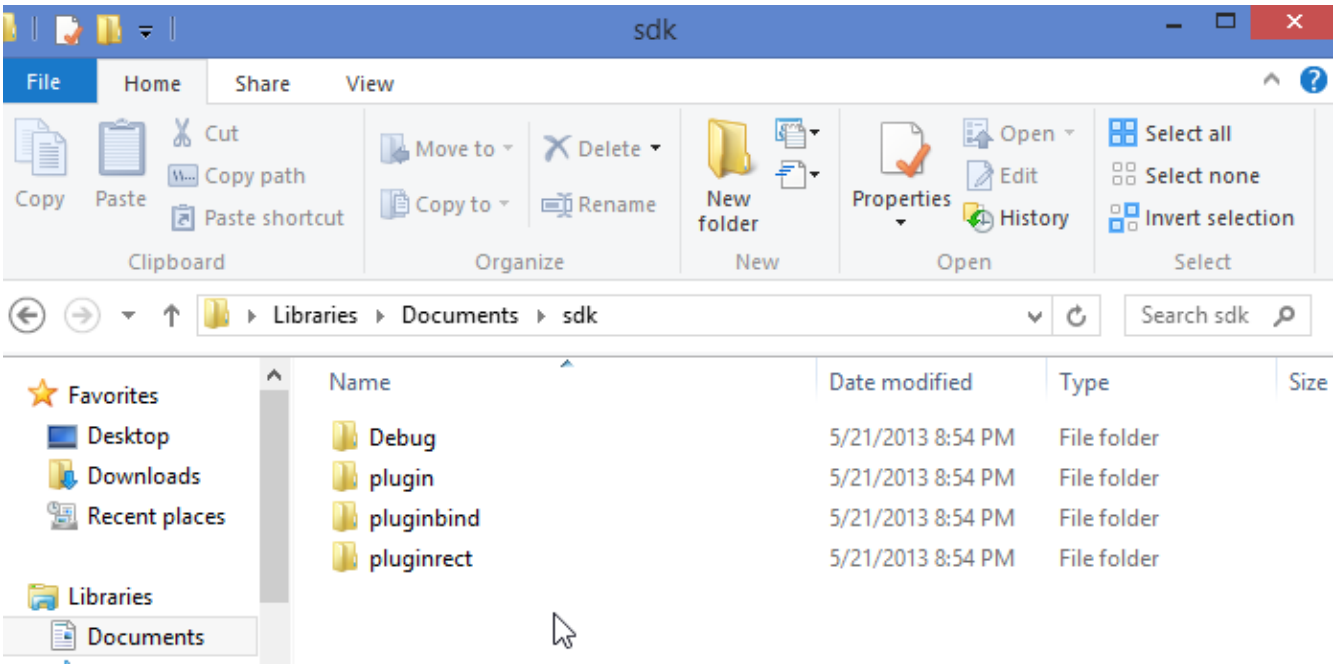
That is all for this step. We have decided which compiler version to use, we have installed and registered the compiler

At the time of this writing Visual Studio express is found at
http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-products

Acquiring the Photo-Reactor SDK

Getting the SDK.

The SDK and sample code are available from Media Chance's website, coming as a ZIP File.



The SDK Folder

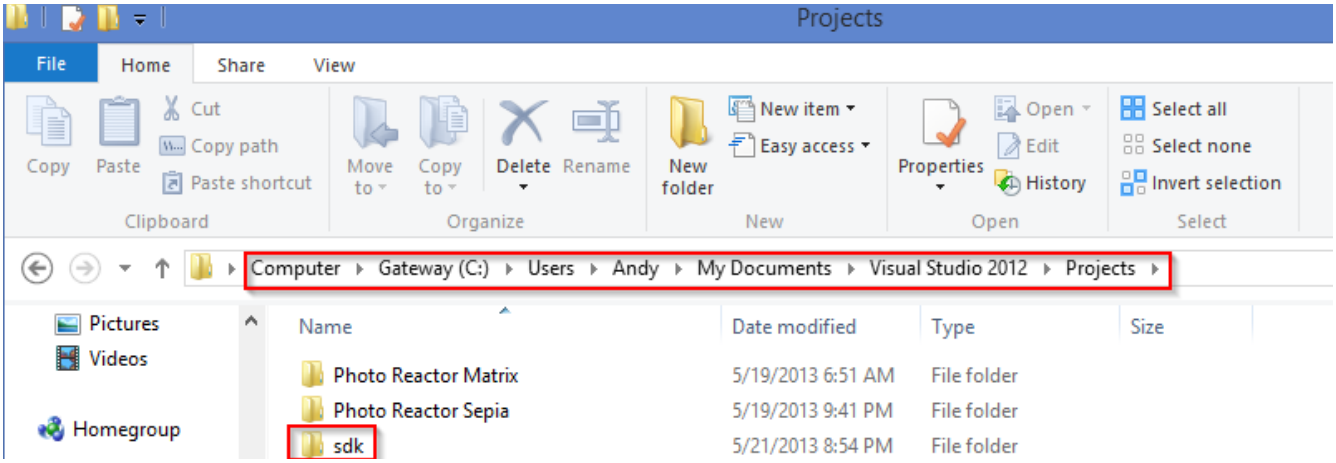
Moving the SDK to the proper folder

Uncompressed the SDK files to the projects directory of visual studio.

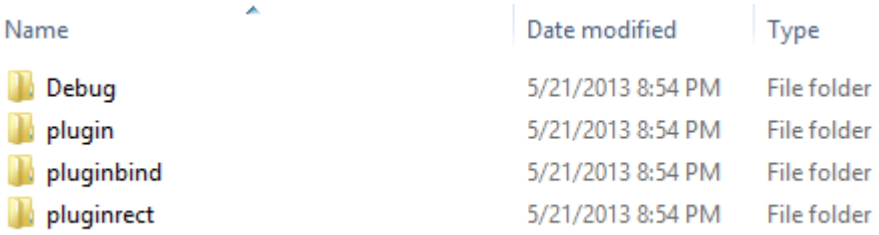
The Directory should be found under C:\Users\user_name\Documents\Visual Studio xxxx\Projects

For example my directory is

C:\Users\Andy\Documents\Visual Studio 2012\Projects



Now let's explore the SDK in a little more detail.

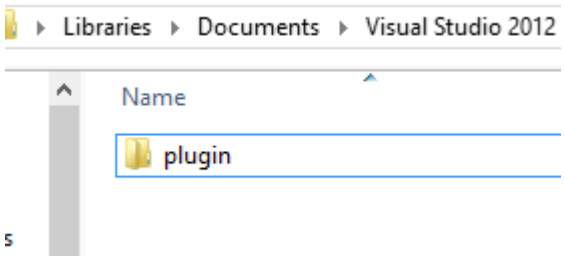


We see that there are 4 directories – Debug, plugin, pluginbind, pluginrect.

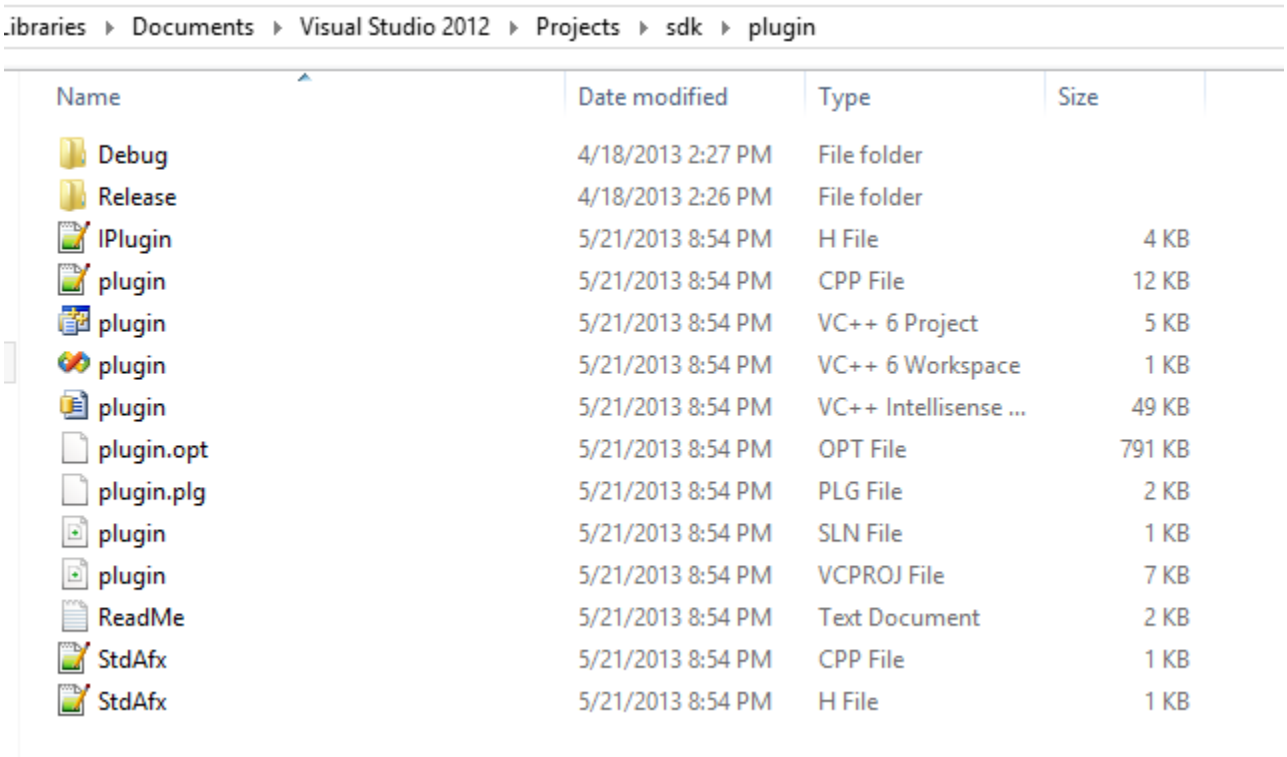
The description of each are found on the Photo-Reactor website.

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)

For this paper, we are only writing a plug-in, the other 3 directories are not needed for this example, so you can either delete them or move them elsewhere, so that we can simplify this example.



Now lets open the plugin directory.



Name	Date modified	Type	Size
Debug	4/18/2013 2:27 PM	File folder	
Release	4/18/2013 2:26 PM	File folder	
IPlugin	5/21/2013 8:54 PM	H File	4 KB
plugin	5/21/2013 8:54 PM	CPP File	12 KB
plugin	5/21/2013 8:54 PM	VC++ 6 Project	5 KB
plugin	5/21/2013 8:54 PM	VC++ 6 Workspace	1 KB
plugin	5/21/2013 8:54 PM	VC++ Intellisense ...	49 KB
plugin.opt	5/21/2013 8:54 PM	OPT File	791 KB
plugin.plg	5/21/2013 8:54 PM	PLG File	2 KB
plugin	5/21/2013 8:54 PM	SLN File	1 KB
plugin	5/21/2013 8:54 PM	VCPROJ File	7 KB
ReadMe	5/21/2013 8:54 PM	Text Document	2 KB
StdAfx	5/21/2013 8:54 PM	CPP File	1 KB
StdAfx	5/21/2013 8:54 PM	H File	1 KB

Here are all the files and 2 directories, debug and release. The Debug and Release is where the final code is built and stored, so they will become important later on. However to again simplify things, lets select all the files and move them to the prior directory.

Libraries ▸ Documents ▸ Visual Studio 2012 ▸ Projects ▸ sdk

Name	Date modified	Type	Size
Debug	4/18/2013 2:27 PM	File folder	
plugin	5/22/2013 6:01 AM	File folder	
Release	4/18/2013 2:26 PM	File folder	
IPlugin	5/21/2013 8:54 PM	H File	4 KB
plugin	5/21/2013 8:54 PM	CPP File	12 KB
plugin	5/21/2013 8:54 PM	VC++ 6 Project	5 KB
plugin	5/21/2013 8:54 PM	VC++ 6 Workspace	1 KB
plugin	5/21/2013 8:54 PM	VC++ Intellisense ...	49 KB
plugin.opt	5/21/2013 8:54 PM	OPT File	791 KB
plugin.plg	5/21/2013 8:54 PM	PLG File	2 KB
plugin	5/21/2013 8:54 PM	SLN File	1 KB
plugin	5/21/2013 8:54 PM	VCPROJ File	7 KB
ReadMe	5/21/2013 8:54 PM	Text Document	2 KB
StdAfx	5/21/2013 8:54 PM	CPP File	1 KB
StdAfx	5/21/2013 8:54 PM	H File	1 KB

Notice the directory location, we have copied all the files to the prior directory. We don't need the plugin directory anymore so we can delete it.

The 2 files that we are interested in at this point are the plugin (VC++ Project) and plugin (CPP).

Debug	4/18/2013 2:27 PM	File folder	
Release	4/18/2013 2:26 PM	File folder	
IPlugin	5/21/2013 8:54 PM	H File	4 KB
plugin	5/21/2013 8:54 PM	CPP File	12 KB
plugin	5/21/2013 8:54 PM	VC++ 6 Project	5 KB
plugin	5/21/2013 8:54 PM	VC++ 6 Workspace	1 KB
plugin	5/21/2013 8:54 PM	VC++ Intellisense ...	49 KB
plugin.opt	5/21/2013 8:54 PM	OPT File	791 KB
plugin.plg	5/21/2013 8:54 PM	PLG File	2 KB
plugin	5/21/2013 8:54 PM	SLN File	1 KB
plugin	5/21/2013 8:54 PM	VCPROJ File	7 KB
ReadMe	5/21/2013 8:54 PM	Text Document	2 KB
StdAfx	5/21/2013 8:54 PM	CPP File	1 KB
StdAfx	5/21/2013 8:54 PM	H File	1 KB

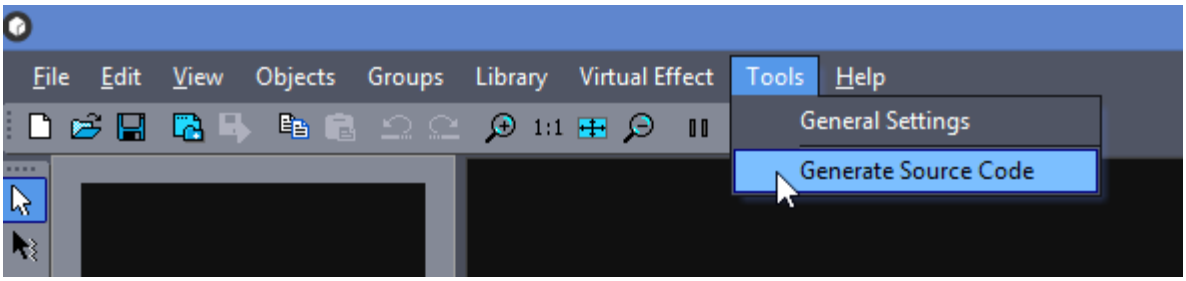
The VC++ 6 Project is how we open the file in Visual Studio and the CPP file is the actual code.

However, do not open either at this point, because now we need to create our plug-in basics with Photo-Reactor.

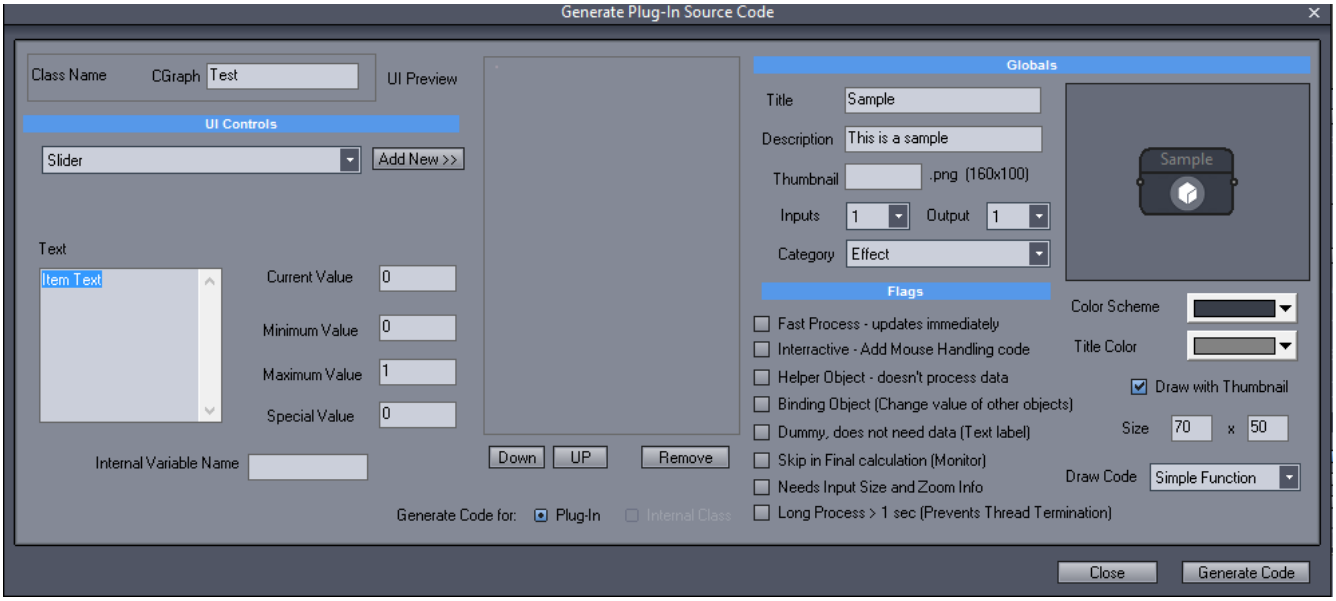
Creating the plug-in Basics

Lets start by opening Photo-Reactor.

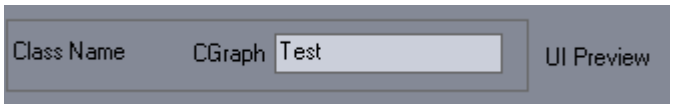
Drill to Tools – Generate Source Code.



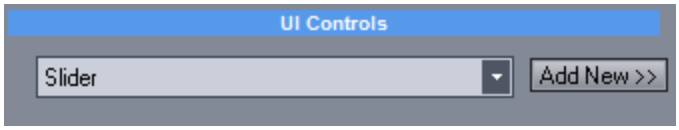
This will open a new page, and there are a good number of option to be found



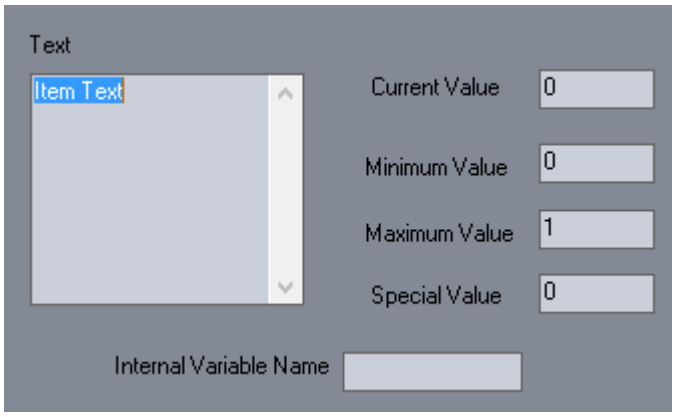
Dissection of the plug-in creator



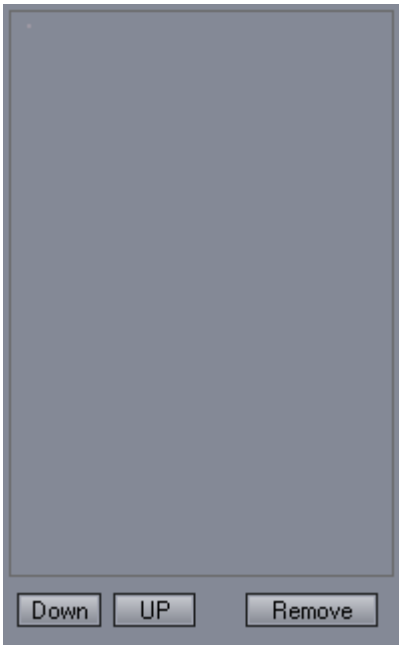
This names your Class and Constructor



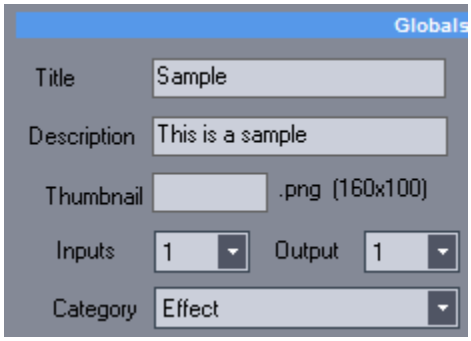
This is where you can add various controls to your plug-in, the control types are Slider, check box, Label Text, radio buttons, color selector, image file, Check box + enable all following, label edit box, combo box, Font Combo, multi line text, exponential slider, push button, gamma slider, logarithmic slider, position control, integer input, multi-line edit box, check box, + enable all / until, horizontal space.



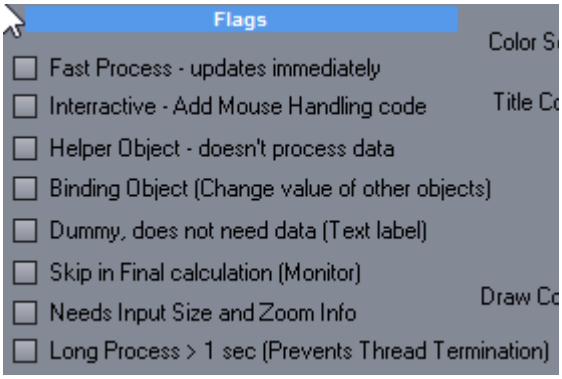
This is where you can name each of the various controls. For each of the controls you can set the minimum and maximum values as well as the initial value.

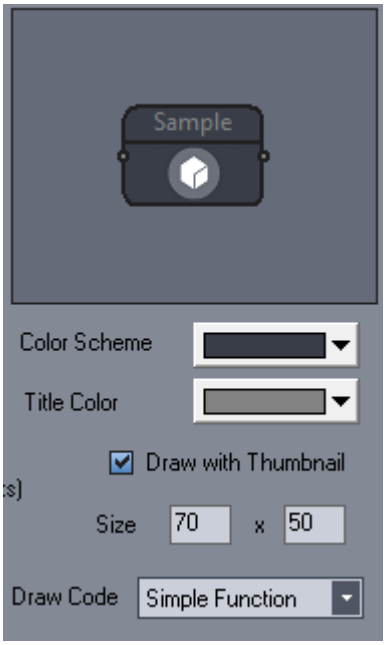


When the controls are added, they are placed in this box, you can move the controls up and down or remove the control.



This is the name that shows in Photo-Reactor. The title should remain short. The description should be longer to display what your plug-in will do.





This is how the icon will appear within Photo-Reactor.

Programming your first filter

In life we learn to crawl, before we start to walk. In programming, and image processing, it's the same.

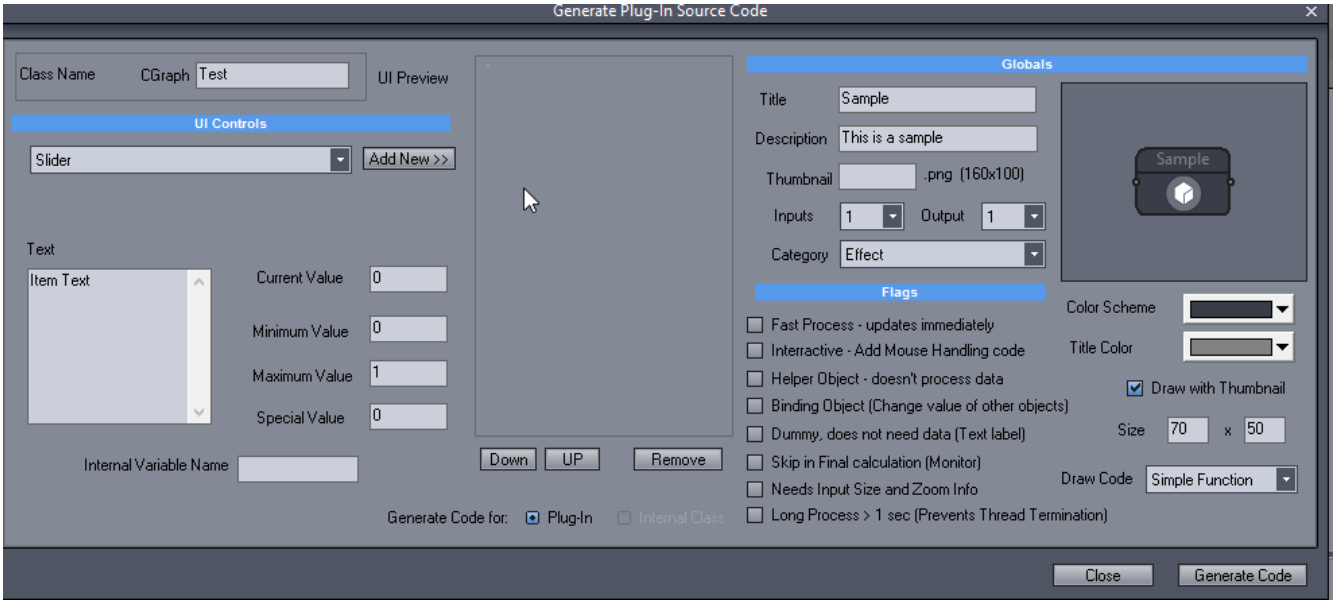
First, we are going to program an easy filter, invert.

We simply take each color Red, Green and Blue and invert it.

Not an impressive thing to do, however it will get the thinking process started.

Let's Open Photo-Reactor.

Go to Tools – Generate Source code.



We are now going to populate some items.

This is going to be a real simple plug-in, so there will be no controls involved.

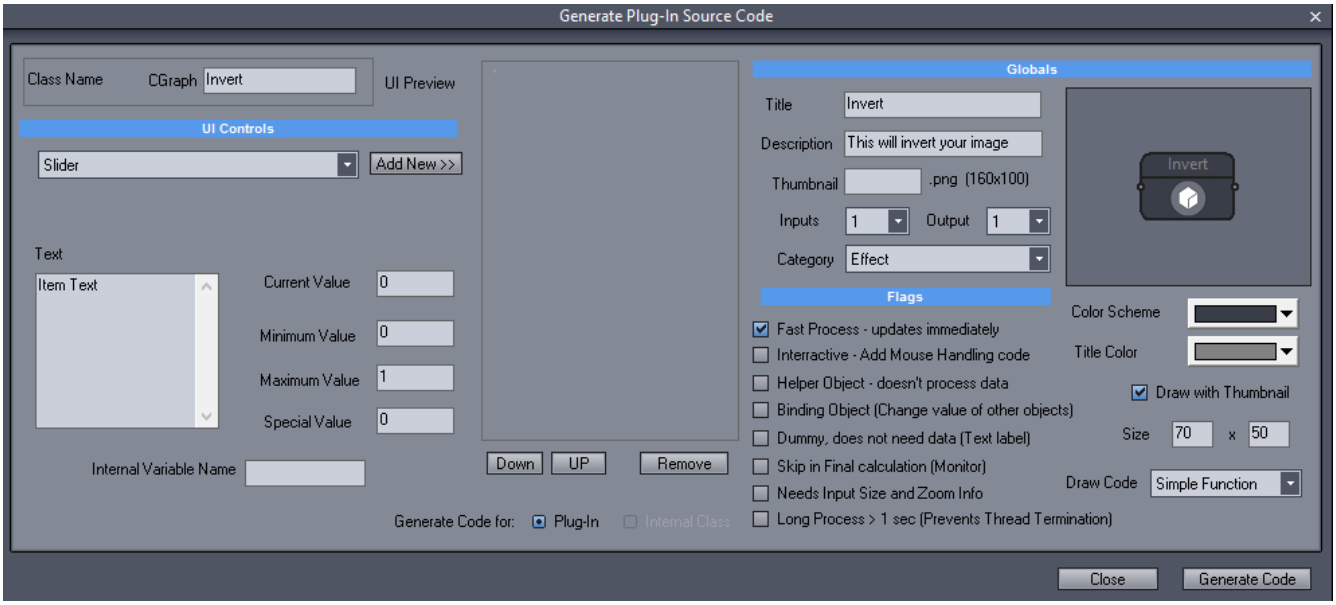
Name your class “Invert.”

Go to the Globals section

Title will be “Invert”
Description “This will invert your image”

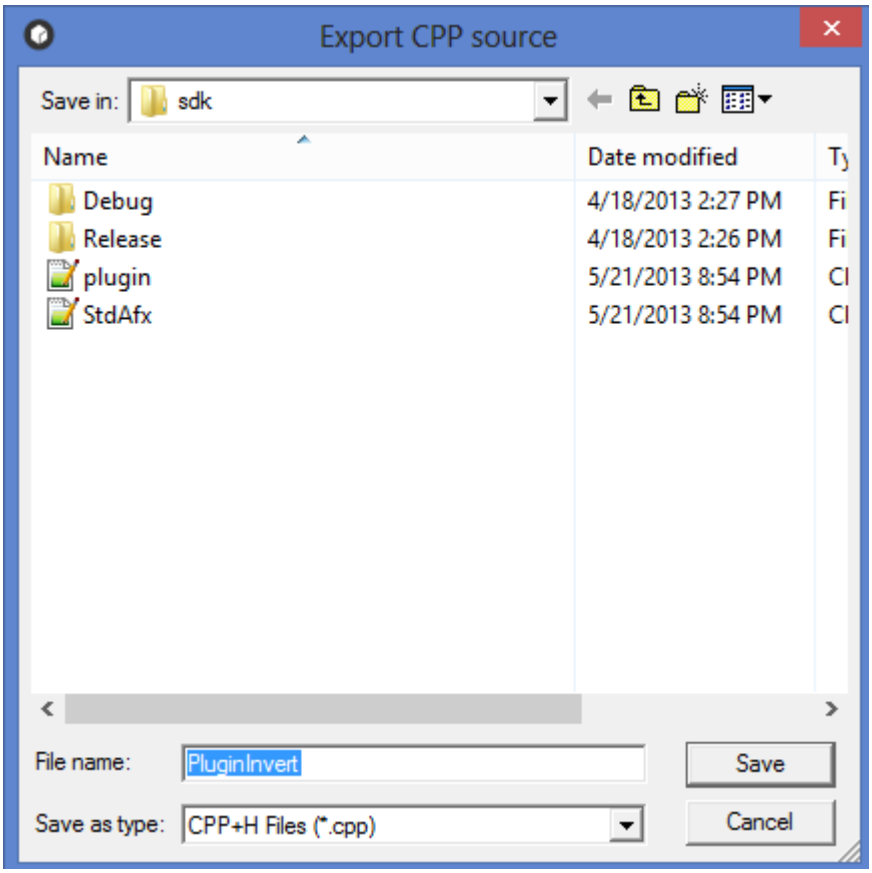
Check the fast process flag.

It should look like this.



Let's select the “Generate Code Button.”

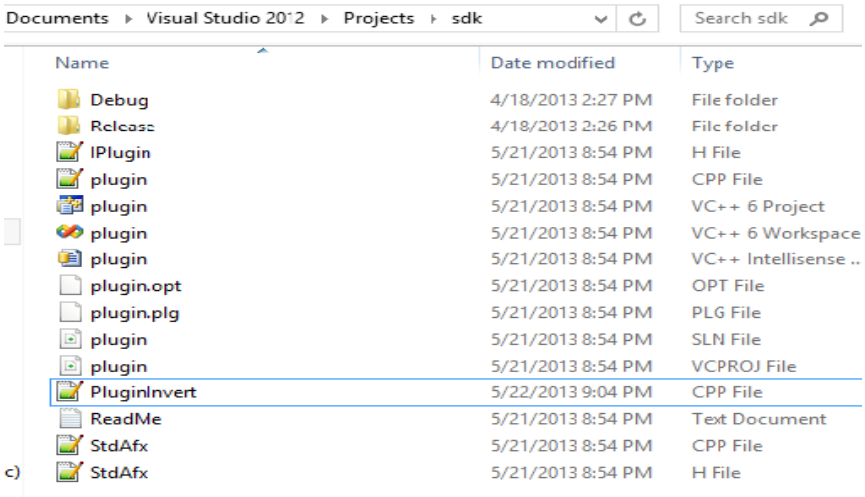
You will be presented with a standard windows save screen. Navigate to your SDK folder, created earlier, within the Visual Studio directory and save your file.



After you save, exit Photo-Reactor, we won't need it for a while.

If you followed the SDK steps earlier, you have created in your compiler directory a folder called “sdk”

Let's go to that directory via Windows Explorer.



Let's keep this open for now.

Now lets find that plug-in we just created. We see it just as we saved it, PluginInvert.

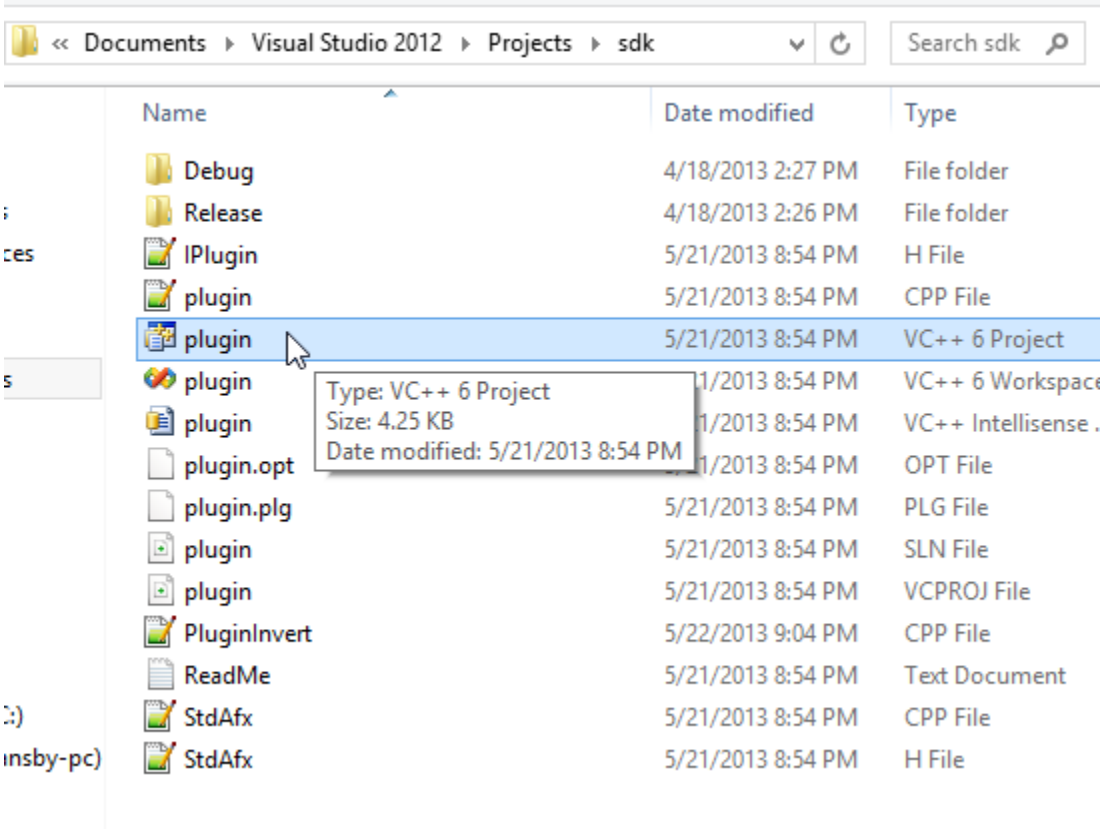
Let's open that with Windows Notepad (or in my case Notepad ++)

PluginInvert.cpp

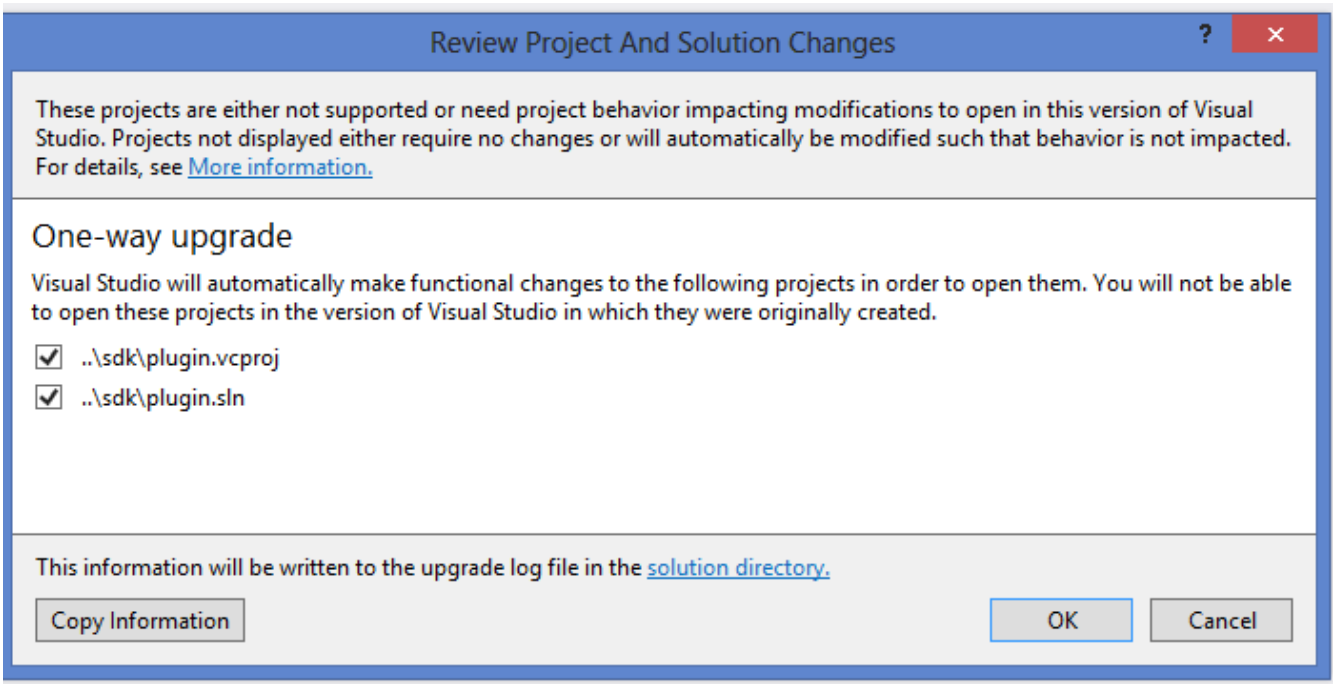
```
// plugin.cpp : Defines the entry point for the DLL application.
//

#include "stdafx.h"
#include "IPlugin.h"

////////////////////////////////////////////////////////////////////////
// A concrete plugin implementation
////////////////////////////////////////////////////////////////////////

// Photo-Reactor Plugin class

//*****************************************************************************
//This code has been generated by the Mediachance photo reactor Code generator.


#define AddParameter(N,S,V,M1,M2,T,D) {strcpy (pParameters[N].m_sLabel,S);pParam
M2;pParameters[N].m_nType = T;pParameters[N].m_dSpecialValue = D;}

#define GetValue(N) (pParameters[N].m_dValue)
#define GetValueY(N) (pParameters[N].m_dSpecialValue)

#define SetValue(N,V) {pParameters[N].m_dValue = V;}

#define GetBOOLValue(N) ((BOOL)(pParameters[N].m_dValue==pParameters[N].m_dMax))

```

There's our code. For now do nothing with it, just let it sit there for now.

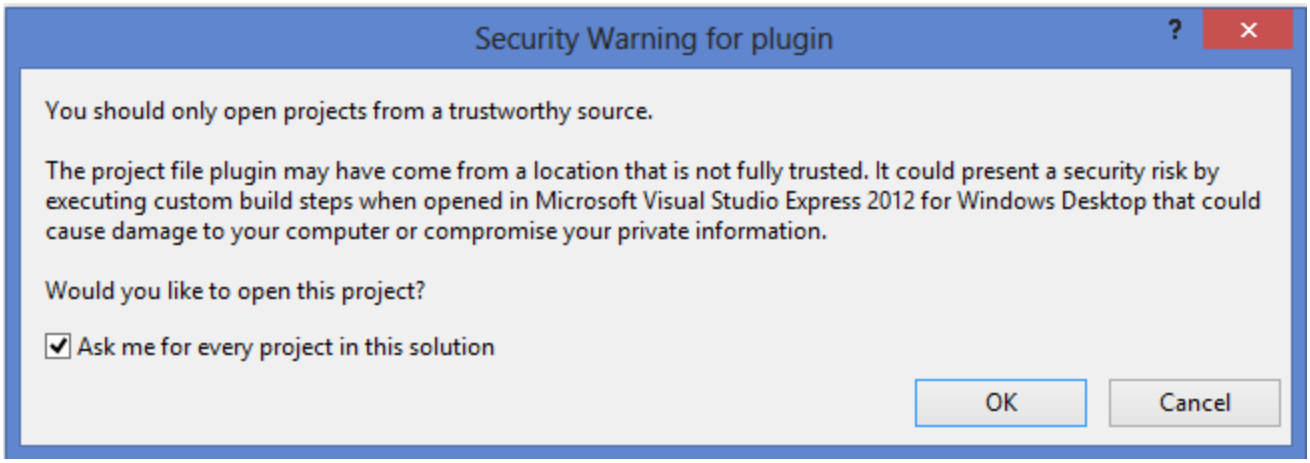
Back to Windows file explorer and our SDK folder.



Let's open our project, it is listed as a VC++ 6 Project.



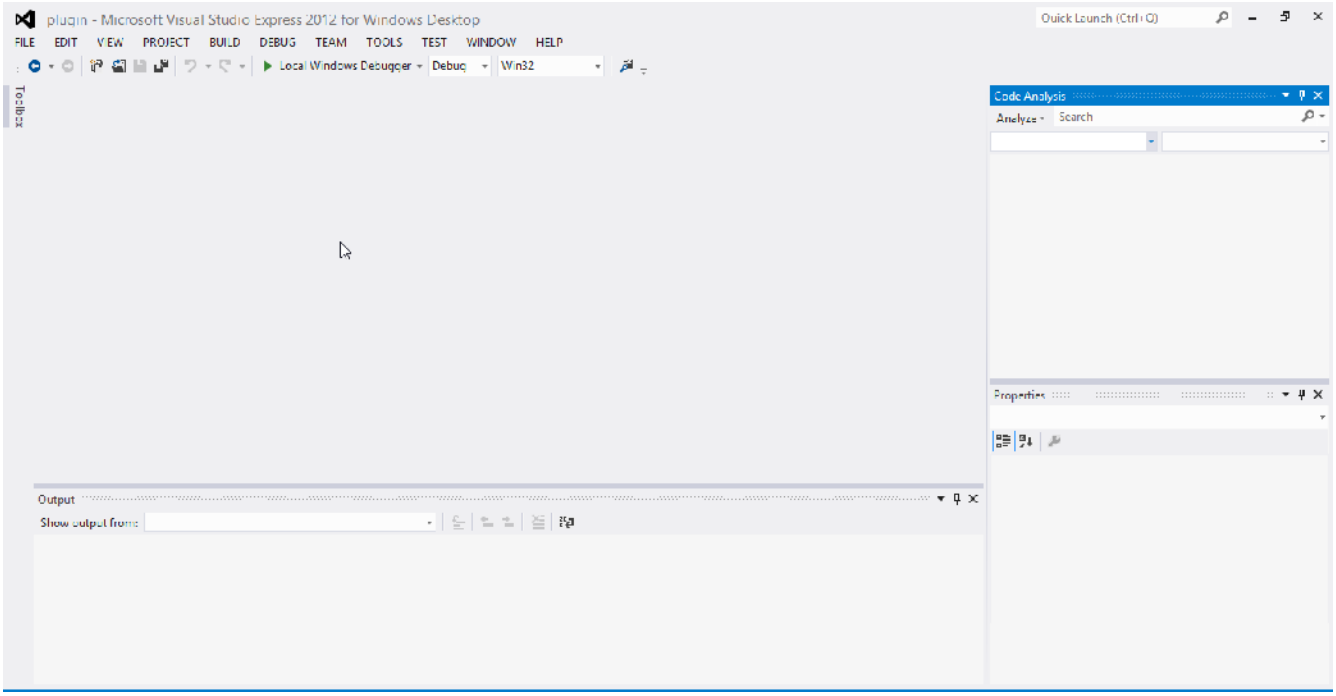
Visual Studio will ask you to convert your project. This is because the plug-in SDK was written with an earlier version of Visual Studio, nothing wrong with that. Let's convert it. Select OK.



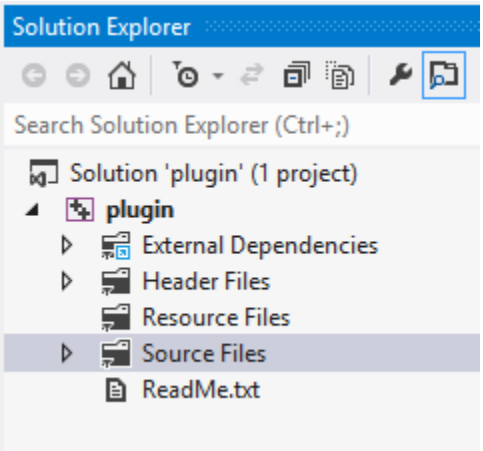
You will get a warning from Microsoft. Only open from a trustworthy source, good advise, but we trust this source, select OK.

Well at this point, Visual Studio will work its magic and convert it, however you may get a warning screen from Microsoft. Not to worry, This is not a web application and we were sternly warned by Microsoft. Nothing to worry about, everything will convert just fine.

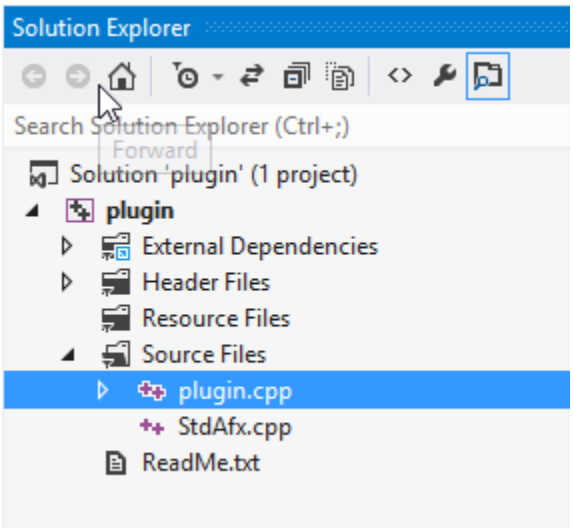
So now we are in, now what?



Drill to View – Solution Explorer



Select that arrow beside Source Files



Select plugin.cpp

```
(Global Scope)
// plugin.cpp : Defines the entry point for the DLL application.
//

#include "stdafx.h"
#include "IPlugin.h"

////////////////////////////////////////////////////////////////////////////
// A concrete plugin implementation
////////////////////////////////////////////////////////////////////////////

// Plugin class
// simple effect plugin


#define AddParameter(N,S,V,M1,M2,T,D) {strcpy (pParameters[N].m_sLabel,S);pParameters[N].m_d

#define GetValue(N) (pParameters[N].m_dValue)
#define GetValueY(N) (pParameters[N].m_dSpecialValue)

#define SetValue(N,V) {pParameters[N].m_dValue = V;}

#define GetBOOLValue(N) ((BOOL)(pParameters[N].m_dValue==pParameters[N].m_dMax))
```

There is some code there. However it's not the code we generated with Photo-Reactor. That is in your notepad that we left open.

Let's open notepad again.

Select anywhere in Notepad and
Press CTRL-A (notice everything is highlighted)
Press CTRL-C (we copied)

Open Visual studio again.

Select anywhere in Visual Studio
Press CTRL-A (notice everything is highlighted)
Press CTRL-V (we pasted)

We have just pasted the code from the Photo-Reactor generated code to Visual Studio. However we still have a ways to go. However let's explore this a bit more in detail.

A Breakdown of the Generated code

```
// plugin.cpp : Defines the entry point for the DLL application.
//

#include "stdafx.h"
#include "IPlugin.h"
```

This defines your includes, this will add addition library modules to be added to your program. Several common includes may be math.h or stdout.h right now we are not going to to this at all.

```
////////////////////////////////////////////////////////////////////
// A concrete plugin implementation
////////////////////////////////////////////////////////////////////

// Photo-Reactor Plugin class

//*****************************************************************************
//This code has been generated by the Mediachance photo reactor Code generator.

#define AddParameter(N,S,V,M1,M2,T,D) {strcpy
(pParameters[N].m_sLabel,S);pParameters[N].m_dValue = V;pParameters[N].m_dMin =
M1;pParameters[N].m_dMax = M2;pParameters[N].m_nType = T;pParameters[N].m_dSpecialValue = D;}

#define GetValue(N) (pParameters[N].m_dValue)
#define GetValueY(N) (pParameters[N].m_dSpecialValue)

#define SetValue(N,V) {pParameters[N].m_dValue = V;}

#define GetBOOLValue(N) ((BOOL)(pParameters[N].m_dValue==pParameters[N].m_dMax))

// if it is not defined, then here it is
//#define RGB(r,g,b) ((COLORREF)(((BYTE)(r)|((WORD)((BYTE)(g))<<8))|(((DWORD)(BYTE)(b))<<16)))



#define NUMBER_OF_USER_PARAMS 0
```

We don't have any controls, so we have no User Params.

```
class PluginInvert : public IPlugin
{
public:

// constructor
PluginInvert()
{
}


//Plugin Icon:
//you can add your own icon by creating 160x100 png file, naming it the same as plugin dll and
then placing it in the plugins folder
//otherwise a generic icon will be used
```

If we want our plug-in to have a custom icon, we can do that by following the above instructions.

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

This is the Short title we created in the Globals section

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

This is the description we created in the Globals section.

```
// BASIC PARAMETERS
// number of inputs 0,1 or 2
int GetInputNumber ()
{
	return 1;
}
```

This is the number of inputs as selected in the Globals section.

```
// number of outputs 0 or 1
int GetOutputNumber ()
{
	return 1;
}
```

This is the number of inputs as selected in the Globals section.

```
int GetBoxColor ()
{
	return RGB(56,61,72);
}
```

This was found in the color scheme

```
int GetTextColor ()
{
	return RGB(130,130,130);
}
```

This was found in Title color

```
// width of the box in the workspace
// valid are between 50 and 100
int GetBoxWidth ()
{
	return 70;
}
```

This was found in size of box

```
// set the flags
// 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
```

Remember all those Flags, here is instructions on their usage.

```
int GetFlags ()
{
	// it is fast process
	int nFlag = FLAG_NONE;

	nFlag = nFlag | FLAG_UPDATE_IMMEDIATELY;

	return nFlag;
}
```

And here is how that flag is set, not to worry the source code generator already did the job.

```
// User Interface Build
// there is maximum 29 Parameters

int GetUIParameters (UIParameters* pParameters)
{
	// label, value, min, max, type_of_control, special_value
	// use the UI builder in the software to generate this

	return NUMBER_OF_USER_PARAMS;
}
```

In the UI controls portion of the source code generator, we could set the parameter, minimum, maximum and default values. This is placed here and generated automatically.

```
// Actual processing function for 1 input
//*****************************************************************************************
// Both buffers are the same size
// don't change the IN buffer or things will go bad for other objects in random fashion
// the pBGRA_out comes already with pre-copied data from pBGRA_in
// Note: Don't assume the nWidth and nHeight will be every run the same or that it contains the
whole image!!!!
// This function receives buffer of the actual preview (it can be just a crop of image when
zoomed in) and during the final calculation of the full buffer
```

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight,
UIParameters* pParameters)
{
	// this is just example to desaturate and to adjust the desaturation with slider
	// Get the latest parameters
	//List of Parameters
	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 = CLAMP255((nR+nG+nB)/3);

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

This is where our magic happens. This is the actual processing routine. This is the most important part of the filter, so let's break it down.

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

virtual void Process_Data is how the routine is called from Photo-Reactor.

It is using **pBGRA_in** as an input. pBGRA stands for picture Blue, Green, Red, Alpha.

pBGRA_out is the output.

It is using **nWidth** (X) as the width of the section, not necessary the width of the image.

It is using **nHeight** (Y) as the height of the section, not necessary the height of the image.

UIParameters* pParameters deals with the user controls.

for (int y = 0; y< nHeight; y++)
This line scrolls through every pixel for the height of the selection, it works with the next statement

for (int x = 0; x< nWidth; x++)
This line scrolls through every pixel for the width of the selection.

Just FYI y++ and x++ are incrementors which tell the software to go to the next pixel.

That is for the outside routine, lets move to the inside routine.

```
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 = CLAMP255((nR+nG+nB)/3);

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

nIdx is very important, this is how your plug-in will keep track of where it is in the image. This is a special function called a pointer. Without it the software would not know where to go to modify or read a pixel.

The nR, nG, nB variables that read the pixel for the Red, Green and Blue channels. The data that these variables read are arrays (remember earlier when I said that images are arrays like a spreadsheet?). The array is pBGRA with the channel being the color (remember the layer / stack analogy?). This all works in conjunction with the nIdx variable.

In the SDK, the generated plug-in already does a sample image processing routine, it desaturates an image, or make the image black and white. This is what the next line does.

The nA variable does just that, makes the image black and white.

It does it by taking the red, green and blue channels, adds them all together and divides by 3. This will produce a greyscale image with all colors being an average or equal weight.

This line also performs a checksum to make sure that the pixels do not go over 255 (remember an 8 bit image only has values from 0 to 255 and no more).

The routine concludes with the lines

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

These lines write our data out to the image.

```
// actual processing function for 2 inputs
//*****************************************************************************
// all buffers are the same size
// don't change the IN buffers or things will go bad
// the pBGRA_out comes already with copied data from pBGRA_in1
virtual void Process_Data2 (BYTE* pBGRA_out, BYTE* pBGRA_in1, BYTE* pBGRA_in2, int nWidth, int
nHeight, UIParameters* pParameters)
	{
	}
This is you are using 2 inputs, more advanced.

//*****************Drawing functions for the BOX *******************************
//how is the drawing handled
//DRAW_AUTOMATICALLY the main program will fully take care of this and draw a box, title, socket
and thumbnail
//DRAW_SIMPLE_A                  will draw a box, title and sockets and call CustomDraw
//DRAW_SIMPLE_B                  will draw a box and sockets and call CustomDraw
//DRAW_SOCKETSONLY      will call CustomDraw and then draw sockets on top of it

// highlighting rectangle around is always drawn except for DRAW_SOCKETSONLY

virtual int GetDrawingType ()
{
	int nType = DRAW_AUTOMATICALLY;
	return nType;
}


// Custom Drawing
// custom drawing function called when drawing type is different than DRAW_AUTOMATICALLY
// it is not always in real pixels but scaled depending on where it is drawn
// the scale could be from 1.0 to > 1.0
// so you always multiply the position, sizes, font size, line width with the scale

virtual void CustomDraw (HDC hDC, int nX,int nY, int nWidth, int nHeight, float scale, BOOL
bIsHighlighted, UIParameters* pParameters)
{
}


//************ Optional Functions
*******************************************************************************
// those functions are not necessary for normal effect, they are mostly for special effects and
objects

// Called when FLAG_HELPER set.
// When UI data changed (user turned knob) this function will be called as soon as user finish
channging the data
// You will get the latest parameters and also which parameter changed
// Normally for effects you don't have to do anything here because you will get the same
parameters in the process function
// It is only for helper objects that may not go to Process Data
BOOL UIParametersChanged (UIParameters* pParameters, int nParameter)
{
	return FALSE;
}



// when button is pressed on UI, this function will be called with the parameter and sub button
```

```
(for multi button line)
BOOL UIButtonPushed (int nParam, int nSubButton, UIParameters* pParameters)
{
	return TRUE;
}

// Called when FLAG_NEEDSIZEDATA set
// Called before each calculation (Process_Data)
// If your process depends on a position on a frame you may need the data to correctly display
it because Process_Data receives only a preview crop
// Most normal effects don't depend on the position in frame so you don't need the data
// Example: drawing a circle at a certain position requires to know what is displayed in
preview or the circle will be at the same size and position regardless of zoom

// Note: Even if you need position but you don't want to mess with the crop data, just ignore
it and pretend the Process_Data are always of full image (they are not).
// In worst case this affects only preview when using zoom. The full process image always sends
the whole data

// nOriginalW, nOriginalH - the size of the original - full image. If user sets Resize on input
- this will be the resized image
// nPreviewW, nPreviewH   - this is the currently processed preview width/height - it is the
same that Process_Data will receive
//                        - in full process the nPreviewW, nPreviewH is equal nOriginalW,
nOriginalH
// Crop X1,Y1,X2,Y2       - relative coordinates of preview crop rectangle in <0...1>, for full
process they are 0,0,1,1 (full rectangle)
// dZoom                  - Zoom of the Preview, for full process the dZoom = 1.0

void SetSizeData(int nOriginalW, int nOriginalH, int nPreviewW, int nPreviewH, double dCropX1,
double dCropY1, double dCropX2, double dCropY2, double dZoom)
{
	// so if you need the position and zoom, this is the place to get it.
	// Note: because of IBM wisdom the internal bitmaps are on PC always upside down, but
the coordinates are not
	// which you  need to take into account. See rectangle demo project for more info
}


// ***** Mouse handling on workplace *************************
// only if FLAG_NEEDMOUSE is set
//*************************************************************
//this is for special objects that need to receive mouse, like a knob or slider on workplace
// normally you use this for FLAG_BINDING objects

// in coordinates relative to top, left corner of the object (0,0)
virtual BOOL MouseButtonDown (int nX, int nY, int nWidth, int nHeight, UIParameters*
pParameters)
{
	// return FALSE if not handled
	// return TRUE if handled
	return FALSE;
}
```

```
// in coordinates relative to top, left corner of the object (0,0)
virtual BOOL MouseMove (int nX, int nY, int nWidth, int nHeight, UIParameters* pParameters)
{
	return FALSE;
}

// in coordinates relative to top, left corner of the object (0,0)
virtual BOOL MouseButtonUp (int nX, int nY, int nWidth, int nHeight, UIParameters* pParameters)
{
	// Note: if we changed data and need to recalculate the flow we need to return TRUE

	// return FALSE if not handled
	// return TRUE if handled

	return TRUE;
}
};

extern "C"
{
	// Plugin factory function
	__declspec(dllexport) IPlugin* Create_Plugin ()
	{
		//allocate a new object and return it
		return new PluginInvert ();
	}

	// Plugin cleanup function
	__declspec(dllexport) void Release_Plugin (IPlugin* p_plugin)
	{
		//we allocated in the factory with new, delete the passed object
		delete p_plugin;
	}

}
```

This deals with the creation and destruction of the plug-in, don't mess with this.

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

Here is the name of the plug-in we created in Globals.

```
// This MUST be unique string for each plugin so we can save the data

extern "C" __declspec(dllexport) char* GetPluginID()
{

//      IMPORTANT:you have to fill unique ID for every plugin:
//      The ID must be unique or loading and saving will not be able to find correct plugin
//  Comment out this line below so you can compile
        ATTENTION

        return "com.yourdomain.testplugin";
}
```

Your plugin will NOT compile unless you take care of this line. This is so your plug-in can have a unique ID all the copies of Photo-Reactor in the world, not just within your copy Photo-Reactor. Just give the copy a unique name such as com.lumafilters.myawesomeplugin, each plug-in you create needs to have its own unique name.

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

Getting to work on Invert

Well that is enough of looking through the source code and it's time to get to business.

Here is the formula for inverting an image

Inverted Color = Image Depth – Image Color

In the construct of Photo-Reactor.
nR = 255 - nR;
nG = 255 - nG;
nB = 255 - nB;

The variables nR, nG and nB have the color depth 255 subtracted from the color itself (nR, nG, nB).

Remember the section where the “magic happens”, it looked like this

```
	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 = CLAMP255((nR+nG+nB)/3);

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

Replace with the following

```
	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];

			nR = 255 - nR;
			nG = 255 - nG;
			nB = 255 - nB;

			pBGRA_out[nIdx+CHANNEL_R] = nR;
			pBGRA_out[nIdx+CHANNEL_G] = nG;
			pBGRA_out[nIdx+CHANNEL_B] = nB;
		}
	}
```

Now go down to the bottom and find the line

```
extern "C" __declspec(dllexport) char* GetPluginID()
{

//     IMPORTANT:you have to fill unique ID for every plugin:
//     The ID must be unique or loading and saving will not be able to find correct plugin
//  Comment out this line below so you can compile
       ATTENTION

       return "com.yourdomain.testplugin";

}
```

Comment out the Attention line by adding two slashes //

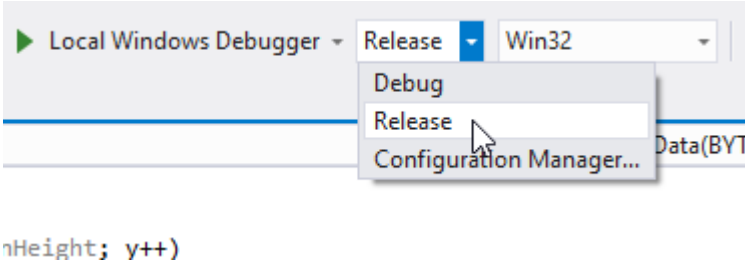
Rename the return function with your own identification such as

com.lumafilters.invert

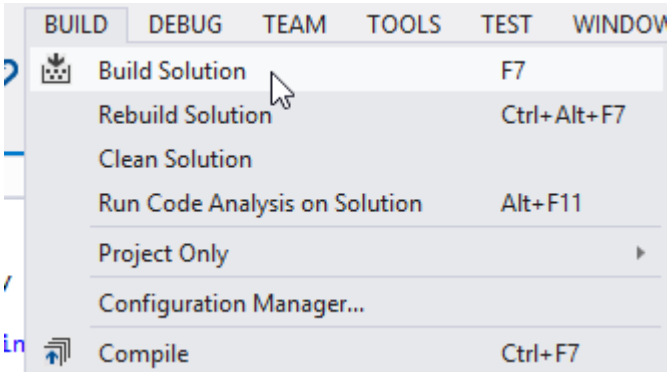
and the whole routine should look like

```
extern "C" __declspec(dllexport) char* GetPluginID()
{

//     IMPORTANT:you have to fill unique ID for every plugin:
//     The ID must be unique or loading and saving will not be able to find correct plugin
//  Comment out this line below so you can compile
//     ATTENTION

       return "com.lumafilters.invert";

}
```

At the top of your compiler, pull down and select release.

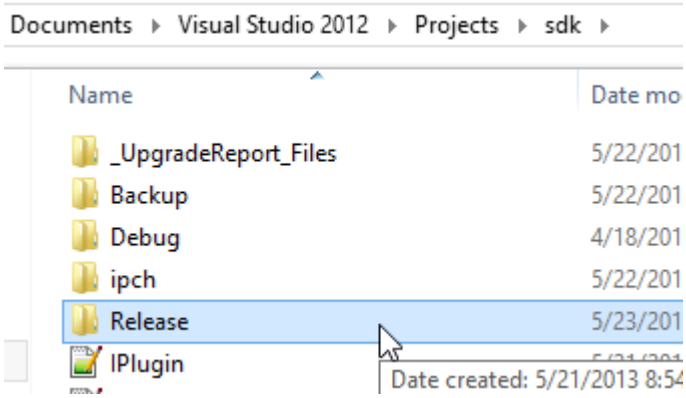


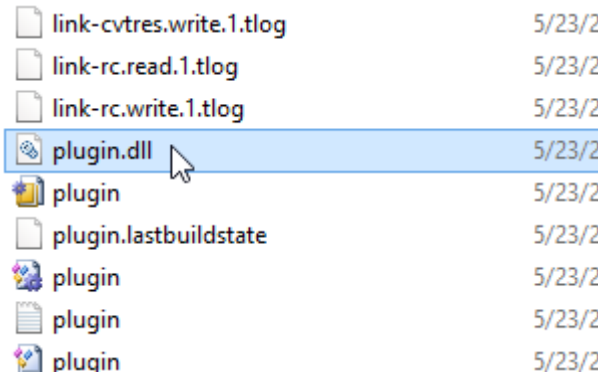
Select Build – Build Solution



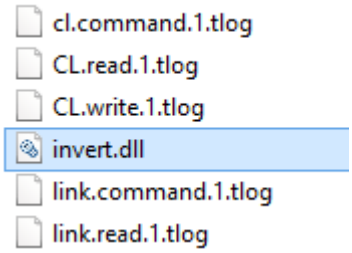
In a few short moments, the compiler will complete.

Open File Explorer and open the Release folder



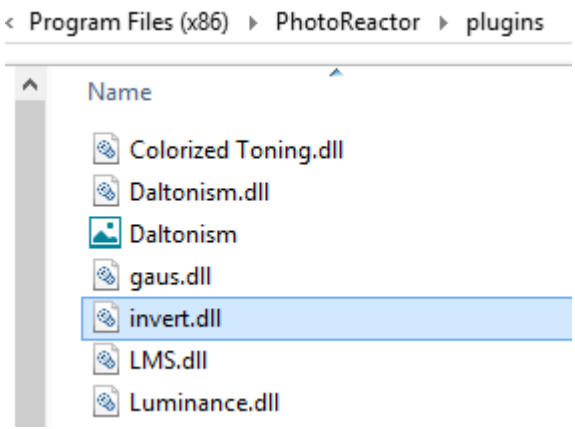


You will find your plugin.dll, we have just compiled our first plug-in, let's rename your plug-in to something else like invert.

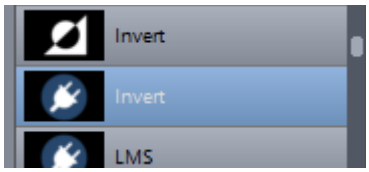


That's more like it.

All we need to do is move your plug-in to the Photo-Reactor Directory / plugins



There we go, now just open Photo-Reactor.



Scroll down and you will find our invert filter. Ours is the bottom with the default icon.

Try it an test, sure does a swell job. Sadly, it's redundant. However, we have programmed our first plug-in, and you should be proud of that!

Luminance Filter

As I stated earlier, we will be using an example from a plug-in already created “Luminance”. This sample is so you can get those creative juices flowing as there a a number of things that can be done just with this sample as a root.

Let's introduce Matrix Multiplication with the formula

$$\text{Luma} = 0.299 * 0.587 * 0.114 \quad \begin{matrix} R' \\ G' \\ B' \end{matrix}$$

Looks fancy, this is the formula for YCbCr, in this formula Y is the luminance. However, this is a rather simple algebra formula.

All this formula really is

(0.299 * R') + (0.587 * G') + (0.114 * B')

You should notice 2 things here and there is one more that is not quite as apparent.

First notice that all the colors do not carry the same weight. Green carries the most weight, followed by Red and finally Blue. This has been tested and calculated by color scientists and engineer's back in the 1950's when the NTSC standard was created. This was based on the XYZ color space that was created in the 1930's before any video standards were created.

Those numbers are how TV's display Black and White images.

However that is just one formula for displaying luminance.

The second thing that should be noticed is that when all added together 0.299 + 0.587 + 0.114 the sum is 1.

This brings up the next things that is not necessarily noticed, Each of the R, G and B have a little apostrophe ' beside them. This means that the color channel is normalized between 0 and 1 floating point. This means 0 is black, 1 is white and .5 would be gray.

How would we calculate that, fairly easy, we make the variable a floating point variable and the we divide it by it's color depth so that

```
red' = red / 255;
green' = green / 255;
blue' = blue / 255;
```

Here are the Matrix for some commonly used color spaces.

ITU-R BT.601-5 Luminance = (0.299 * R') + (0.587 * G') + (0.114 * B')
ITU-R BT.709 Luminance = (0.2126 * R') + (0.7152 * G') + (0.0722 * B');
SMPTE 240M-1995 Luminance = (0.202 * R') + (0.701 * G') + (0.087 * B');
YES Luminance = (0.253 * R') + (0.684 * G') + (0.063 * B');
Sony Triton Luminance = (.3346 * R') + (.6654 * G') + (.0161 * B');
XYZ Luminance = (0.2126 * R') + (0.7152 * G') + (0.0722 * B');

We we have matrix multiplication down pretty easy. However, there are quite a few different color spaces out there that are not calculated with a matrix. In fact, we are going to need some math functions.

Introducing the Include

C and C++ are built on libraries meaning routines. These routines are like calling an additional software package to do some of the work for you. There are many different libraries out there, including one of the most standard ones used for image processing – ***math.h***. In fact I cannot stress how important it is having these math functions available to you. I place them in just about every thing I write, just because I might need it, and I most likely do.

How do we make sure that we have this function, it's easy.

At the top of your plugin.cpp file, you should find a line that looks like this

```
#include "stdafx.h"
#include "Iplugin.h"
```

These lines include stdadx and Iplugin into your plug-in. Stdadx is a standard include and its useage is for precompilied headers. Iplugin.h would be a set of routines for plug-ins.

How then do we place the ability to call math functions? Simply place

```
#include <math.h>
```

Just below the other includes and you are set.

The beginning of your code should now look like

```
#include "stdafx.h"
#include "IPlugin.h"
#include <math.h>
```

Other Color Spaces

Back to color spaces now. Just by using matrix calculations, we can calculate many, however there are some color spaces that need some additional calculations. We can calculate some additional color spaces using the new math functions.

HSI (Hue, Saturation, Intensity) can be calculated without using the new math include, but I will place it here any way. What we are looking for is Intensity.

HSI Luminance = (temp_red + temp_green + temp_blue) / 3;

HSV does need the math routines to use MAX (the maximum value of each Red, Green and Blue).

HSV Luminance = max(R', G', B')

How about its opposite

MIN Luminance = min(R', G', B')

min is the minimum of value of each Red, Green and Blue.

Then there is the 800 pound gorilla of the color spaces LAB. That formula is

$$L^\star = 116 f(Y/Y_n) - 16$$

Boy that looks complex from the formula and it is a little trickier, but others did it so why can't we.

You homework assignment is to read up on the LAB color space and the XYZ color space from Wikipedia.

Well first we convert your RGB color space to the XYZ color space. Then we will convert the XYZ color space to the LAB color space.

I snagged this formula from a site called, www.easyrgb.com

xyz Luminance = (.2126 * R') + (.7152 * G') + (0.0722 * B');

reference_Y = xyzY / 1.0;

```
//standard D65 L*A*B*
if ( ref_Y >= 0.008856 )
        ref_Y = pow(( ref_Y / 1.0) , (1.0/3.0));
else
        ref_Y = ( 7.787 * ref_Y ) + ( 16.0 / 116.0 );
```

LAB Luminance = ((116.0 * ref_Y) - 16.0) /100;//standard lab is from 0-100 we need it from 0-1 so divide by 100

So on our Luminance plug-in, we have talked about a lot of formulas, there will be one more formula to discuss for color spaces and then we will talk about some actual code.

SRGB to RGB conversion.

SRGB is how the monitors work with just about any computer and all it is is an adjustment of the Gamma. Again, this was formulated by engineers and scientists far smarter than me. However, we need to convert from sRGB to RGB before we apply the formulas to and then afterwards convert back to sRGB from RGB.

Not too difficult to do however. Here's the formula to convert from sRGB to RGB

```
Color = pow(color,(float)(1.0/2.2));
```

to convert back

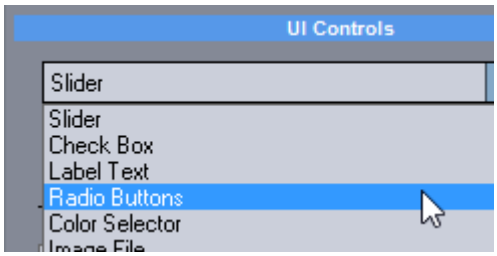
```
Color = pow(Color, (float)2.2);
```

Bringing it all together

When we created the Invert plug-in, we slowly stepped through everything, this time we will go a little faster in 25 quick steps.

Here we go.

1) Download the SDK again and place it in the Visual Studio projects folder.
2) Rename the SDK folder to Luminance
3) Delete the unused files.
4) Open Photo-Reactor
5) Drill to Tools – Generate Source Code
6) Name the Class Name to Luminance
7) In the UI controls pull down Radio Buttons



8) Select the Add New button
9) In the Text section of the UI controls paste the following text values
   0 ITU-R BT.601-5
   1 ITU-R BT.709
   2 SMPTE 240M-1995
   3 YES|4 Sony Triton Sim
   5 HSI
   6 HSV
   7 MIN
   8 Ac1c2
   9 LMS Long
   10 Erik Reinhard LAB
   11 Hunter LAB
   12 LAB
   13 Andy Special
   14 Andy Special 2
   15 Andy Special 3
   16 Andy Special 4
   17 Andy Special 5
10) Select the Fast process flag.
11) Name your Internal Variable Name LUMA
12) In your Globals, give a Title of Luminance
13) In your Globals, give a description of "Extracts the Luminance from a RGB image"
14) Select the Generate Code Button and save the code to the SDK Folder under a different name
15) Exit Photo-Reactor
16) open your newly generated source code in Notepad
17) open Visual studio
18) Convert the plug-in to the version of Visual Studio you have.

19) Open the Plugin.CPP in Visual Studio
20) Paste your code from Notepad to the plugin.cpp
21) In the Plugin ID section give your plug-in an unique identifier.
22) In the `virtual void Process_Data` that performs the processing paste the below code. The code is found in between the code block section.
23) In Visual Studio Build solution, making sure you are set to “release”
24) In your Release folder of Visual Studio, rename your plug-in, to Luminance.DLL
25) Copy your plug-in to the Photo-Reactor Plugins folder and open Photo-Reactor to test your plug-in

----------Begin code block

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight,
UIParameters* pParameters)
{
	// this is just example to desaturate and to adjust the desaturation with slider
	// Get the latest parameters

	//List of Parameters
	int nluma = (int)GetValue(PARAM_LUMA);
	float temp_value_red;
	float temp_value_green;
	float temp_value_blue;

	float temp_red;
	float temp_green;
	float temp_blue;

	float luma = 0;

	int luminance = 0;

	float xyzX;
	float xyzY;
	float xyzZ;

	float ref_Y;

	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];

			temp_value_red = (float)nR / 255.0;
			temp_value_green = (float) nG / 255.0;
			temp_value_blue = (float) nB / 255.0;

			temp_red = pow(temp_value_red,(float)(1.0/2.2));
			temp_green = pow(temp_value_green,(float)(1.0/2.2));
			temp_blue = pow(temp_value_blue,(float)(1.0/2.2));

			if (nluma == 0)//ITU-R BT.601-5
			{
			luma = (0.299 * temp_red) + (0.587 * temp_green) + (0.114 * temp_blue);
			}
			if (nluma == 1)//ITU-R BT.709
			{
			luma = (0.2126 * temp_red) + (0.7152 * temp_green) + (0.0722 * temp_blue);
			}
			if (nluma == 2)//SMPTE 240M-1995
			{
			luma = (0.202 * temp_red) + (0.701 * temp_green) + (0.087 * temp_blue);
			}
```

```
                    if (nluma == 3)//YES
                    {
                    luma = (0.253 * temp_value_red) + (0.684 * temp_value_green) + (0.063 *
temp_value_blue);
                    }
                    if (nluma == 4)//Sony Triton Sim
                    {
                    luma = (.3346 * temp_red) + (.6654 * temp_green) + (.0161 * temp_blue);
                    }
                    if (nluma == 5)//HSI
                    {
                    luma = (temp_red + temp_green + temp_blue) / 3;
                    }
                    if (nluma == 6)//HSV
                    {
                    float temp_luma = max(temp_red,temp_green);
                    luma = max(temp_luma,temp_blue);
                    }
                    if (nluma == 7)//MIN
                    {
                    float temp_luma = min(temp_red,temp_green);
                    luma = min(temp_luma,temp_blue);
                    }

                    if (nluma == 8)//Ac1c2
                    {
                    xyzX = (0.4124 * temp_red) + (0.3576 * temp_green) + (0.1805 * temp_blue);
                    xyzY = (0.2126 * temp_red) + (0.7152 * temp_green) + (0.0722 * temp_blue);
                    xyzZ = (0.0193 * temp_red) + (0.1192 * temp_green) + (0.9505 * temp_blue);
                    luma = (0.2787 * xyzX) + (0.7218 * xyzY) + (-0.1066 * xyzZ) ;
                    }

                    if (nluma == 9)//LMS Long
                    {
                    luma = (0.3811 * temp_red) + (0.5783 * temp_green) + (0.0402 * temp_blue);
                    }

                    if (nluma == 10)//Erik Reinhard LAB
                    {
                    xyzX =  (0.3811 * temp_value_red) + (0.5783 * temp_value_green) + (0.0402 *
temp_value_blue) ;
                    xyzY =  (0.1967 * temp_value_red) + (0.7244 * temp_value_green) + (0.0782 *
temp_value_blue) ;
                    xyzZ =  (0.0241 * temp_value_red) + (0.1288 * temp_value_green) + (0.8444 *
temp_value_blue) ;

                           ref_Y = (1.0 * xyzX) + (1.0 * xyzY) + (1.0 * xyzZ);

                           luma = (1/(sqrt(3.0))) * ref_Y;
                    }
```

```
                    if (nluma == 11)//Hunter LAB
                    {
                    xyzX = (0.4124 * temp_value_red) + (0.3576 * temp_value_green) + (0.1805 *
temp_value_blue);
                    xyzY = (0.2126 * temp_value_red) + (0.7152 * temp_value_green) + .0722 *
temp_value_blue);
                    xyzZ = (0.0193 * temp_value_red) + (0.0193 * temp_value_green) + (0.1192 *
temp_value_blue);

                    luma = 1 * sqrt (xyzY / 1.0);
                    }

                    if (nluma == 12)//LAB
                    {
                    // LAB formula concept from http://www.easyrgb.com

                    //convert this to XYZ (only worry about Y
                    xyzY = (.2126 * temp_value_red) + (.7152 * temp_value_green) + (0.0722 *
temp_value_blue);

                    ref_Y = xyzY / 1.0;// Yn is defined as 1.0 in all LAB formulas I've seen

                    //standard D65 L*A*B*
                    if ( ref_Y >= 0.008856 )
                    ref_Y = pow(( ref_Y / 1.0) , (1.0/3.0));
                    else
                    ref_Y = ( 7.787 * ref_Y ) + ( 16.0 / 116.0 );

                    luma = ((116.0 * ref_Y ) - 16.0) /100;//standard lab is from 0-100 we need
it from 0-1 so divide by 100
                    // trying to normalize lab normally from 0-100 to 0 to 1
                    // just by diving by 100
                    }
                    if (nluma == 13)//Andy Special
                    {
                    xyzY = (0.2126 * temp_red) + (0.7152 * temp_green) + (0.0722 * temp_blue);
                    xyzX = sqrt(xyzY)*1.0;
                    luma =  pow((xyzX), (float)(4.0 / 3.0));
                    }

                    if (nluma == 14)//Andy Special 2
                    {
                    xyzX = (0.4124 * temp_value_red) + (0.3576 * temp_value_green) + (0.1805 *
temp_value_blue);
                    xyzY = (0.2126 * temp_value_red) + (0.7152 * temp_value_green) + (0.0722 *
temp_value_blue);
                    xyzZ = (0.0193 * temp_value_red) + (0.0193 * temp_value_green) + (0.1192 *
temp_value_blue);;

                    ref_Y = (1.0 * xyzX) + (1.0 * xyzY) + (1.0 * xyzZ);
                    luma = (1/(sqrt(3.0))) * ref_Y;
                    }
```

```
            if (nluma == 15)//Andy Special 3
            {
            xyzX = (0.4306 * temp_value_red) + (0.3415 * temp_value_green) + (0.1784 *
temp_value_blue);
            xyzY = (0.2220 * temp_value_red) + (0.7067 * temp_value_green) + (0.0713 *
temp_value_blue);
            xyzZ = (0.0202 * temp_value_red) + (0.1295 * temp_value_green) + (0.9394 *
temp_value_blue);;

            ref_Y = (1.0 * xyzX) + (1.0 * xyzY) + (1.0 * xyzZ);

            luma = (1/(sqrt(3.0))) * ref_Y;
            }

            if (nluma == 16)//Andy Special 3 CIE RGB D50
            {
            xyzX = (0.4868870 * temp_value_red) + (0.3062984 * temp_value_green) + (0.1710347
* temp_value_blue);
            xyzY = (0.1746583 * temp_value_red) + (0.8247541 * temp_value_green) + (0.0005877
* temp_value_blue);
            xyzZ = (-0.0012563 * temp_value_red) + (0.0169832 * temp_value_green) +
(0.8094831 * temp_value_blue);;

            ref_Y = (1.0 * xyzX) + (1.0 * xyzY) + (1.0 * xyzZ);
            luma = (1/(sqrt(3.0))) * ref_Y;
            }

            if (nluma == 17)//Andy Special 4 CIE RGB E
            {
            xyzX = (0.4887180 * temp_value_red) + (0.3106803 * temp_value_green) + (0.2006017
* temp_value_blue);
            xyzY = (0.1762044 * temp_value_red) + (0.8129847 * temp_value_green) + (0.0108109
* temp_value_blue);
            xyzZ = (0.0000000 * temp_value_red) + (0.0102048 * temp_value_green) + (0.9897952
* temp_value_blue);;

            ref_Y = (1.0 * xyzX) + (1.0 * xyzY) + (1.0 * xyzZ);
            luma = (1/(sqrt(3.0))) * ref_Y;
            }

            //convert to srgb
            float luma1 = pow(luma, (float)2.2);

            luminance = (int)(luma1 * 255);

            int nA = CLAMP255(luminance);

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

pBGRA_out[nIdx+CHANNEL_B] = nA;
                    }
            }
      }
```
----------end code block

User Interface

One of the most important things to the user experience with your filter is the user interface. You need to get the information passed on by Photo-Reactor to your filter. We listed the controls earlier in this document
They were

Slider, check box, Label Text, radio buttons, color selector, image file, Check box + enable all following, label edit box, combo box, Font Combo, multi line text, exponential slider, push button, gamma slider, logarithmic slider, position control, integer input, multi-line edit box, check box, + enable all / until, horizontal space.

We used radio buttons with our Luminance filter. When you pressed a button, a variable was passed to the main loop. The main loop compared this with the **if** statement

ex.

```
if (nluma == 0)//ITU-R BT.601-5
{
        luma = (0.299 * temp_red) + (0.587 * temp_green) + (0.114 * temp_blue);
}
if (nluma == 1)//ITU-R BT.709
{
        luma = (0.2126 * temp_red) + (0.7152 * temp_green) + (0.0722 * temp_blue);
}
```

The variable *nluma* is sent to the loop from earlier in the source code via the line

```
int nluma = (int)GetValue(PARAM_LUMA);
```

This is where nluma is defined and is nothing more than a copy of the variable (PARAM_LUMA).

Now I could have possibly made the if statements

```
if ((int)GetValue(PARAM_LUMA) == 0)//ITU-R BT.601-5
        {
                luma = (0.299 * temp_red) + (0.587 * temp_green) + (0.114 * temp_blue);
        }
```

However that is harder to read and harder to trouble shoot. We want our programs to be readable by us so that we can troubleshoot later on (to the end user it makes no difference, as they don't see this at all).

A good programming rule is to remember to make sure your code is readable to yourself and to anyone else who may view your code.

This goes without saying, make sure to comment your code.

Back to the subject at hand.

The user control for the radio buttons is first created by the Photo-Reactor generated code

```
int GetUIParameters (UIParameters* pParameters)
        {

                // label, value, min, max, type_of_control, special_value
                // use the UI builder in the software to generate this

                AddParameter( PARAM_LUMA ,"0 ITU-R BT.601-5|1 ITU-R BT.709|2 SMPTE 240M-1995|3 YES|4 Sony Triton
Sim|5 HSI|6 HSV|7 MIN|8 Ac1c2|9 LMS Long|10 Erik Reinhard LAB|11 Hunter LAB|12 LAB|13 Andy Special|14 Andy
Special 2|15 Andy Special 3|16 Andy Special 4|17 Andy Special 5", 0, 0, 17, TYPE_ONEOFMANY, 0);


                return NUMBER_OF_USER_PARAMS;
        }
```

The long line AddParameter is where the magic happens. Let's disassemble it.

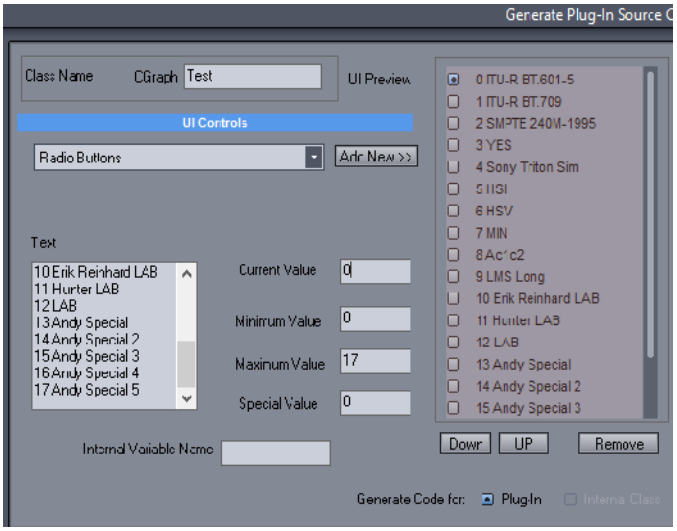
AddParameter, this is defined earlier and its purpose is a interface between Photo-Reactor and your plugin to pull the variable when the user selects a control.

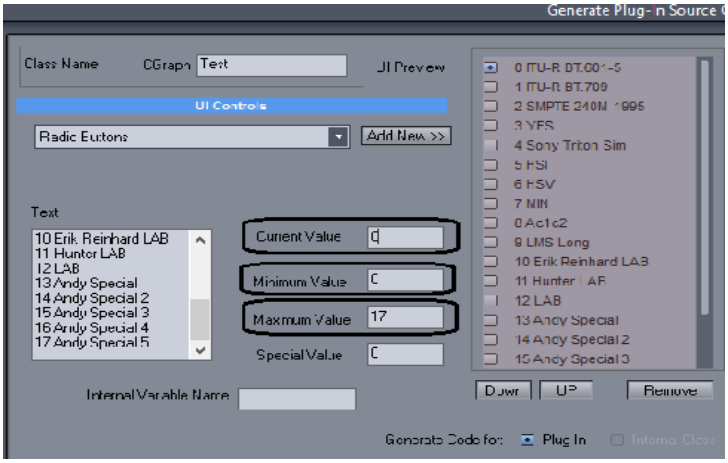
PARAM_LUMA, is the name of the variable passed onto your processing loop under list of parameters.

"0 ITU-R BT.601-5|1 ITU-R BT.709|2 SMPTE 240M-1995|3 YES|4 Sony Triton Sim|5 HSI|6 HSV|7 MIN|8 Ac1c2|9 LMS Long|10 Erik Reinhard LAB|11 Hunter LAB|12 LAB|13 Andy Special|14 Andy Special 2|15 Andy Special 3|16 Andy Special 4|17 Andy Special 5"
Here are the Names associated with all of your controls, in this example there are 18 different controls, the symbol | is a separator.

0, 0, 17,
The first number is the current value, this is defined in the “General Source Code” of Photo-Reactor, under the heading UI controls, the second number is the minimum value and the last is the maximum value.

Remember this interface?





Here are where those values are generated.

The very last number after the command `TYPE_ONEOFMANY` is the Special Value, it is defined in this control as 0.

Programming Controls

Now that we know how the controls are defined, lets expand on that by working with some controls.

Action Controls

An Action controls is a user interface (UI) control that performs an action such as a sliding control that adjust the brightness of an image or perhaps select a different internal routine.

Please Note the following

In order to use any of these test, it is assumed that you have the following include

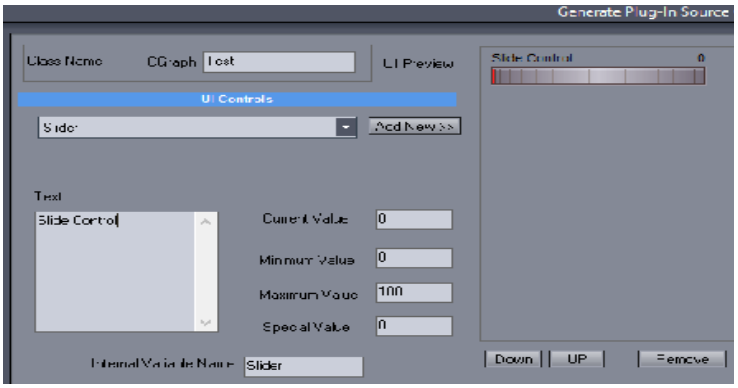
#include <stdio.h>

However, on your final code you do not need to use this include unless you need it.

It is meant for the purposes of these control demos to use the messagebox outputs.

Slider

This is used to set a user defined value to your routine. It could be used to set a radius or strength of an effect. Think of it as a volume control, with the minimum and maximum set by you the programmer.



In this case the minimum is 0 and the maximum is 100.

When the code is generated, it creates the line

```
AddParameter( PARAM_SLIDER ,"Slide Control", 0.0, 0.0, 100.0, TYPE_SLIDER, 0.0);
```

Here we can see that the values are:
Name of Value PARAM_SLIDER (even though we named the variable just Slider).
Name presented to the end user “Slide Control”
Current Value = 0
Minimum Value = 0
Maximum Value = 100
Type of control is Slider
Special Value = 0

We are going to use this for our example

If we move further down the code we will see the lines

```
//List of Parameters
double dslider = GetValue(PARAM_SLIDER);
```

Here Photo-Reactor has automatically defined a variable dslider for you. The d in dslider means that it is a double. What is a double, it is a double precision floating point number with a 15 digit precision.

Testing the Slide control

Create a slider code using the example above with the control named Slider

At the top of your plug-in code, add #include <stdio.h>

```
// plugin.cpp : Defines the entry point for the DLL application.
//

#include "stdafx.h"
#include "IPlugin.h"
#include <stdio.h>
```

In your GetPluginName routine paste the following

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

In your GetPluginID() routine paste the following

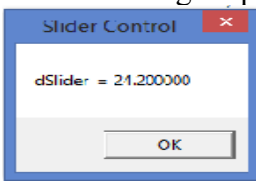
```
extern "C" __declspec(dllexport) char* GetPluginID()
{

//	IMPORTANT:you have to fill unique ID for every plugin:
//	The ID must be unique or loading and saving will not be able to find correct plugin
//  Comment out this line below so you can compile
//	ATTENTION
	return "com.lumafilters.test";
}
```

****Note that these first three steps are required for all of the control tests*

In your processing loop, paste the following

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
	// Get the latest parameters

	//List of Parameters
	double dSlider = GetValue(PARAM_SLIDER);

	for (int y = 0; y< nHeight; y++)
	{
		for (int x = 0; x< nWidth; x++)
		{
			int nIdx = x*4+y*4*nWidth;
		}
	}
		char sBuffer1[100]; sprintf(sBuffer1,
		"dSlider  = %f" ,	dSlider
		);MessageBox(NULL,sBuffer1,"Slider Control", MB_OK);
}
```

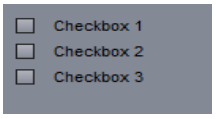
Compile and move your DLL, when you run Photo-Reactor, load your image, select your new routine, and move your slider to 24, you should get the following output.

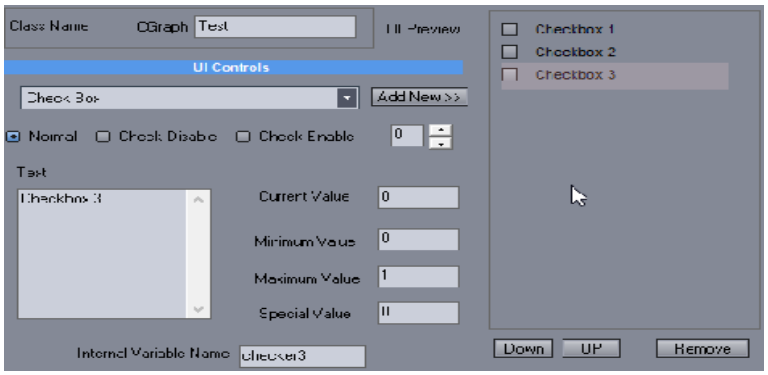


That is the expected output.

Checkbox

This is used to set a selection or multiple selections to your routine. It could be used to set an option. Think of it as a check list.





To test, create 3 check box items and name each as Checkbox 1, Checkbox 2 and Checkbox. Name each variable as checker1, checker2 and checker3.

The control that it sets is

```
AddParameter( PARAM_CHECKER1 ,"Checkbox 1", 0, 0, 1, TYPE_CHECKBOX, 0);
AddParameter( PARAM_CHECKER2 ,"Checkbox 2", 0, 0, 1, TYPE_CHECKBOX, 0);
AddParameter( PARAM_CHECKER3 ,"Checkbox 3", 0, 0, 1, TYPE_CHECKBOX, 0);
```

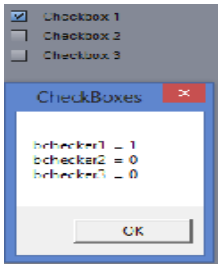
paste the following in the Virtual Void

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // Get the latest parameters

        //List of Parameters
        BOOL bchecker1 = GetBOOLValue(PARAM_CHECKER1);
        BOOL bchecker2 = GetBOOLValue(PARAM_CHECKER2);
        BOOL bchecker3 = GetBOOLValue(PARAM_CHECKER3);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }
        char sBuffer1[100]; sprintf(sBuffer1,
        "bchecker1  = %d" "\n"
        "bchecker2  = %d" "\n"
        "bchecker3  = %d" "\n",
        bchecker1,bchecker2,bchecker3
        );MessageBox(NULL,sBuffer1,"CheckBoxes", MB_OK);
}
```

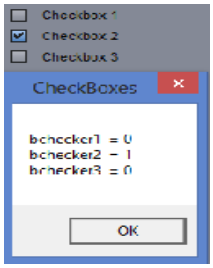
Testing the Checkbox control

Follow the same steps as the Slider control.

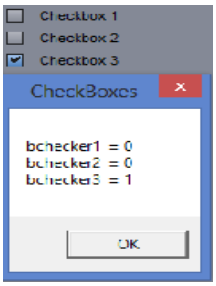
Checkbox 1 selected



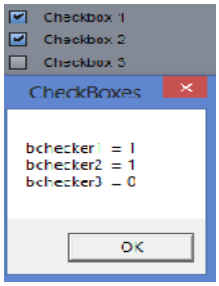
Checkbox 2 selected

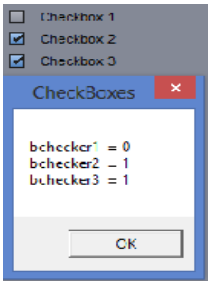


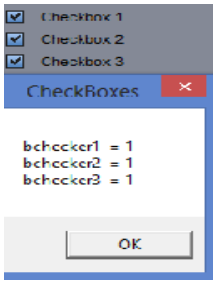
Checkbox 3 selected



How about multiples

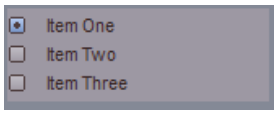


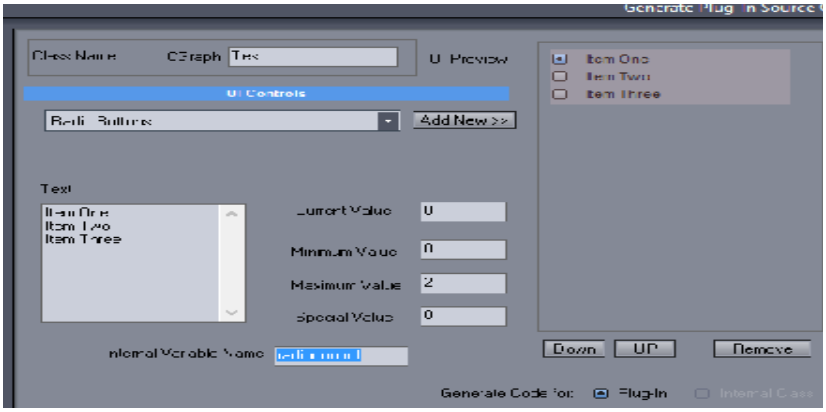




Radio Buttons

This is used to set a selection to your routine. It is used to set a single option.





To test, create 3 check box items and name each as Item One, Item Two and Item Three. Name the variable as radiocontrol.

The control that it sets is

```
AddParameter( PARAM_RADIOCONTROL ,"Item One|Item Two|Item Three", 0, 0, 2, TYPE_ONEOFMANY, 0);
```

paste the following in the Virtual Void

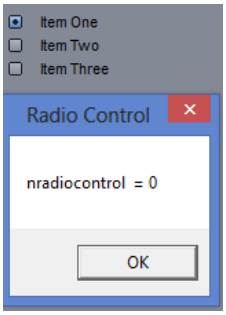
```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // Get the latest parameters

        //List of Parameters
        int nradiocontrol = (int)GetValue(PARAM_RADIOCONTROL);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }
        char sBuffer1[100]; sprintf(sBuffer1,
        "nradiocontrol  = %d" , nradiocontrol
        );MessageBox(NULL,sBuffer1,"Radio Control", MB_OK);

}
```

Output is an INT

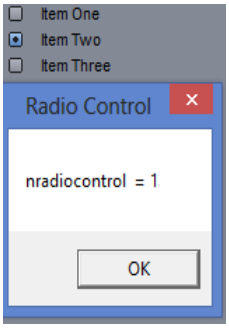
Testing the Radio Button control

Follow the same steps as the Slider control.

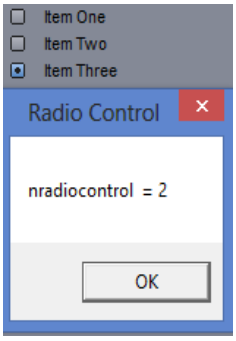
Item 1 selected, note that it starts with 0



Item 2 selected, shows 1

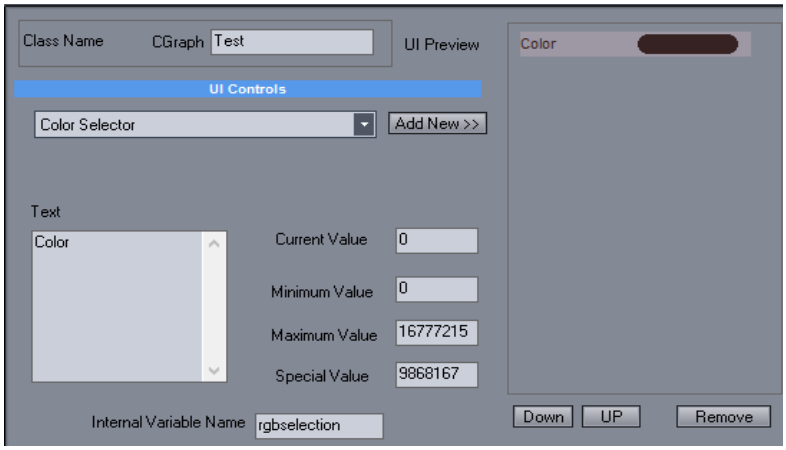


Item 3 selected, shows 2



Color Selection

This is used to select a color so that you can use it for processing.



To test, create Color selector and name it Color. Name the variable as rgbselection.

The control that it sets is

```
AddParameter( PARAM_RGBSELECTION ,"Color ", RGB(0,0,0), RGB(0,0,0), RGB(255,255,255), TYPE_COLOR,
RGB(135,147,150));
```

The output of this control is Hex, however it can be converted back to RGB.

There are at least 2 methods of doing this.

You can use the Visual Studio Macro to accomplish this or build a bit shift routine to do this I give both examples below.

To use the Macro Method uncomment the following line in the generated source code

```
#define RGB(r,g,b) ((COLORREF)(((BYTE)(r)|((WORD)((BYTE)(g))<<8))|(((DWORD)(BYTE)(b))<<16)))
```

paste the following in the Virtual Void

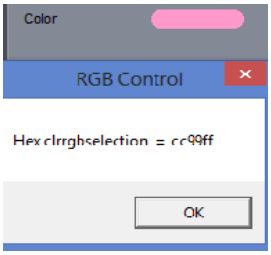
```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // Get the latest parameters

        //List of Parameters
        COLORREF clrrgbselection = (int)GetValue(PARAM_RGBSELECTION);

        //doing it the non macro method
        int colorr = (BYTE)clrrgbselection;//convert back to decimal
        int colorg = (WORD)clrrgbselection >>8;//convert back to decimal
        int colorb = (DWORD)clrrgbselection >>16;//convert back to decimal
        //doing it the non macro method

        //Macro Method
        int RR = GetRValue(clrrgbselection);
        int GG = GetGValue(clrrgbselection);
        int BB = GetBValue(clrrgbselection);
        //Macro Method

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }


        char sBuffer0[100]; sprintf(sBuffer0,
        "Hex clrrgbselection  = %x" ,   clrrgbselection
        );MessageBox(NULL,sBuffer0,"RGB Control", MB_OK);

        char sBuffer1[100]; sprintf(sBuffer1,
        "Int Red    = %d" "\n"
        "Int Green  = %d" "\n"
        "Int Blue   = %d" "\n"
        ,       colorr,colorg,colorb
        );MessageBox(NULL,sBuffer1,"Non Macro RGB Control", MB_OK);

        char sBuffer2[100]; sprintf(sBuffer2,
        "Int Red    = %d" "\n"
        "Int Green  = %d" "\n"
        "Int Blue   = %d" "\n"
        ,       RR,GG,BB
        );MessageBox(NULL,sBuffer2,"Macro RGB Control", MB_OK);
}
```

Testing the Color Selector control

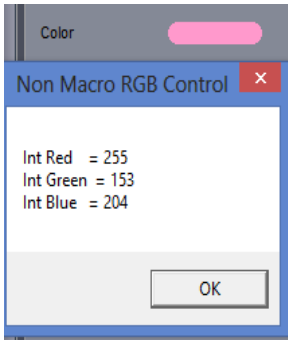
Follow the same steps as the Slider control.

With the color Rose selected

Hex output



The Non-Macro Method



The Macro Method

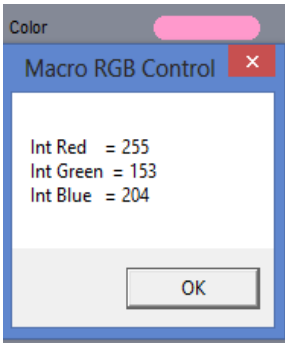
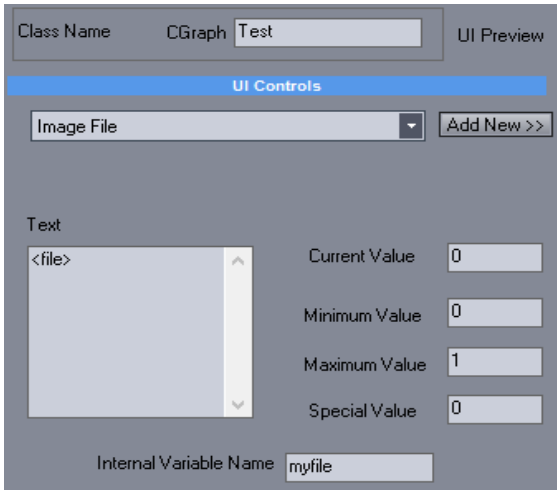


Image File

This is used open a file such as a texture so that you can use it for processing.

** Please note that at this time, this feature is more on the advanced side, Photo-Reactor will provide a dialog box to load the image, however the actual code for loading the image is up to the programmer. There are image loading libraries available on the internet, however it is up to the programmer to decide on using a library or writing their own code. Perhaps in a future version of the SDK, this will be provided.



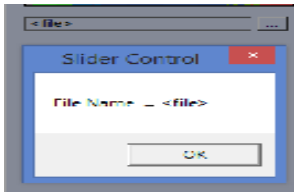
To test, create Image File Control and the default name. Name the variable as myfile.

The control that it sets is

```
AddParameter( PARAM_MYFILE ,"<file> ", 0, 0, 1, TYPE_IMGFILE, 0);

        virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters*
pParameters)
        {
                // Get the latest parameters
                //List of Parameters

                for (int y = 0; y< nHeight; y++)
                {
                        for (int x = 0; x< nWidth; x++)
                        {
                                int nIdx = x*4+y*4*nWidth;
                        }
                }

                char sBuffer1[100]; sprintf(sBuffer1,
                "File Name  = %s" ,     pParameters
                );MessageBox(NULL,sBuffer1,"Slider Control", MB_OK);

        }
```

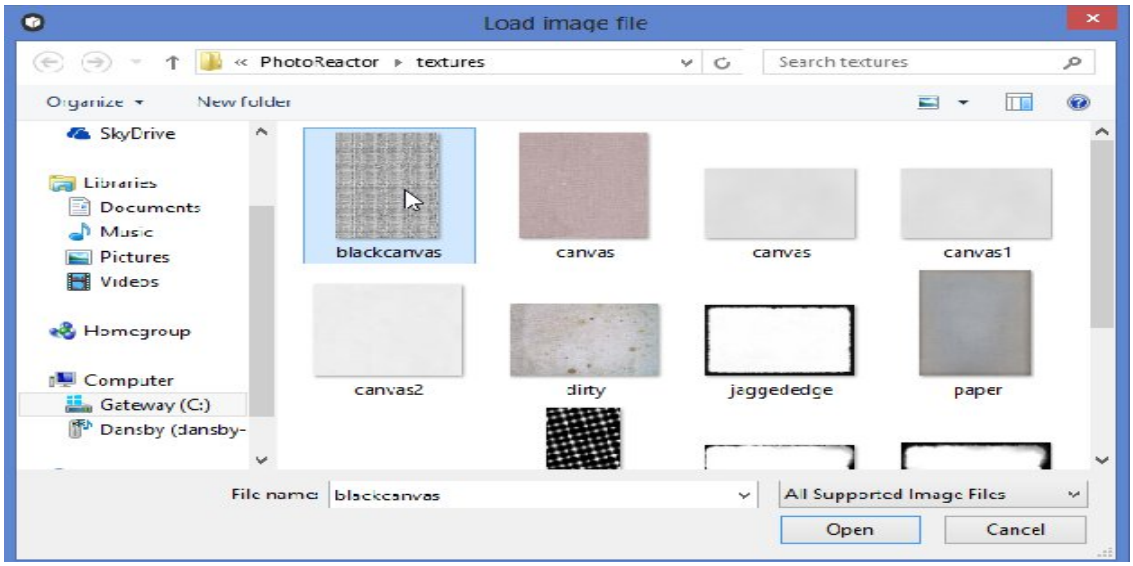
Testing Image File

What shows initially when no file loaded

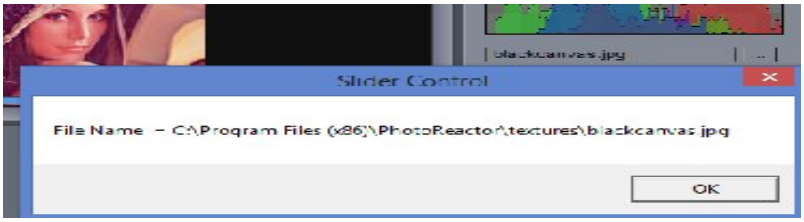


Select to get a file

Select a file



Here is your output



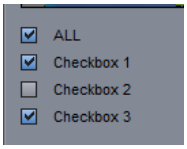
Checkbox Enable all Following

This is used to control a series of check boxes. It is used to set a multiple options, however only works if the primary first box is selected. This control is a little more complex.

All Off

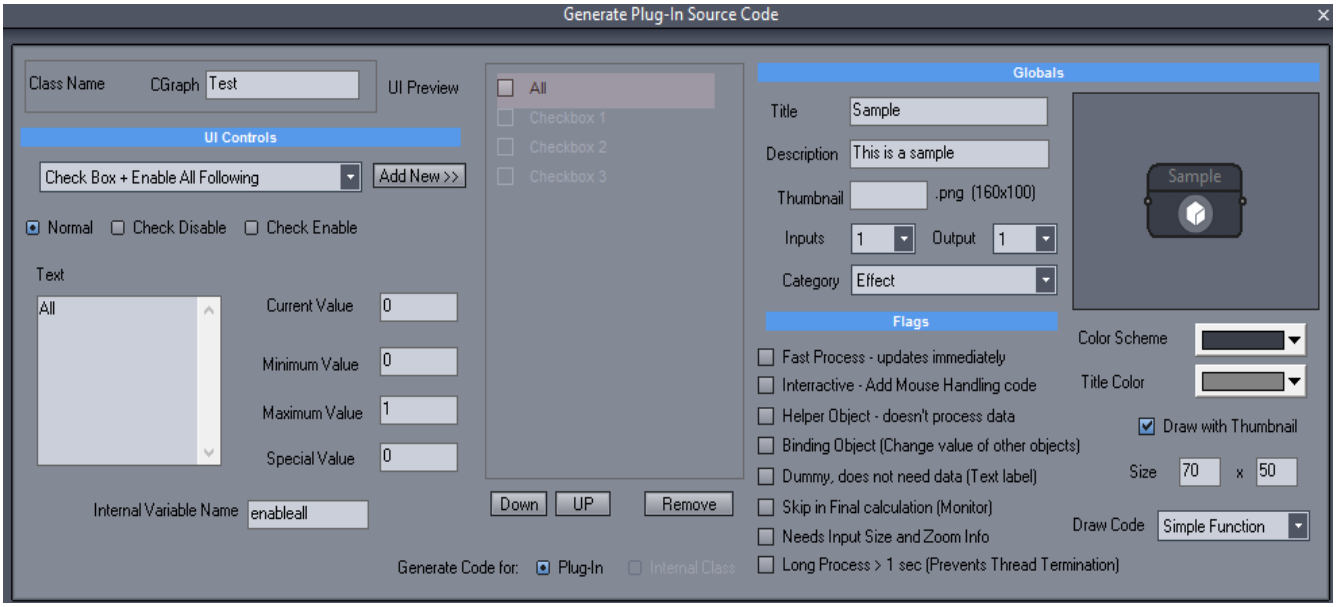
All off, however now you can make selections.

First and third option selected.



To test, create a Check Box + Enable all Following first and Name the variable as enableall.

Then create 3 standard check box items and name each as Checkbox 1, Checkbox 2 and Checkbox 3. Name the variables as checkbox1, checkbox2 and checkbox3.



Make sure that the All Button is unchecked, you may have to check it and then uncheck it by selecting the control twice.

The controls that are set

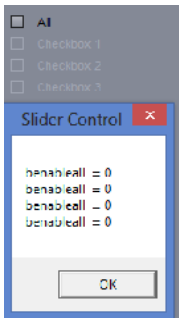
```
AddParameter( PARAM_ENABLEALL ,"All", 1, 0, 1, TYPE_CHECKBOXDISABLENEXT, 0);
AddParameter( PARAM_CHECKBOX1 ,"Checkbox 1", 0, 0, 1, TYPE_CHECKBOX, 0);
AddParameter( PARAM_CHECKBOX2 ,"Checkbox 2", 0, 0, 1, TYPE_CHECKBOX, 0);
AddParameter( PARAM_CHECKBOX3 ,"Checkbox 3", 0, 0, 1, TYPE_CHECKBOX, 0);
```

paste the following in the Virtual Void

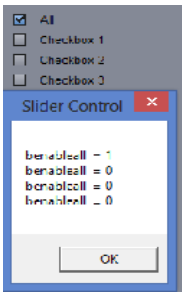
```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // Get the latest parameters

        //List of Parameters
        BOOL benableall = GetBOOLValue(PARAM_ENABLEALL);
        BOOL bCheckbox1 = GetBOOLValue(PARAM_CHECKBOX1);
        BOOL bCheckbox2 = GetBOOLValue(PARAM_CHECKBOX2);
        BOOL bCheckbox3 = GetBOOLValue(PARAM_CHECKBOX3);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                }
        }
        char sBuffer1[100]; sprintf(sBuffer1,
        "benableall  = %d" "\n"
        "benableall  = %d" "\n"
        "benableall  = %d" "\n"
        "benableall  = %d" "\n"
        ,
        benableall,
        bCheckbox1,
        bCheckbox2,
        bCheckbox3
        );MessageBox(NULL,sBuffer1,"Slider Control", MB_OK);
}
```

Testing Checkbox Enable all Following

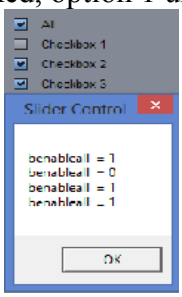
All Unchecked



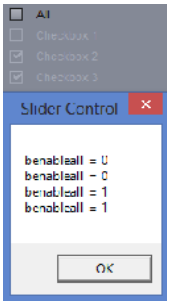
All Checked, everything else unchecked



All Checked, option 2 and option 3 checked, option 1 unchecked.



All unchecked, but option 2 and 3 remains checked.

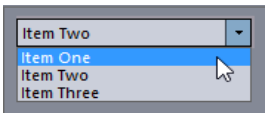


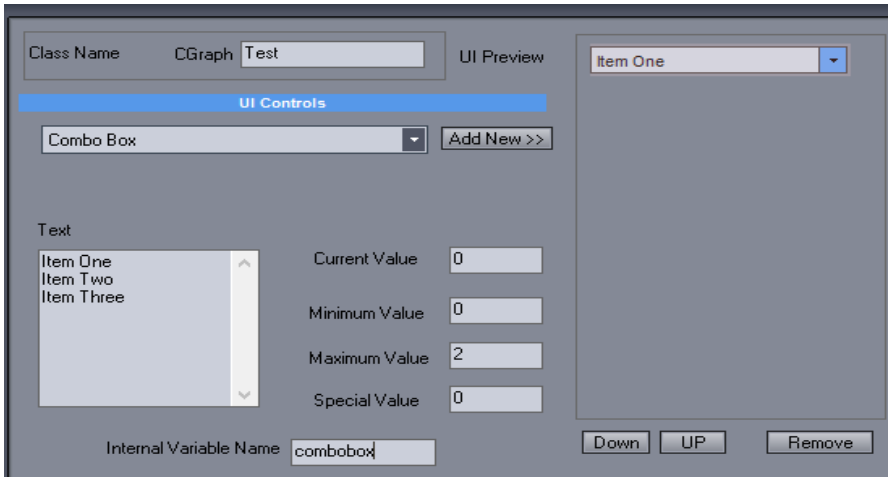
There are a couple of things that should be noticed. Simply un-checking the All Button, has no effect on the Bool for option 2 or 3. It simply means that the end user cannot change the control once the all button is unselected.

If you want to un-check the All button and then disable the following options, you will need to accommodate this in programming by using the AND operand.

Combo Box

This is used to set a selection to your routine. It is used to set a single option. Works exactly like Radio Buttons, the difference is a pull down box.





To test, create 3 check box items and name each as Item One, Item Two and Item Three. Name the variable as combobox.

The control that it sets is

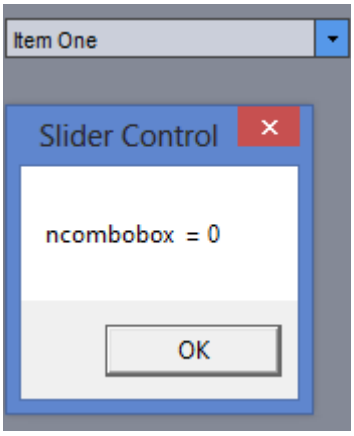
```
AddParameter( PARAM_COMBOBOX ,"Item One|Item Two|Item Three", 0, 0, 2, TYPE_COMBO, 0);
```

paste the following in the Virtual Void

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // this is just example to desaturate and to adjust the desaturation with slider
        // Get the latest parameters

        //List of Parameters
        int ncombobox = (int)GetValue(PARAM_COMBOBOX);
        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }
        char sBuffer1[100]; sprintf(sBuffer1,
        "ncombobox  = %d" ,     ncombobox
        );MessageBox(NULL,sBuffer1,"Slider Control", MB_OK);
}
```

Testing Combo Box

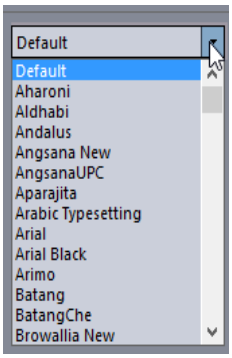
It will test the same way as Radio Buttons

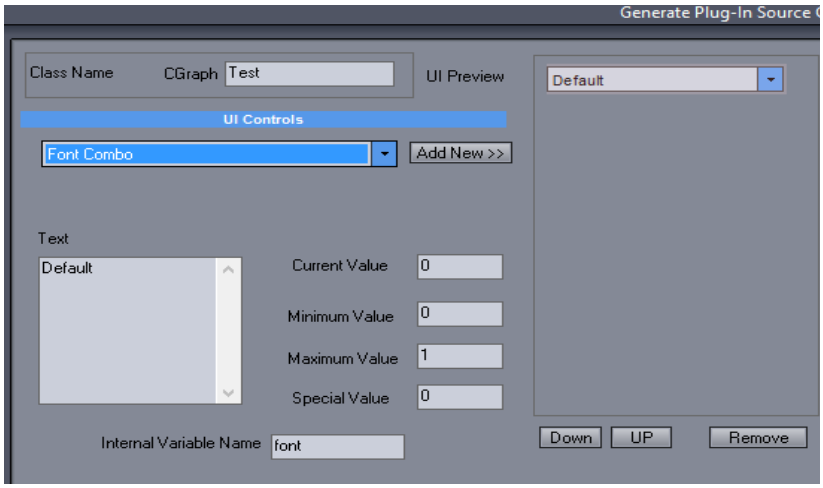


Font Combo

This is used to choose a font.

** Please note that at this time, this feature is more on the advanced side, Photo-Reactor will provide a dialog box to select a font and will provide the font the end user selects, however the actual code for placing text on an image and using the selected font is up to the programmer. Perhaps in a future version of the SDK, this will be provided. Media Chance has noted that drawing text to an image buffer is **NOT** a trivial operation as there is no simple function that can do that in plain c++. The largest problem is reliably converting the HBITMAP to the BGRA buffer.





To test, create a Font Combo. Name the variable as font.

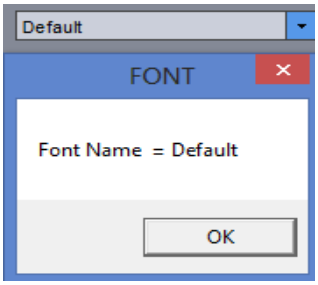
The control that it sets is

```
AddParameter( PARAM_FONT ,"Default ", 0, 0, 1, TYPE_FONTCOMBO, 0);
```

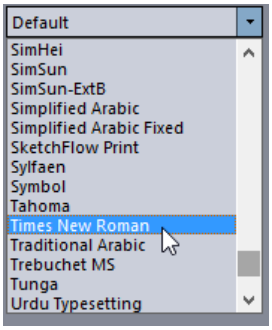
paste the following in the Virtual Void

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // Get the latest parameters

        //List of Parameters

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }
        char sBuffer1[100]; sprintf(sBuffer1,
        "Font Name  = %s" ,     pParameters
        );MessageBox(NULL,sBuffer1,"FONT", MB_OK);

}
```

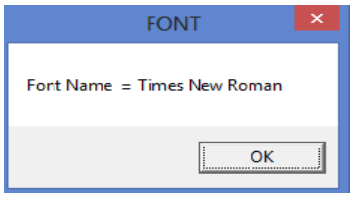
Testing the Font Control



No font selected.



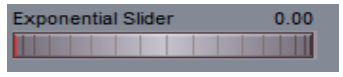
Choose Times New Roman

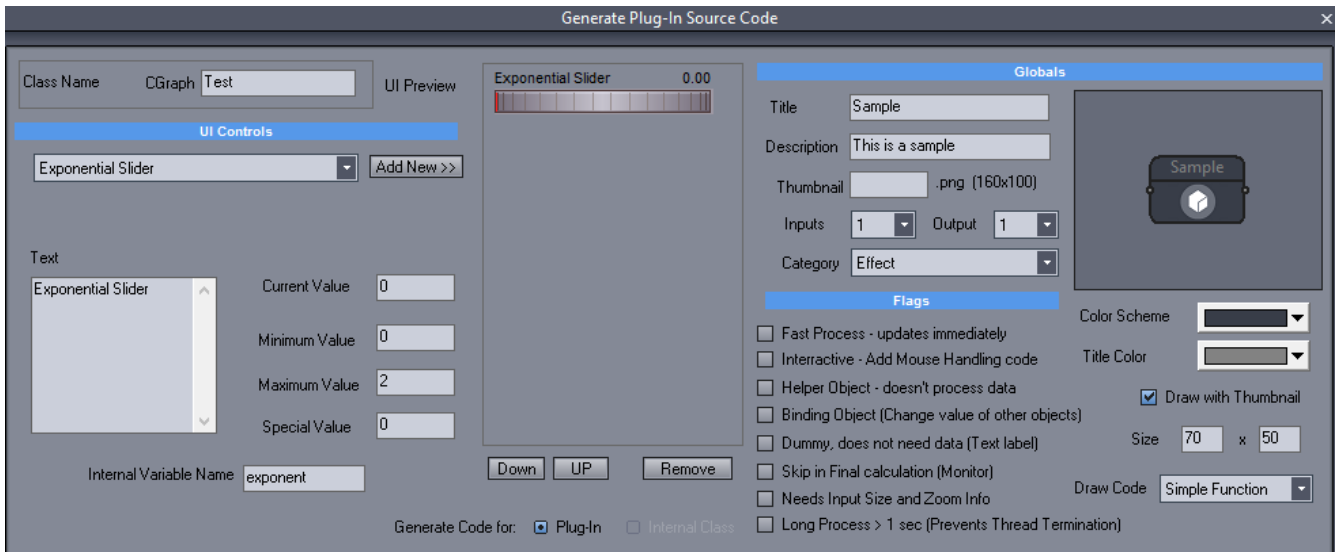


Exponential Slider

This control works almost exactly the same as a Slider.

This is used to set a user defined value to your routine. It could be used to set a radius or strength of an effect. Think of it as a volume control, with the minimum and maximum set by you the programmer.





To test, create a control with the current value as 0, minimum as 0, maximum as 2 and special as 0. Name the variable as exponent.

The control that it sets is

```
AddParameter( PARAM_EXPONENT ,"Exponential Slider", 0.0, 0.0, 2.0, TYPE_EXPSLIDER, 0.0);
```

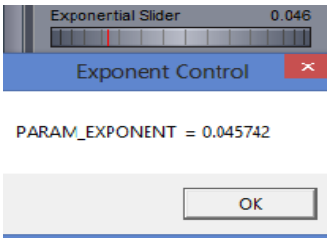
paste the following in the Virtual Void

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // Get the latest parameters

        //List of Parameters
        double dexponent = GetValue(PARAM_EXPONENT);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }
        char sBuffer1[100]; sprintf(sBuffer1,
        "PARAM_EXPONENT  = %f" ,dexponent
        );MessageBox(NULL,sBuffer1,"Exponent Control", MB_OK);
}
```

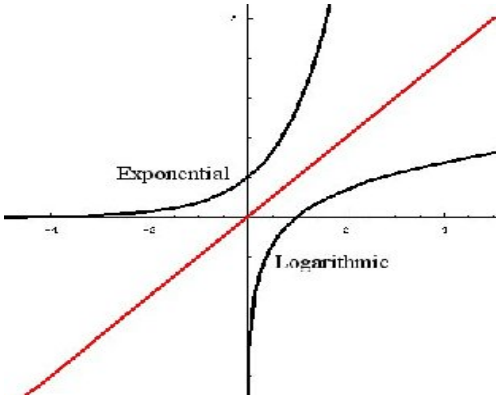
The output of the Exponential Slider is a double

Testing Exponential Control

Move your slider to .46



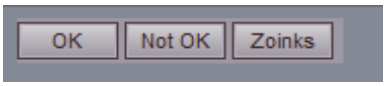
As you should realize by the name, the slider values grow exponentially.

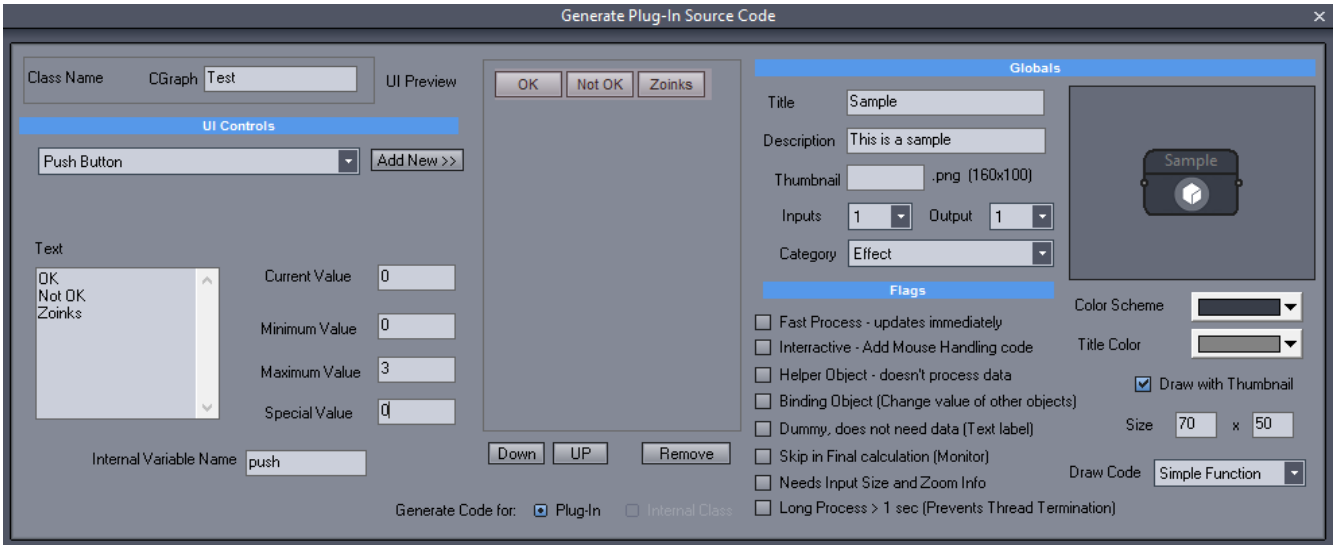


A graphical representation of the Exponential and Logarithmic curves

Push Button

This type of control is a momentary press selection good resetting all controls.





To test, create a push button and add 3 entries, OK, Not OK and Zoinks. Current Value 1, Maximum Value 3. Name your variable as push.

To test, create a control with the current value as 0, minimum as 0, maximum as 3 and special as 0. Name the variable as push.

The control that it sets is

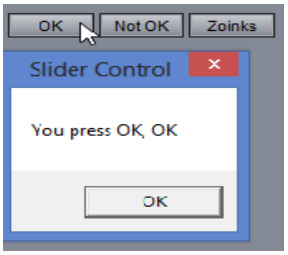
```
AddParameter( PARAM_PUSH ,"OK|Not OK|Zoinks", 0, 0, 3, TYPE_PUSHBUTTON, 0);
```

paste the following in the BOOL UIButtonPushed

```
// when button is pressed on UI, this function will be called with the parameter and sub button (for multi button line)
BOOL UIButtonPushed (int nParam, int nSubButton, UIParameters* pParameters)
{
	if (nParam == PARAM_PUSH)
	{
		// Each line can have multiple buttons: nSubButton  = 1,2,3...
		if (nParam == PARAM_PUSH )
		{
			// Each line can have multiple buttons: nSubButton  = 1,2,3...
			if (nSubButton == 1)
			{
				char sBuffer1[100]; sprintf(sBuffer1,
				"You press OK, OK"
				);MessageBox(NULL,sBuffer1,"Slider Control", MB_OK);
			}

			if (nSubButton == 2)
			{
				char sBuffer1[100]; sprintf(sBuffer1,
				"Really, Not OK"
				);MessageBox(NULL,sBuffer1,"Slider Control", MB_OK);
			}

			if (nSubButton == 3)
			{
				char sBuffer1[100]; sprintf(sBuffer1,
				"Yikes, you pressed Zoinks"
				);MessageBox(NULL,sBuffer1,"Slider Control", MB_OK);
			}
		}
	}
	return TRUE;
}
```

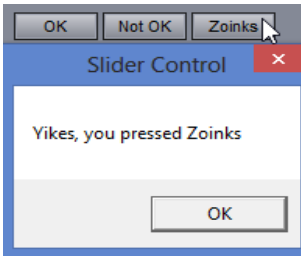
Testing Push Button Control

Press OK



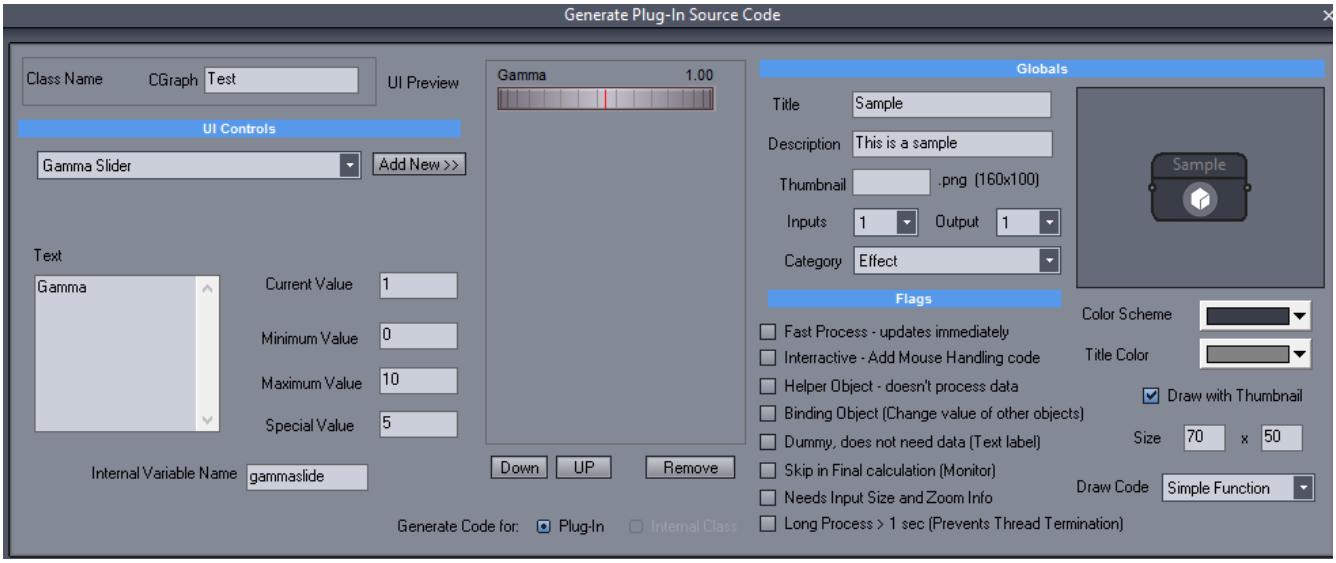
Press Not OK

Press Zoinks



Gamma Slider

As titled, this control is good for setting Gamma. It works opposite to the other sliders, counting down as you pull the slider to the right and counting up as you move the slider to the left.



To test, create a control with the current value as 0, minimum as 0, maximum as 2 and special as 0. Name the variable as gammaslide.

The control that it sets is

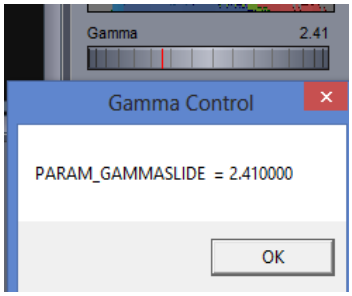
```
AddParameter( PARAM_GAMMASLIDE ,"Gamma", 1.0, 0.0, 10.0, TYPE_GAMMASLIDER, 5.0);
```

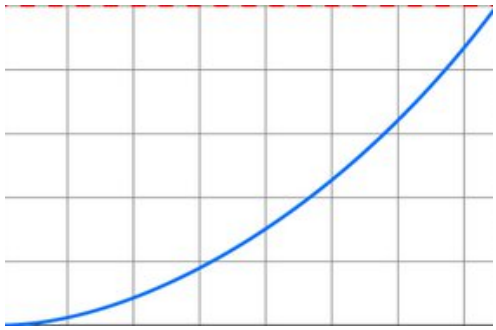
```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // Get the latest parameters
        //List of Parameters
        double dgammaslide = GetValue(PARAM_GAMMASLIDE);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }
        char sBuffer1[100]; sprintf(sBuffer1,
        "PARAM_GAMMASLIDE  = %f" ,      dgammaslide
        );MessageBox(NULL,sBuffer1,"Gamma Control", MB_OK);

}
```

The output of the Gamma Slider is a double

Testing the Gamma Slider

Move your slider to 2.41

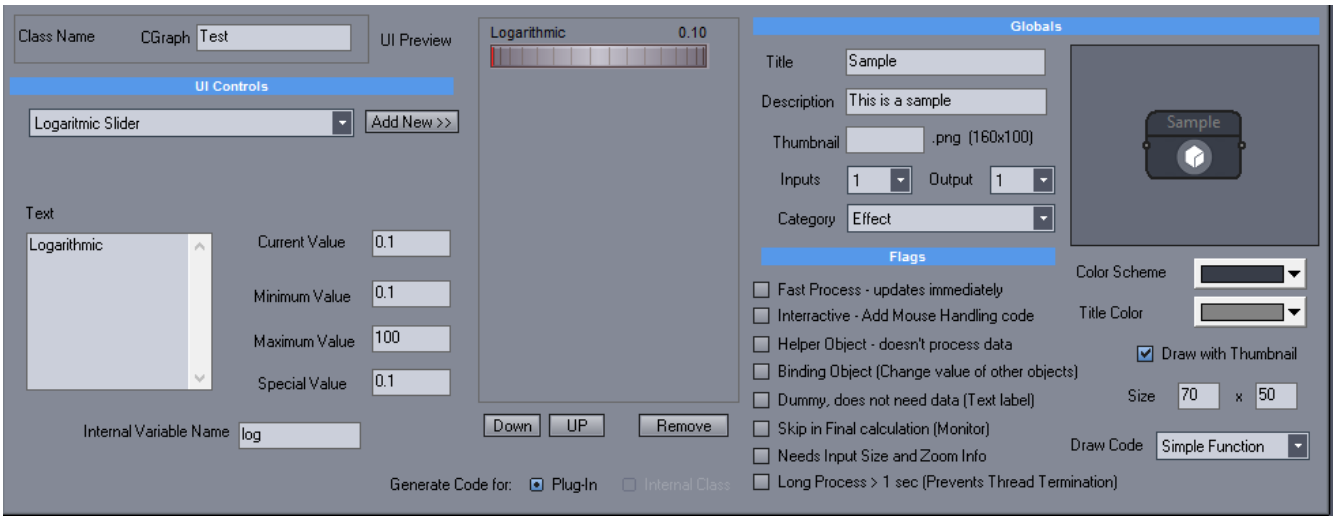




A graphical representation of the Gamma curve

Logarithmic Slider

Earlier we covered Exponential Sliders, the opposite to exponents is logarithm.



This control works almost exactly the same as a Slider.

This is used to set a user defined value to your routine. It could be used to set a radius or strength of an effect. Think of it as a volume control, with the minimum and maximum set by you the programmer.

To test, create a control with the current value as .01, minimum as .01, maximum as 100 and special as .01. Name the variable as log.

The control that it sets is

```
AddParameter( PARAM_LOG ,"Logarithmic Slider", 0.100, 0.100, 100.0, TYPE_LOGSLIDER, 0.100);
```

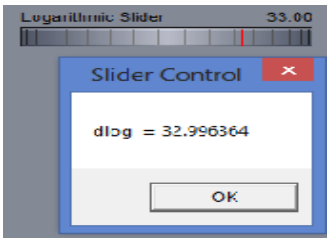
paste the following in the Virtual Void

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // Get the latest parameters

        //List of Parameters
        double dlog = GetValue(PARAM_LOG);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }

        char sBuffer1[100]; sprintf(sBuffer1,
        "dlog  = %f" ,  dlog
        );MessageBox(NULL,sBuffer1,"Slider Control", MB_OK);
}
```

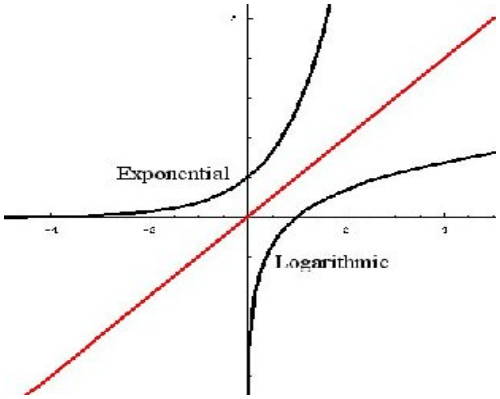
The output of the Logarithmic Slider is a double

Testing Logarithmic Control

Move your slider to 33

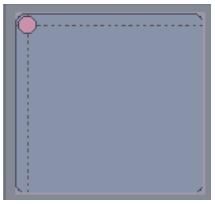


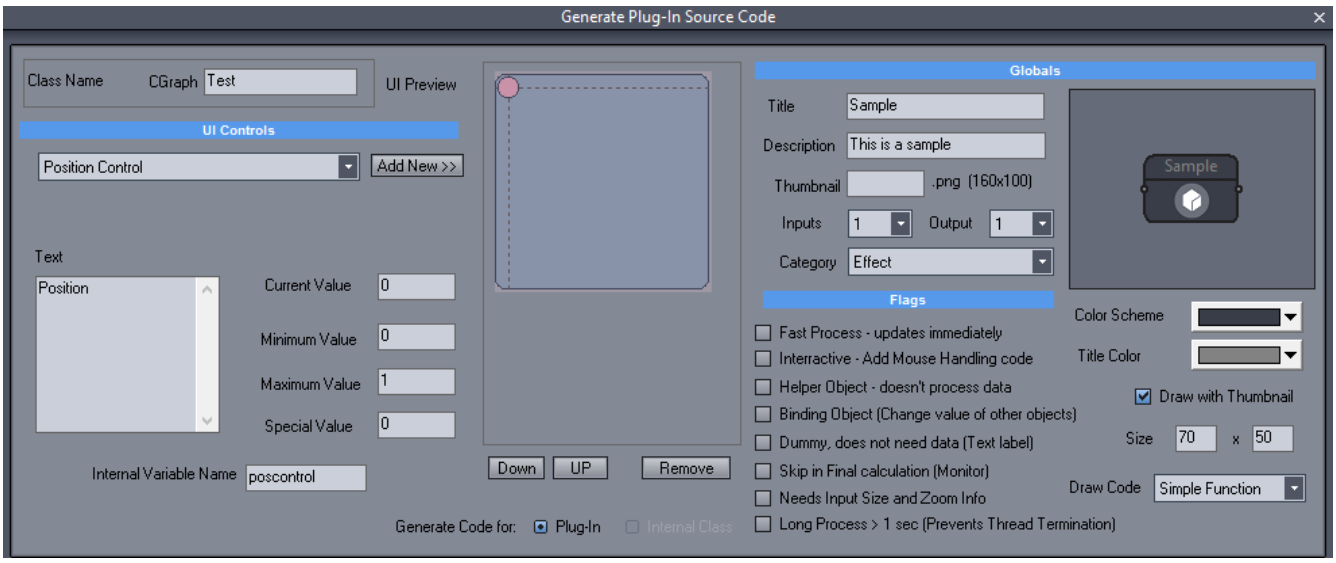
As you should realize by the name, the slider values grow Logarithmically.



A graphical representation of the Exponential and Logarithmic curves

Position Control





This control would be good for cropping or perhaps selection of one sort or another.

Create the control with Variable name poscontrol.

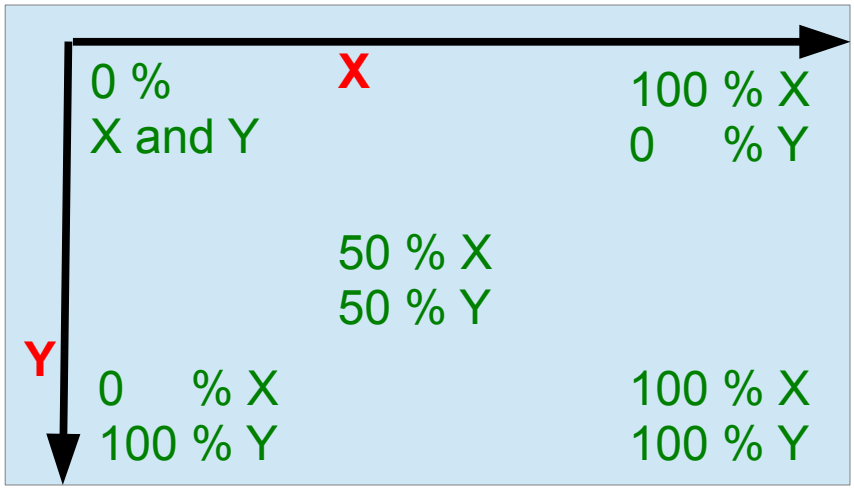
The control that is creates is

```
AddParameter( PARAM_POSCONTROL ,"Position", 0, 0, 1, TYPE_POSITION, 0);
```

The sample code for Position Controls is found in Media Changes SDK and it is more complex than the other controls found in Photo-Reactor. This will take a few pages to describe accurately.

First and foremost, this control will output a percentage of the image relative to where the cursor is. So for example, if you have your cursor at the top left X and Y, the percentage is X=0 and Y=0 because that is 0 percent of the Height and 0 percent of the width.

If you move your cursor all the way to the bottom, that affects the Y position. Then the Y would be at 100 percent. If you move your cursor all the way to the right, that affects the X position, Then the X would be at 100 percent.



So the Position control will output a percentage of the entire image which is retaliative to the position of the cursor in X and Y. How then do we convert that percentage to an actual pixel position, we will need to know the size of the image, not only the preview image (proxy image) but also the real image.

To do this we will need to adjust the `FLAG_NEEDSIZEDATA` to obtain the actual image Height and Width.

That is found in the routine

```
int GetFlags ()
{
        // it is fast process
        int nFlag = FLAG_NONE;

        return nFlag;
}
```

We need to find out some information on the image loaded, so we can place the position control.

```
        int GetFlags ()
        {
                // it is fast process
                int nFlag = FLAG_NONE;
                nFlag = nFlag | FLAG_NEEDSIZEDATA;

                return nFlag;
        }
```

```
nFlag = nFlag | FLAG_NEEDSIZEDATA;
```

Is a trigger inside Photo-Reactor to provide us the Height and Width of the complete image.

Now that the Flag/Trigger is set for Photo-Reactor to output the complete Height and Width, we will need to set some variables to be able to use. We will also add 2 variable to obtain the preview height and width.

Find the class and add some variables.

```
class PluginTest : public IPlugin
{
public:
```

Change it to

```
class PluginTest : public IPlugin
{
public:

int fullimagewidth;
int fullimageheight;
int previewwidth;
int previewheight;
```

Next we need to obtain the information from Photo-Reactor, find the routine

```
void SetSizeData(int nOriginalW, int nOriginalH, int nPreviewW, int nPreviewH, double dCropX1, double dCropY1,
double dCropX2, double dCropY2, double dZoom)
{
        // so if you need the position and zoom, this is the place to get it.
        // Note: because of IBM wisdom the internal bitmaps are on PC always upside down, but the coordinates are
        // not which you  need to take into account. See rectangle demo project for more info
}
```

and modify it to obtain the values from Photo-Reactor

```
void SetSizeData(int nOriginalW, int nOriginalH, int nPreviewW, int nPreviewH, double dCropX1, double dCropY1,
double dCropX2, double dCropY2, double dZoom)
{
        // so if you need the position and zoom, this is the place to get it.
        // Note: because of IBM wisdom the internal bitmaps are on PC always upside down, but the coordinates are
        // not which you  need to take into account. See rectangle demo project for more info
        fullimagewidth = nOriginalW;
        fullimageheight = nOriginalH;
        previewwidth = nPreviewW;
        previewheight = nPreviewH;
}
```

At this point we should have enough data to provide some actual coordinates to provide an output. Well almost, we will need a formula to convert the percentage that is sent by position control and the actual resolution of the image to obtain an actual X, Y coordinate.

That would be

output coordinate X = position coordinate X (which is a percentage) * full image width
and
output coordinate Y = position coordinate Y (which is a percentage) * full image height

Putting it all together

Find the class and under public 4 variables
fullimagewidth, fullimageheight, previewwidth, previewheight
so it looks like the below code block

```
class PluginTest : public IPlugin
{
public:

        int fullimagewidth;//added
        int fullimageheight;//added
        int previewwidth;//added
        int previewheight;//added
```

Find the GetFlags routine

```
int GetFlags ()
{
        // it is fast process
        int nFlag = FLAG_NONE;

        return nFlag;
}
```

and modify it like

```
int GetFlags ()
{
        // it is fast process
        int nFlag = FLAG_NONE;
        nFlag = nFlag | FLAG_NEEDSIZEDATA;//added

        return nFlag;
}
```

Your GetUIParameters should be

```
int GetUIParameters (UIParameters* pParameters)
{
        // label, value, min, max, type_of_control, special_value
        // use the UI builder in the software to generate this

        AddParameter( PARAM_POSITION_NR0 ,"Position", 0, 0, 100, TYPE_POSITION, 0);

        return NUMBER_OF_USER_PARAMS;
}
```

Your SetSizeData should be.

```
void SetSizeData(int nOriginalW, int nOriginalH, int nPreviewW, int nPreviewH, double dCropX1, double dCropY1,
double dCropX2, double dCropY2, double dZoom)
{
        // so if you need the position and zoom, this is the place to get it.
        // Note: because of IBM wisdom the internal bitmaps are on PC always upside down, but the coordinates are
        // not which you  need to take into account. See rectangle demo project for more info
        fullimagewidth = nOriginalW;
        fullimageheight = nOriginalH;
        previewwidth = nPreviewW;
        previewheight = nPreviewH;
}
```

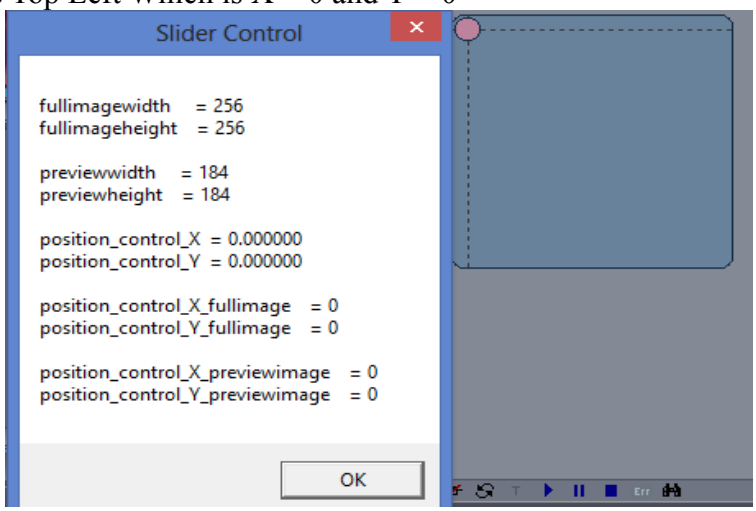
Your Virtual Void Routine should be

```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // Get the latest parameters

        //List of Parameters
        double dPosition = GetValue(PARAM_POSITION_NR0);

        double position_control_X = GetValue(PARAM_POSITION_NR0)/100.0;
        double position_control_Y = GetValueY(PARAM_POSITION_NR0)/100.0;

        int position_control_X_fullimage = int(position_control_X * fullimagewidth);
        int position_control_Y_fullimage = int(position_control_Y * fullimageheight);

        int position_control_X_previewimage = int(position_control_X * previewwidth);
        int position_control_Y_previewimage = int(position_control_Y * previewheight);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }

        char sBuffer1[500]; sprintf(sBuffer1,
        "fullimagewidth     = %d" "\n"
        "fullimageheight    = %d" "\n"
        "\n"
        "previewwidth     = %d" "\n"
        "previewheight    = %d" "\n"
        "\n"
        "position_control_X  = %f" "\n"
        "position_control_Y  = %f" "\n"
        "\n"
        "position_control_X_fullimage    = %d" "\n"
        "position_control_Y_fullimage    = %d" "\n"
        "\n"
        "position_control_X_previewimage    = %d" "\n"
        "position_control_Y_previewimage    = %d" "\n"
        ,
        fullimagewidth,
        fullimageheight,

        previewwidth,
        previewheight,

        position_control_X,
        position_control_Y,

        position_control_X_fullimage,
        position_control_Y_fullimage,

        position_control_X_previewimage,
        position_control_Y_previewimage

        );MessageBox(NULL,sBuffer1,"Slider Control", MB_OK);

}
```

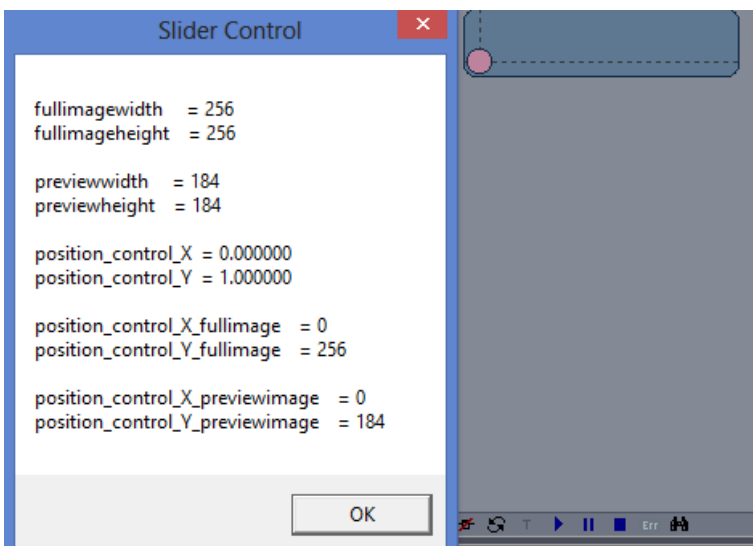
Testing the Position Control

The test image I loaded for this demo was 256 X and 256 Y

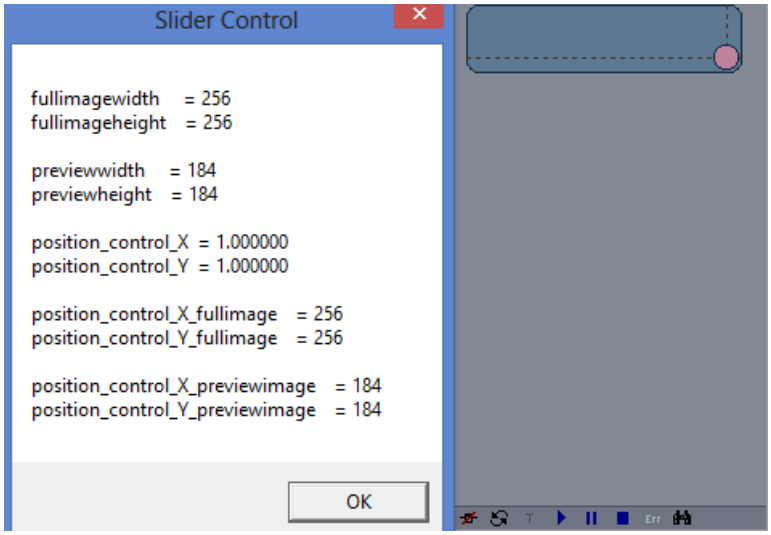
The position control is Top Left Which is X = 0 and Y = 0



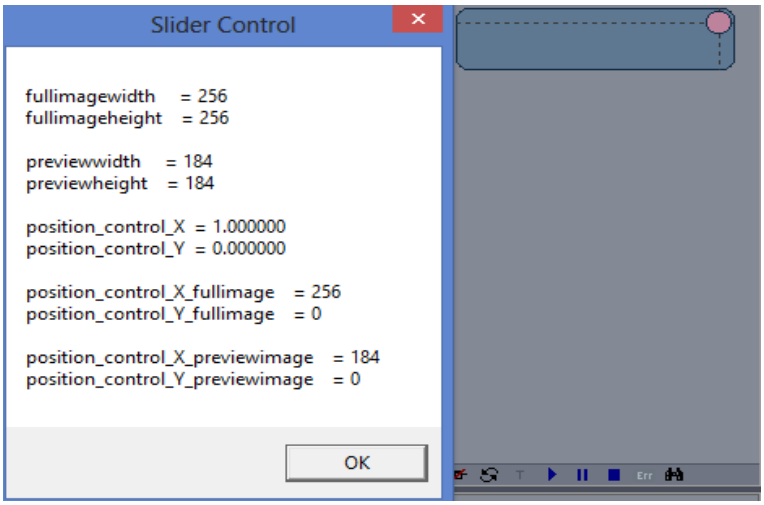
Lets pull the position control down to the bottom



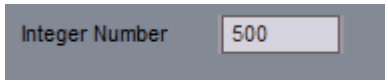
Now, with the position control down lets pull it all the way to the right.

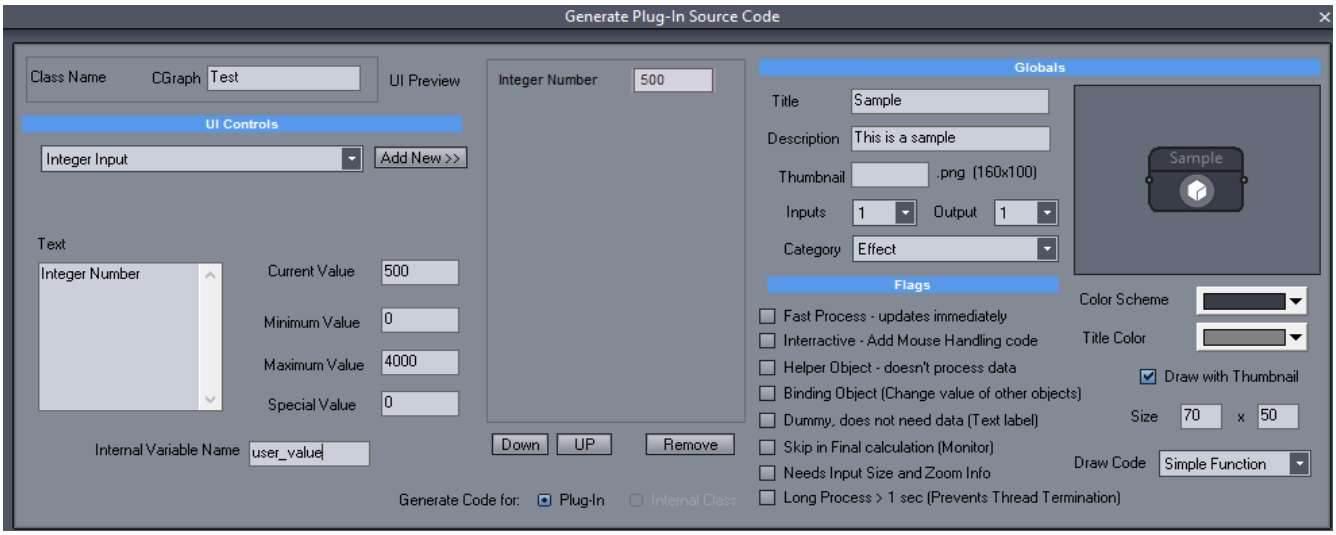


With the position control down and to the right, lets pull the control all the way up, while still to the right.



Integer Input





This control takes an end user integer input and sets a variable.

To test, create a control with the current value as 500, minimum as 0, maximum as 4000 and special as 0. Name the variable as user_value.

The control that it sets is

```
AddParameter( PARAM_INTEGER_NUMBER_NR0 ,"Integer Number", 500, 0, 4000, TYPE_INTEGER, 0);
```

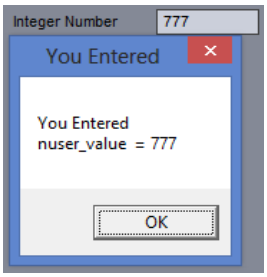
paste the following in the Virtual Void

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

        // Get the latest parameters

        //List of Parameters
        int nuser_value = (int)GetValue(PARAM_USER_VALUE);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }
        char sBuffer1[100]; sprintf(sBuffer1,
        "You Entered" "\n"
        "nuser_value  = %d" ,   nuser_value
        );MessageBox(NULL,sBuffer1,"You Entered", MB_OK);

}
```

The output of the Logarithmic Slider is an integer.

Testing Integer Input control

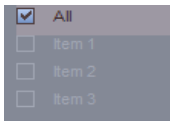
Enter in 777 in your Integer input



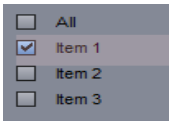
Checkbox Enable All / Until

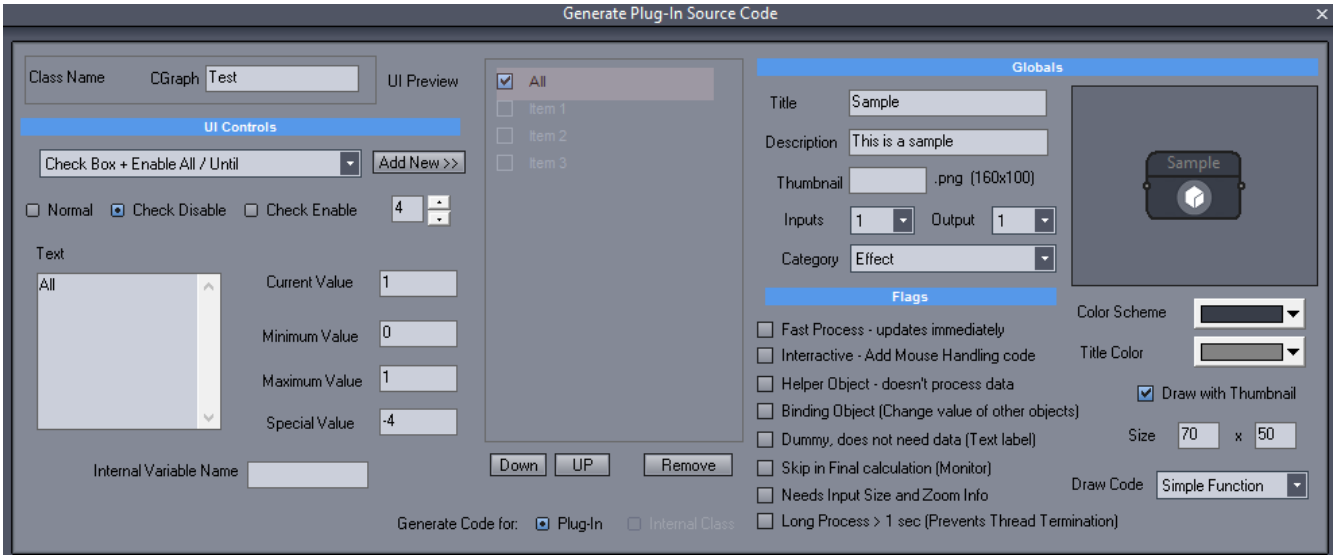
This is used to control a series of check boxes. It is used to set a multiple options, however only works if the primary first box is selected. This control is a little more complex and similar to Checkbox Enable all Following.

All checked on and other items checked off



All check is off, now you can make selections.



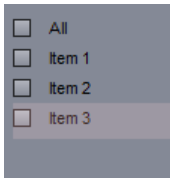


Here are the steps for creation of the Check Box + Enable All / Until.

For the purposes of this demo I will outline the steps involved.

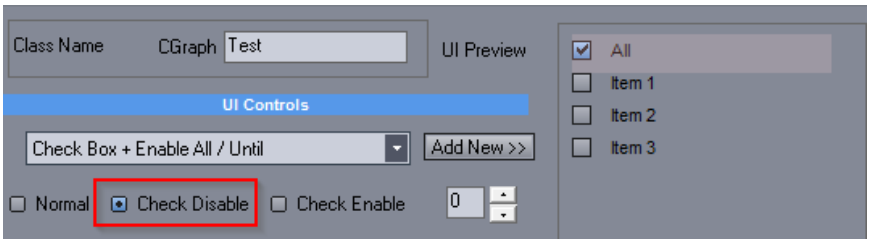
From a new Generate Plug-In Source code window
In UI Control pull down a Check Box + Enable All / Until
Select Add New
Name the control anything you want but for demo name it "ALL"
Do not set any values now
From the UI Control, Select Check Box and Add New
Name the control Item 1
Do not set any values now
From the UI Control, Select Check Box and Add New
Name the control Item 2
Do not set any values now
From the UI Control, Select Check Box and Add New
Name the control Item 3
Do not set any values now

The screen should look like



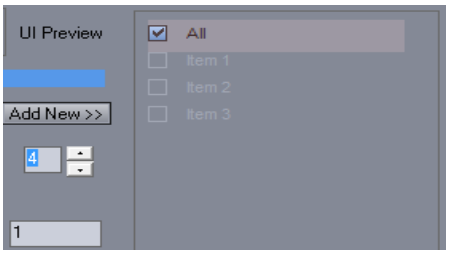
Now select the All Button

Select Check Disable



Select the down arrow button until Item 1,2 and 3 are disabled (greyed out)

Your Screen should look like this



Now lets name each variable

For the All, name the variable CBALL
For Item 1, name the variable item1
For Item 2, name the variable item2
For Item 3, name the variable item3

The controls that get set are

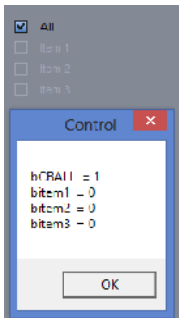
```
AddParameter( PARAM_CBALL ,"All", 1, 0, 1, TYPE_CHECKBOXDISABLENEXTUNTIL, -4);
AddParameter( PARAM_ITEM1 ,"Item 1", 0, 0, 1, TYPE_CHECKBOX, 0);
AddParameter( PARAM_ITEM2 ,"Item 2", 0, 0, 1, TYPE_CHECKBOX, 0);
AddParameter( PARAM_ITEM3 ,"Item 3", 0, 0, 1, TYPE_CHECKBOX, 0);
```

paste the following in the Virtual Void

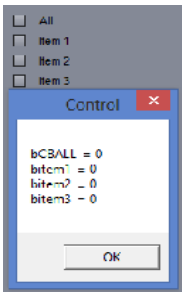
```
virtual void Process_Data (BYTE* pBGRA_out,BYTE* pBGRA_in, int nWidth, int nHeight, UIParameters* pParameters)
{
        // Get the latest parameters

        //List of Parameters
        BOOL bCBALL = GetBOOLValue(PARAM_CBALL);
        BOOL bitem1 = GetBOOLValue(PARAM_ITEM1);
        BOOL bitem2 = GetBOOLValue(PARAM_ITEM2);
        BOOL bitem3 = GetBOOLValue(PARAM_ITEM3);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }

        char sBuffer1[100]; sprintf(sBuffer1,
        "bCBALL  = %d" "\n"
        "bitem1  = %d" "\n"
        "bitem2  = %d" "\n"
        "bitem3  = %d" "\n"
        ,
        bCBALL,
        bitem1,
        bitem2,
        bitem3
        );MessageBox(NULL,sBuffer1,"Control", MB_OK);

}
```

Testing Checkbox Enable All / Until

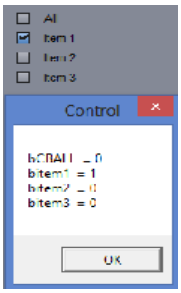
All Checked



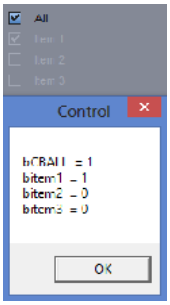
All UnChecked, everything else unchecked



All Checked, Item 1 checked and Item 2 and Item 3 unchecked.



All Unchecked, but Item 1 remains checked, Item 2 and Item 3 unchecked.



There are a couple of things that should be noticed. Simply un-checking the All Button, has no effect on the Bool for Item 1, 2 or 3. It simply means that the end user cannot change the control once the all button is unselected.

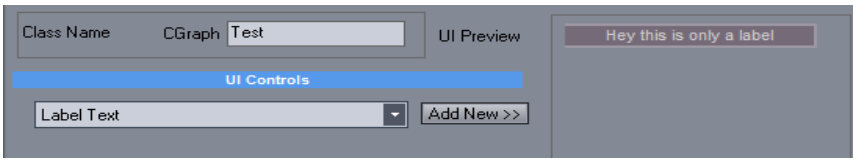
If you want to un-check the All button and then disable the following options, you will need to accommodate this in programming by using the AND operand.

Non Action Controls

A Non Action controls is a user interface (UI) control that enhances the user experience however does not effect the processing such a a label to label a section of Action controls or a separator that separates other action controls from one another.

Label Text

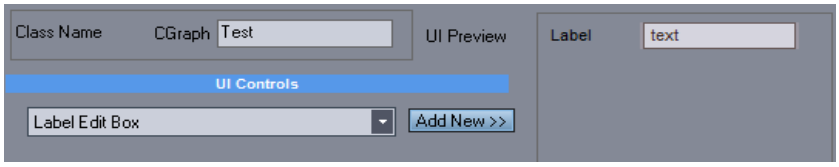
This is a label to Display above or below a control.



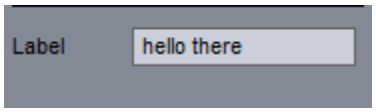
Useful to provide more details about a control.

Label Edit Box

This is a Label box that the end user can edit.

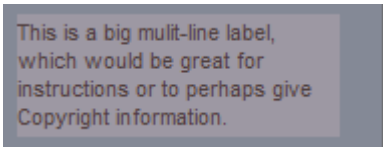


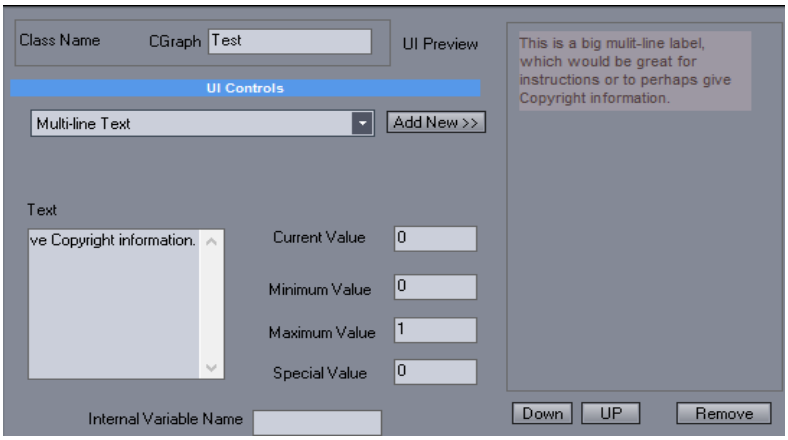
Useful for the end user to make notes about a control. This could also be used to output a string to an image. However, please note that Media Chance has stated that drawing text to an image buffer is **NOT** a trivial operation as there is no simple function that can do that in plain c++. For this reason, we are placing this as a non action control.



Multi-Line Text

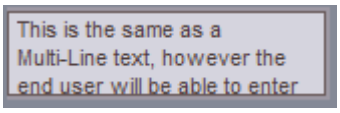
This is a label to Display long detailed information.





Multi-Line Edit Box

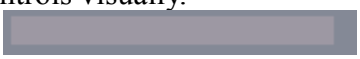
This is like a Label edit box and Mult-line text where the end user can make detailed notes.



Useful for the end user to make notes about a control. However this could also be used to place a string onto an image. However, please note that Media Chance has stated that drawing text to an image buffer is **NOT** a trivial operation as there is no simple function that can do that in plain c++. For this reason, we are placing this as a non action control.

Horizontal Space

This is a separator used to separate controls visually.



Undocumented Features for controls

There are certainly going to be a number of Undocumented features in any software, Photo-Reactor is no different. These feature may or may not disappear from future versions. Media Chance was generous to share these features. I am placing these here as I received from Media Chance, without any additional explanation.

There are also few undocumented values for other controls as the last parameter m_dSpecialValue
for TYPE_SLIDER
#define DEFVALUE_MINISLIDER 10
makes a thinner blue slider
#define DEFVALUE_INTSLIDER 1
force the slider to display integer values

example:
AddParameter("Edge", 0.0, 0.0, 50.0, TYPE_SLIDER, DEFVALUE_INTSLIDER);

TYPE_ONEOFMANY
#define DEFVALUE_ONOFMANYLIST 10
makes the radio buttons look like a list box
Ex:
AddParameter("Polarize Blue Light|Polarize Red Light|Polarize Green Light", 0, 0, 2, TYPE_ONEOFMANY, DEFVALUE_ONOFMANYLIST);

TYPE_EDIT
#define DEFVALUE_EDIT_NOLABEL 10
Will not draw the 'Label" text in front
AddParameter("", 0, 0, 1, TYPE_EDIT, DEFVALUE_EDIT_NOLABEL);

You would need to put those define in the plugin.cpp as they are not defined in sdk

Other tricks:
TYPE_PUSHBUTTON
ending label with * will make the button red
ending it with ~ will make it blue
ending it with & will make it blue and 2x height
AddParameter("My Button&", 0, 0, 1, TYPE_PUSHBUTTON, 0);

Filters with Two inputs
(When just One image won't do)

A common operation in Large Image Editors (like GIMP) is to blend 2 images together on a layer. Photo-Reactor has made with its node editing capacity, the need to use layers obsolete. However, the operation of blending two image on a layer is an important operation. Luckily in the SDK there is a rather easy way to perform this, a Multiple Input Operation.

Now before we get started on this filter creation, know that blending operations are not the only function that you will use multiple inputs for, however it is a common operation. This will introduce another type of math, one that we learned in middle school – Linear Algebra.

We are going to use several common blend operations in this test scenario, we will not duplicate all of them, just a few, however you can find the formulas to many if not all of the different blend formulas across the internet.

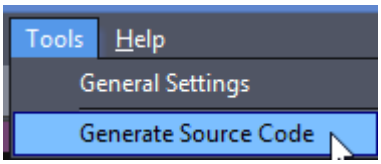
Here are the ones we will deal with

Lighten (Main Image,Secondary Image) = ((Secondary Image > Main Image) ? Secondary Image:Main Image)
Darken (Main Image,Secondary Image) = ((Secondary Image > Main Image) ? Main Image:Secondary Image)
Multiply (Main Image,Secondary Image) = ((Main Image * Secondary Image) / 255)
Average (Main Image,Secondary Image) = ((Main Image + Secondary Image) / 2)
Add (Main Image,Secondary Image) = min((Main Image + Secondary Image) , 255)
Subtract (Main Image,Secondary Image) = max((Main Image - Secondary Image) , 0)
Difference (Main Image,Secondary Image) = abs(Main Image - Secondary Image)
Negation (Main Image,Secondary Image) = 255 - abs(255 - Main Image - Secondary Image))
Screen (Main Image,Secondary Image) = (255 - (((255 - Main Image) * (255 - Secondary Image)) / 255))
Exclusion (Main Image,Secondary Image) = (Main Image + Secondary Image - 2 * Main Image * Secondary Image / 255)

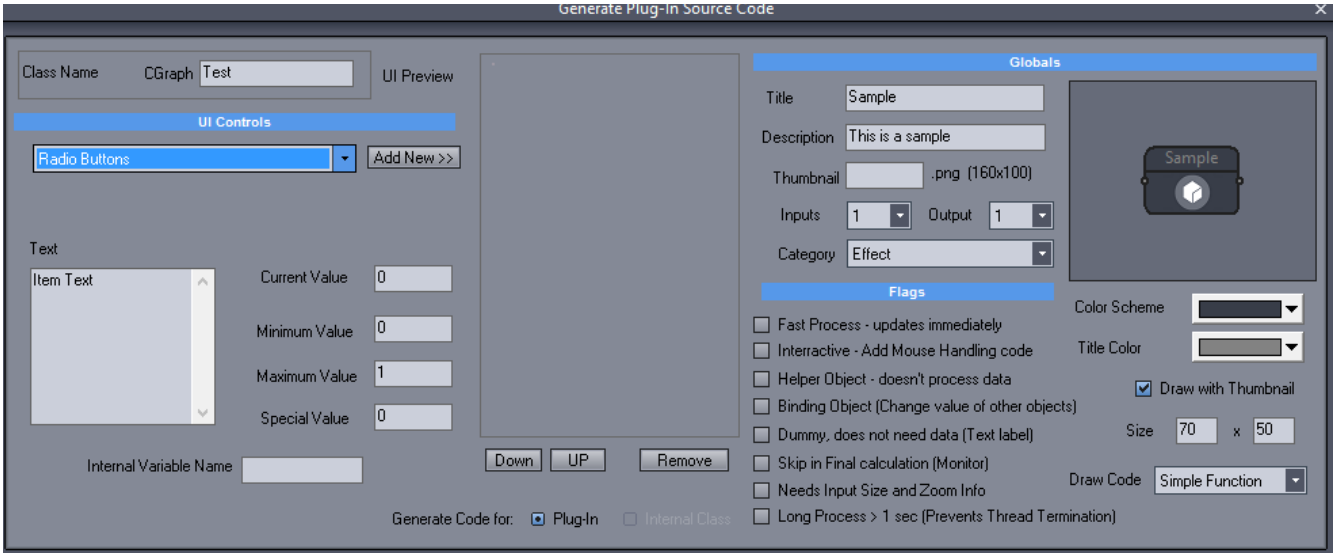
So now we have our formulas, what do we do with them? Well, first we need to create our filter.

We have 10 different formulas here, so we will need a radio buttons or combo box. I prefer radio buttons, so my sample will be using that.

Just like the other filters we need to start by generating to source code. Drill to Tools – Generate Source Code.

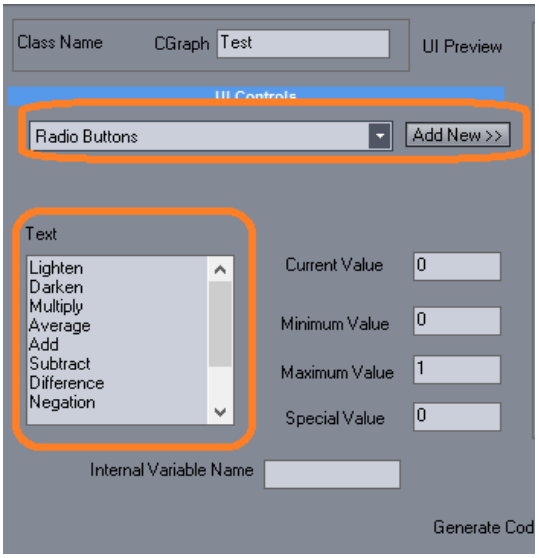


Here is the familiar screen

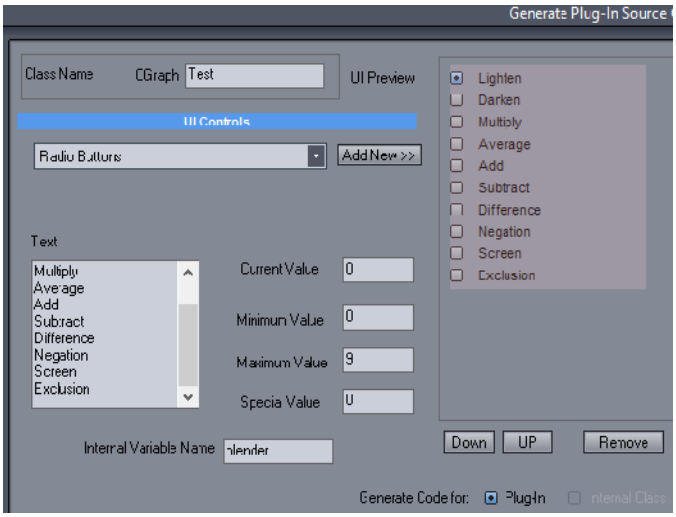


We will concentrate on the UI controls first, select Radio Buttons and select Add New. Paste the names in the text field.

Lighten
Darken
Multiply
Average
Add
Subtract
Difference
Negation
Screen
Exclusion

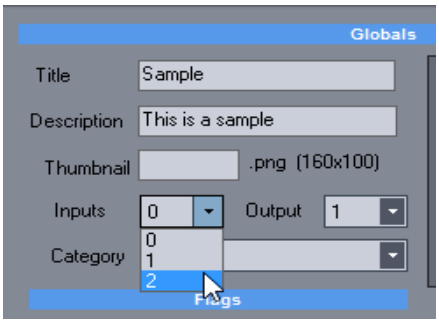


Set the Internal Variable name to Blender.



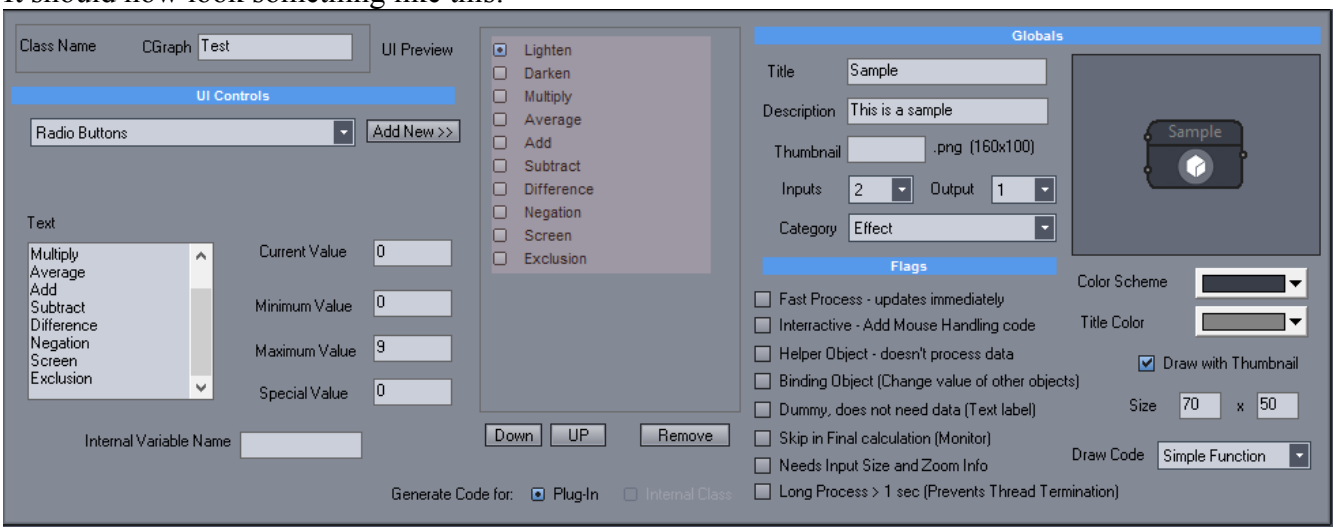
You screen should now look like this.

Now let's go to the Globals section.



Select 2 inputs

It should now look something like this:



Generate your code just like you did with your single input filter. Now let's open the filter in Visual Studio.

In our single input filters, we used the routine.

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

However, with the dual input filter, we need to work with the routine just below it

```
virtual void Process_Data2 (BYTE* pBGRA_out, BYTE* pBGRA_in1, BYTE* pBGRA_in2, int nWidth, int nHeight,
UIParameters* pParameters)
```

** One note, as of Beta 3, when you create a filter with 2 inputs, the x,y loop is not created, you can however copy it from the single input filter.

Just by casually looking at the overloads in the Process_Data2 routine, we can see 2 changes. pBGRA_in has changed to pBGRA_in1 and we can see there is a new overload pBGRA_in2. This will be for the Primary and Secondary images.

Just as with the single input filter, we will want to perform our include for math. Move to the top of the listing in Visual Studio

Find the comment

```
// plugin.cpp : Defines the entry point for the DLL application.
```

And place below the other includes

```
#include <math.h>
```

The entire section should look like

```
// plugin.cpp : Defines the entry point for the DLL application.
//
#include "stdafx.h"
#include "IPlugin.h"
#include <math.h>
//////////////////////////////////////////////////////////////////////
// A concrete plugin implementation
//////////////////////////////////////////////////////////////////////
```

Scroll downwards to find our virtual void Process_Data routine

Nothing is really needed in this routine, so I would like to copy and then delete everything in that particular routine.

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

Paste that same routine in the dual input routine Process_Data1

```
// actual processing function for 2 inputs
//*****************************************************************************
// all buffers are the same size
// don't change the IN buffers or things will go bad
// the pBGRA_out comes already with copied data from pBGRA_in1
virtual void Process_Data2 (BYTE* pBGRA_out, BYTE* pBGRA_in1, BYTE* pBGRA_in2, int nWidth, int nHeight,
UIParameters* pParameters)
{
        int nblender = (int)GetValue(PARAM_BLENDER);

        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 = CLAMP255((nR+nG+nB)/3);

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

Of course you will get a error because one of the main variables is not misnamed : `pBGRA_in`. Simply rename it to `pBGRA_in1`.

Now it should now look like

```
// actual processing function for 2 inputs
//*****************************************************************************
// all buffers are the same size
// don't change the IN buffers or things will go bad
// the pBGRA_out comes already with copied data from pBGRA_in1
virtual void Process_Data2 (BYTE* pBGRA_out, BYTE* pBGRA_in1, BYTE* pBGRA_in2, int nWidth, int nHeight,
UIParameters* pParameters)
{
        int nblender = (int)GetValue(PARAM_BLENDER);

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

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

                        int nR = pBGRA_in1[nIdx+CHANNEL_R];
                        int nG = pBGRA_in1[nIdx+CHANNEL_G];
                        int nB = pBGRA_in1[nIdx+CHANNEL_B];

                        int nA = CLAMP255((nR+nG+nB)/3);

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

Now we are back to the plug-in working like the default Photo-Reactor plug-in.

So now as referenced in the control section of this guide, we are going to need to work with the Radio Button controls. This is what the nblender variable is for. The variables are int (integer) ranging from 0 to 9.

We need to do some house cleaning in this routine to prepare for our code, lets cut out some unneeded code.

```
// actual processing function for 2 inputs
//*****************************************************************************
// all buffers are the same size
// don't change the IN buffers or things will go bad
// the pBGRA_out comes already with copied data from pBGRA_in1
virtual void Process_Data2 (BYTE* pBGRA_out, BYTE* pBGRA_in1, BYTE* pBGRA_in2, int nWidth, int nHeight,
UIParameters* pParameters)
{
        int nblender = (int)GetValue(PARAM_BLENDER);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;
                }
        }
}
```

Notice that I cut out just about everything inside the loop except out nIdx, we will need that for just about anything.

We will need to add the Radio Button variables possibilities, remember the range is anywhere between 0 and 9 integer. If we are going to add one, we might as well add them all.

While we are at it, we might also comment on what each routine will do.

See the code below on how this should happen.

```
// actual processing function for 2 inputs
//*****************************************************************************
// all buffers are the same size
// don't change the IN buffers or things will go bad
// the pBGRA_out comes already with copied data from pBGRA_in1
virtual void Process_Data2 (BYTE* pBGRA_out, BYTE* pBGRA_in1, BYTE* pBGRA_in2, int nWidth, int nHeight,
UIParameters* pParameters)
{
        //List of Parameters
        int nblender = (int)GetValue(PARAM_BLENDER);

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;

                        if (nblender == 0)//Lighten
                        {
                        }

                        if (nblender == 1)//Darken
                        {
                        }

                        if (nblender == 2)//Multiply
                        {
                        }

                        if (nblender == 3)//Average
                        {
                        }

                        if (nblender == 4)//Add
                        {
                        }

                        if (nblender == 5)//Subtract
                        {
                        }

                        if (nblender == 6)//Difference
                        {
                        }

                        if (nblender == 7)//Negation
                        {
                        }

                        if (nblender == 8)//Screen
                        {
                        }

                        if (nblender == 9)//Exclusion
                        {
                        }
                }
        }

}
```

As you can see all nblender variables are accounted for. Each looks to see if nblender is equal to any of the 10 different possibilities.

Now before we can start running with each of the formulas, we will need to declare some more variables. We will need to find the color of the primary image in each Red, Green, Blue, but also the color of the secondary image in each Red, Green and Blue. Finally, we will need variables for the output image in each Red, Green and Blue.

In your list of Parameters lets declare your variables, it should look something like.

```
//List of Parameters
int nblender = (int)GetValue(PARAM_BLENDER);

int primary_image_red;
int primary_image_green;
int primary_image_blue;

int secondary_image_red;
int secondary_image_green;
int secondary_image_blue;

int output_image_red;
int output_image_green;
int output_image_blue;
```

Now we need to define where these pixels are being obtained, from either the primary image or the secondary image.

Inside our X, Y loop, we need to gather the pixel information from each image. Underneath the nIdx variable let's do just that. It should look like.

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

primary_image_red = pBGRA_in1[nIdx+CHANNEL_R];
primary_image_green = pBGRA_in1[nIdx+CHANNEL_G];
primary_image_blue = pBGRA_in1[nIdx+CHANNEL_B];

secondary_image_red = pBGRA_in2[nIdx+CHANNEL_R];
secondary_image_green = pBGRA_in2[nIdx+CHANNEL_G];
secondary_image_blue = pBGRA_in2[nIdx+CHANNEL_B];
```

Now we should be ready to start.

For Routine 0, Lighten the formula was

```
Lighten = ((Secondary Image > Main Image) ? Secondary Image:Main Image)
```

This is saying that if the secondary image pixel is greater than the Main image pixel, then set the pixel to the secondary, otherwise set the pixel to the main image. Remember that we have to do this for each Red, Green and Blue.

It should look something like this

```
if (nblender == 0)//Lighten
{
        output_image_red   = ((secondary_image_red   > primary_image_red)   ? secondary_image_red   : primary_image_red);
        output_image_green = ((secondary_image_green > primary_image_green) ? secondary_image_green : primary_image_green);
        output_image_blue  = ((secondary_image_blue  > primary_image_blue)  ? secondary_image_blue  : primary_image_blue);
}
```

For Routine 1, Darken the formula was

```
Darken = ((Secondary Image > Main Image) ? Main Image:Secondary Image)
```

This is saying that if the secondary image pixel is greater than the Main image pixel, then set the pixel to the primary, otherwise set the pixel to the secondary. Remember that we have to do this for each Red, Green and Blue.

```
if (nblender == 1)//Darken
{
        output_image_red   = ((secondary_image_red   > primary_image_red)   ? primary_image_red   : secondary_image_red);
        output_image_green = ((secondary_image_green > primary_image_green) ? primary_image_green : secondary_image_green);
        output_image_blue  = ((secondary_image_blue  > primary_image_blue)  ? primary_image_blue  : secondary_image_blue);
}
```

1. For Routine 2, Multiply the formula was

```
Multiply = ((Main Image * Secondary Image) / 255)
```

Here's the code.

```
if (nblender == 2)//Multiply
{
        output_image_red =   ((primary_image_red   * secondary_image_red)/ 255);
        output_image_green = ((primary_image_green * secondary_image_green)/ 255);
        output_image_blue =  ((primary_image_blue * secondary_image_blue) / 255);
}
```

For Routine 3, Average the formula was

```
//Average = ((Main Image + Secondary Image) / 2)
```

```
if (nblender == 3)//Average
{
        output_image_red =   ((primary_image_red   + secondary_image_red)/ 2);
        output_image_green = ((primary_image_green + secondary_image_green)/ 2);
        output_image_blue =  ((primary_image_blue  + secondary_image_blue) / 2);
}
```

For Routine 4, Add the formula was

```
//Add =  MIN((Main Image + Secondary Image) , 255)
```

```
if (nblender == 4)//Add
{
        output_image_red   = min((primary_image_red   + secondary_image_red)  , 255);
        output_image_green = min((primary_image_green + secondary_image_green), 255);
        output_image_blue  = min((primary_image_blue  + secondary_image_blue) , 255);
}
```

For Routine 5, Subtract the formula was

```
//Subtract = MAX((Main Image - Secondary Image) , 0)
```

```
if (nblender == 5)//Subtract
{
        output_image_red   = max((primary_image_red   - secondary_image_red)  , 0);
        output_image_green = max((primary_image_green - secondary_image_green), 0);
        output_image_blue  = max((primary_image_blue  - secondary_image_blue) , 0);
}
```

For Routine 6, Difference the formula was

```
//Difference = ABS(Main Image - Secondary Image)

if (nblender == 6)//Difference
{
        output_image_red   = abs(primary_image_red   - secondary_image_red);
        output_image_green = abs(primary_image_green - secondary_image_green);
        output_image_blue  = abs(primary_image_blue  - secondary_image_blue);
}
```

For Routine 7, Negation the formula was

```
//Negation = 255 – ABS( 255 - Main Image - Secondary Image))

if (nblender == 7)//Negation
{
        output_image_red   = 255 - abs(255 - primary_image_red   - secondary_image_red);
        output_image_green = 255 - abs(255 - primary_image_green - secondary_image_green);
        output_image_blue  = 255 - abs(255 - primary_image_blue  - secondary_image_blue);
}
```

For Routine 8, Screen the formula was

```
//Screen = (255 - (((255 - Main Image) * (255 - Secondary Image)) / 255))

if (nblender == 8)//Screen
{
        output_image_red   = (255 - (((255 - primary_image_red)   * (255 - secondary_image_red))   / 255));
        output_image_green = (255 - (((255 - primary_image_green) * (255 - secondary_image_green)) / 255));
        output_image_blue  = (255 - (((255 - primary_image_blue)  * (255 - secondary_image_blue))  / 255));
}
```

For Routine 9, Exclusion the formula was

```
//Exclusion = (Main Image + Secondary Image - 2 * Main Image * Secondary Image / 255)

if (nblender == 9)//Exclusion
{
    output_image_red   = (primary_image_red   + secondary_image_red   - 2 * primary_image_red   * secondary_image_red   / 255);
    output_image_green = (primary_image_green + secondary_image_green - 2 * primary_image_green * secondary_image_green / 255);
    output_image_blue  = (primary_image_blue  + secondary_image_blue  - 2 * primary_image_blue  * secondary_image_blue  / 255);
}
```

There we have all of the formulas and code for the blend methods listed, there are many more out there.

Now we will want to clamp the values to make sure none go over the color space maximum (in this case 255) by using the Clamp255 macro. This will go below all of the blend methods.

```
output_image_red = CLAMP255(output_image_red);
output_image_green = CLAMP255(output_image_green);
output_image_blue = CLAMP255(output_image_blue);
```

Finally, we will need to output the data back to the screen. This happens with the pBGRA command, this will go below the Clamp macro.

```
pBGRA_out[nIdx+CHANNEL_R] = output_image_red;
pBGRA_out[nIdx+CHANNEL_G] = output_image_green;
pBGRA_out[nIdx+CHANNEL_B] = output_image_blue;
```

Your code listing found in `Process_Data2` Should look like:

---------- CODE BLOCK ----------

```
// actual processing function for 2 inputs
//*****************************************************************************
// all buffers are the same size
// don't change the IN buffers or things will go bad
// the pBGRA_out comes already with copied data from pBGRA_in1
virtual void Process_Data2 (BYTE* pBGRA_out, BYTE* pBGRA_in1, BYTE* pBGRA_in2, int nWidth, int nHeight, UIParameters*
pParameters)
{
	//List of Parameters
	int nblender = (int)GetValue(PARAM_BLENDER);

	int primary_image_red;
	int primary_image_green;
	int primary_image_blue;

	int secondary_image_red;
	int secondary_image_green;
	int secondary_image_blue;

	int output_image_red;
	int output_image_green;
	int output_image_blue;

	for (int y = 0; y< nHeight; y++)
	{
		for (int x = 0; x< nWidth; x++)
		{
			int nIdx = x*4+y*4*nWidth;

			primary_image_red = pBGRA_in1[nIdx+CHANNEL_R];
			primary_image_green = pBGRA_in1[nIdx+CHANNEL_G];
			primary_image_blue = pBGRA_in1[nIdx+CHANNEL_B];

			secondary_image_red = pBGRA_in2[nIdx+CHANNEL_R];
			secondary_image_green = pBGRA_in2[nIdx+CHANNEL_G];
			secondary_image_blue = pBGRA_in2[nIdx+CHANNEL_B];

			if (nblender == 0)//Lighten
			{
				output_image_red   = ((secondary_image_red   > primary_image_red)   ?
secondary_image_red   : primary_image_red);
				output_image_green = ((secondary_image_green > primary_image_green) ?
secondary_image_green : primary_image_green);
				output_image_blue  = ((secondary_image_blue  > primary_image_blue)  ?
secondary_image_blue  : primary_image_blue);
			}

			if (nblender == 1)//Darken
			{
				output_image_red   = ((secondary_image_red   > primary_image_red)   ? primary_image_red   :
secondary_image_red);
				output_image_green = ((secondary_image_green > primary_image_green) ? primary_image_green :
secondary_image_green);
				output_image_blue  = ((secondary_image_blue  > primary_image_blue)  ? primary_image_blue  :
secondary_image_blue);
			}

			if (nblender == 2)//Multiply
			{
				output_image_red   = ((primary_image_red   * secondary_image_red)  / 255);
				output_image_green = ((primary_image_green * secondary_image_green)/ 255);
				output_image_blue  = ((primary_image_blue  * secondary_image_blue) / 255);
			}

			if (nblender == 3)//Average
			{
				output_image_red =   ((primary_image_red   + secondary_image_red)/ 2);
				output_image_green = ((primary_image_green + secondary_image_green)/ 2);
				output_image_blue =  ((primary_image_blue  + secondary_image_blue) / 2);
			}

			if (nblender == 4)//Add
			{
				output_image_red   = min((primary_image_red   + secondary_image_red)  , 255);
				output_image_green = min((primary_image_green + secondary_image_green), 255);
				output_image_blue  = min((primary_image_blue  + secondary_image_blue) , 255);
			}
```

```
                        if (nblender == 5)//Subtract
                        {
                                output_image_red   = max((primary_image_red   - secondary_image_red)  , 0);
                                output_image_green = max((primary_image_green - secondary_image_green), 0);
                                output_image_blue  = max((primary_image_blue  - secondary_image_blue) , 0);
                        }

                        if (nblender == 6)//Difference
                        {
                                output_image_red   = abs(primary_image_red   - secondary_image_red);
                                output_image_green = abs(primary_image_green - secondary_image_green);
                                output_image_blue  = abs(primary_image_blue  - secondary_image_blue);
                        }

                        if (nblender == 7)//Negation
                        {
                                output_image_red   = 255 - abs(255 - primary_image_red   - secondary_image_red);
                                output_image_green = 255 - abs(255 - primary_image_green - secondary_image_green);
                                output_image_blue  = 255 - abs(255 - primary_image_blue  - secondary_image_blue);
                        }

                        if (nblender == 8)//Screen
                        {
                                output_image_red   = (255 - (((255 - primary_image_red)   * (255 - secondary_image_red))   /
255));
                                output_image_green = (255 - (((255 - primary_image_green) * (255 - secondary_image_green)) /
255));
                                output_image_blue  = (255 - (((255 - primary_image_blue)  * (255 - secondary_image_blue))  /
255));
                        }

                        if (nblender == 9)//Exclusion
                        {
                                output_image_red   = (primary_image_red   + secondary_image_red   - 2 * primary_image_red
* secondary_image_red   / 255);
                                output_image_green = (primary_image_green + secondary_image_green - 2 * primary_image_green
* secondary_image_green / 255);
                                output_image_blue  = (primary_image_blue  + secondary_image_blue  - 2 * primary_image_blue
* secondary_image_blue  / 255);
                        }

                        output_image_red = CLAMP255(output_image_red);
                        output_image_green = CLAMP255(output_image_green);
                        output_image_blue = CLAMP255(output_image_blue);

                        pBGRA_out[nIdx+CHANNEL_R] = output_image_red;
                        pBGRA_out[nIdx+CHANNEL_G] = output_image_green;
                        pBGRA_out[nIdx+CHANNEL_B] = output_image_blue;
                }
        }

}
```

---------- END CODE BLOCK ----------

Just like any other filter, we will need to give the filter a unique ID found in `GetPluginID` and give it a name found in `GetPluginName.`

And there we have it, a filter with 2 inputs, there are a good number of things this can be useful for: Blending of all sorts, gathering statics on one image and applying to another, image comparison, image masking, attribute copying, color copying, lighting transfer, etc.

Customizing your Icon

Photo-Reactor has built into it, an Icon customizing parameter. This feature is more advanced and thus more complicated to program, however it will allow you in software to have an interactive icon.

First, before we start delving deeply within the software and special parts of the SDK, lets cover some basics.

<u>Section on creation of Custom Icons</u>

Beyond the creation of the custom icon there are a number of other things we can do to change the appearance of the icon. First lets look at the code to change the title of the Icon.

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

This code block is the name of your icon, or aptly named, the Title of your icon. Again, the SDK places a valuable note above the code block. Keep the Title short and sweet.

The next code block of interest, is the color of the icon, that is defined in the block below

```
int GetBoxColor ()
{
	return RGB(56,61,72);
}
```

Next is its companion, the color of the text inside the icon.

```
int GetTextColor ()
{
	return RGB(130,130,130);
}
```

Both are defined in the RGB triplets. The Icon Box color in this case is a dark gray and the text is a light gray.

The next code section is for the Width of your icon, it is found in the following code block.

```
// width of the box in the workspace
// valid are between 50 and 100
int GetBoxWidth ()
{
	return 70;
}
```

That covers the basic Icon interface, however Photo-Reactor has more to offer from within the framework that it provides. The next section provides a more advanced aspect of Icon creation, Icon customization.

Before we delve into this section, we are assuming that you already know how to place in some code snippets, can compile and can move the file as well as test. If you are unsure, please read the basics first.

Icon Customization allows you to display additional information from within the Icon itself, such as text or highlighting colors or perhaps a new control, such as a radio style dial, there are many possibilities.

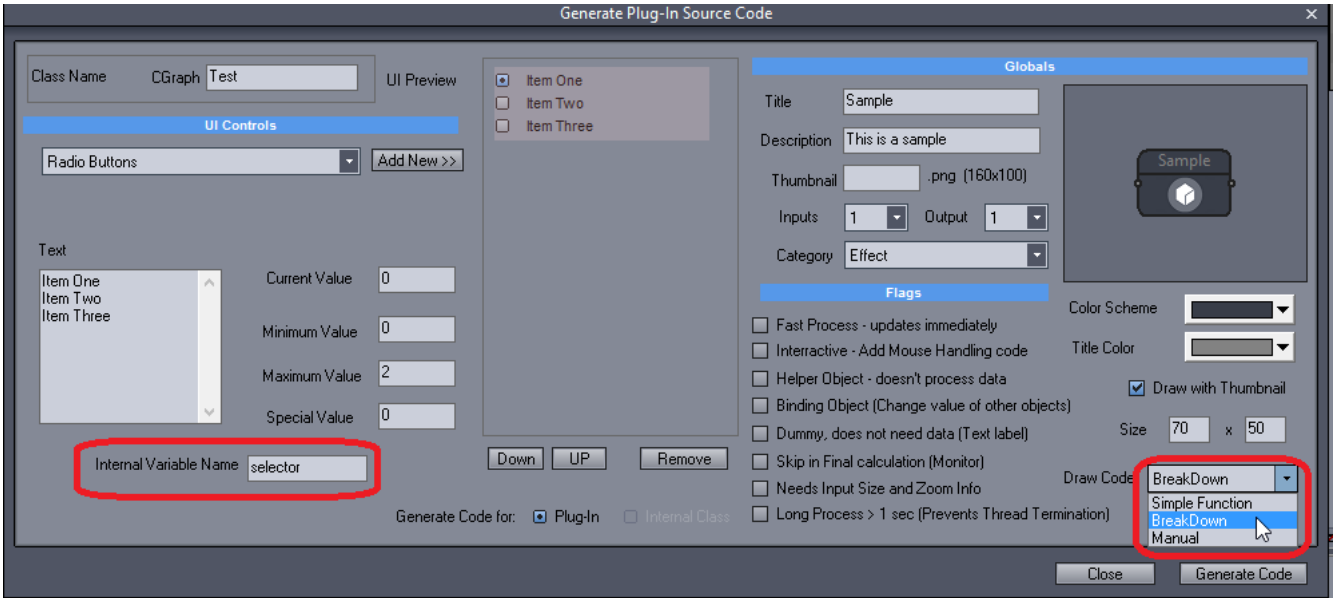
In this next part we are going to cover custom Icon design.

This is not the same as creating a PNG image for your icon.

We are going to start with drawing on the Icon.

First we are going to create a simple effect, I've chosen radio button, the effect control doesn't matter, we are just using it for later testing. The control can be anything.

On the far right bottom side, you will see “Draw Code”, there are three settings: Simple Function, Break Down and Manual.



The section of code we will need to pay attention to first is.

```
//*****************Drawing functions for the BOX *********************************
//how is the drawing handled
//DRAW_AUTOMATICALLY the main program will fully take care of this and draw a box, title, socket
and thumbnail
//DRAW_SIMPLE_A                  will draw a box, title and sockets and call CustomDraw
//DRAW_SIMPLE_B                  will draw a box and sockets and call CustomDraw
//DRAW_SOCKETSONLY      will call CustomDraw and then draw sockets on top of it

// highlighting rectangle around is always drawn except for DRAW_SOCKETSONLY

virtual int GetDrawingType ()
{
       int nType = DRAW_SIMPLE_B;
       return nType;
}
```

Simple Function, automatically draws everything, will draw a box, title, socket and thumbnail and does not call CustomDraw.

Break Down, has two type, Simple A, which draws most of what is seen on simple function and Simple B which draws the box and sockets and calls CustomDraw.

Manual, will call CustomDraw and then draw sockets on top of it and calls CustomDraw.

Below are the types of Icons.

Automatic Icon

Break Down Icon – Simple A

Break Down Icon – Simple B

Manual Icon

As you can see each level down from Automatic to Manual has fewer and fewer items drawn on them, leaving the design more and more dependent to the programmer. As you can see the difference between Simple A and Simple B is the name population and the line underneath the Name.

When you use Automatic or Break Down Icon – Simple A, the code for the name title is found in the code block

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

This of course is assuming that there is no overriding code to overwrite the title field within Break Down Icon – Simple A, which can be done.

In Break Down Icon – Simple B and Manual the title field has to be manually populated. However, this is not quite as bad as it may seem at first.

Let's start by understanding the flow a little bit. Which is simple. If the n_Type in `virtual int GetDrawingType ()` is something other than `DRAW_AUTOMATICALLY`, then look to `virtual void CustomDraw` to draw the icon. The flow is simple, however the code in CustomDraw can get as complicated as you would like.

Testing Tip

if you want your plugin to appear at the top of the plugin list, you will want to change the name in the object library to have a character in front, this should only be for tester code and not full release plugins.

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

Notice the ! (exclamation point) in front of the name, now your plugin will be at the top of the list as !Sample.

Output Text to an Icon

To output text to an Icon, we will first need to be on a draw style other than automatic. Drill to the drawing type section and change it to the following.

```
virtual int GetDrawingType ()
{
	int nType = DRAW_SIMPLE_A;
	return nType;
}
```

Next we will drill down to the next section of code CustomDraw, and change it to the following.

```
virtual void CustomDraw (HDC hDC, int nX,int nY, int nWidth, int nHeight, float scale, BOOL
bIsHighlighted, UIParameters* pParameters)
{
	TextOut (hDC,nX*scale + 20 * scale,nY*scale + 1 *scale,"hello",5);
}
```

Now let's compile the code and move it.

Of course if you look at the Title Bar you can see the obvious error. There is interference between us drawing on the image with text and Photo-Reactor doing the same thing. This is apparent on `DRAW_SIMPLE_A` only and not on any other drawing style.

Let's try that again, however we are going to set the drawing style to B

```
virtual int GetDrawingType ()
{
	int nType = DRAW_SIMPLE_B;

	return nType;
}
```

Again, we will compile and move.

That was the result we are looking for.

If we wanted to stay with `DRAW_SIMPLE_A` we need to make sure we are not going to interfere with the Title area. That can be done by moving our text down lower.

So, now we will again change the drawing style back to `DRAW_SIMPLE_A`

```
virtual int GetDrawingType ()
{
	int nType = DRAW_SIMPLE_A;

	return nType;
}
```

Again we will modify the `CustomDraw` section.

```
virtual void CustomDraw (HDC hDC, int nX,int nY, int nWidth, int nHeight, float scale, BOOL
bIsHighlighted, UIParameters* pParameters)
{
	TextOut (hDC , nX * scale + 20 * scale , nY * scale + 25 * scale , "hello" , 5);
}
```

You will notice the the nY area has changed slightly, it changed from

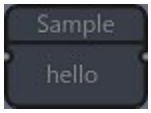
```
TextOut (hDC,nX*scale + 20 * scale,nY*scale + 1 *scale,"hello",5);
```

To

```
TextOut (hDC,nX*scale + 20 * scale,nY*scale + 25 *scale,"hello",5);
```

We are moving the text field down lower.

Compile and move your plugin.



That's much better. We are not placing text on top of other text.

Let's examine the TextOut function.

```
TextOut (hDC , nX * scale + 15 * scale , nY * scale + 25 * scale , "hello",5);
```

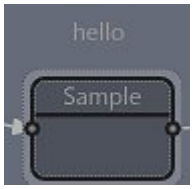
That can be better explained with the following

```
TextOut (hDC,Horizontal Position in relationship to Icon,Vertical Position in relationship to
Icon , String to output, length of string);
```

You will also see three elegant variable – *scale* , nX and nY – These tell Photo-Reactor, where to draw the text. For example, if we mess with scale using the following code.

```
TextOut (hDC , nX * scale + 20 * scale , nY * (scale / 2) + 25 * (scale / 2),"hello",5);
```

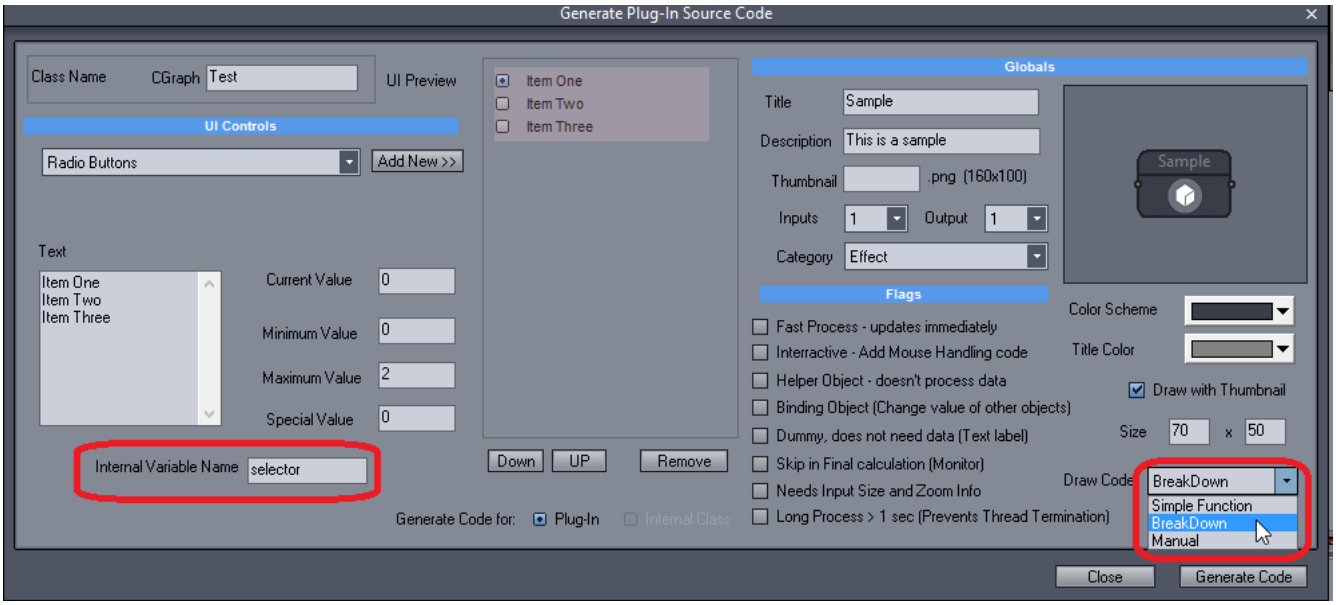
We will see the following output



Now we have the hello floating above the Icon.

Now let's make things a little more interesting

We can introduce a number of different variables to this mix. If you recall, when we created our code in Photo-Reactor, we used a radio button, with some generic names and called that variable selector.



We can use that variable selector to output different text to the Icon.

Try the next code block

```
virtual void CustomDraw (HDC hDC, int nX,int nY, int nWidth, int nHeight, float scale, BOOL
bIsHighlighted, UIParameters* pParameters)
{
	int nselector = (int)GetValue(PARAM_SELECTOR);

	if (nselector == 0)
	{
		TextOut (hDC , nX * scale + 20 * scale , nY * scale + 25 * scale ,"zebra",5);
	}

	if (nselector == 1)
	{
		TextOut (hDC , nX * scale + 20 * scale , nY * scale + 25 * scale ,"kitty",5);
	}

	if (nselector == 2)
	{
		TextOut (hDC , nX * scale + 20 * scale , nY * scale + 25 * scale ,"doggy",5);
	}
}
```

*For simplicity sake, the three text strings are the exact number of characters, if you had more or less, you would have to chage the text string length as well to match the number of characters, it is the 5 at the end of the code line. You would also have to change the + 20 on the nX * scale section to adjust the vertical allignment.*

Now again, we compile and move the plugin.

Select Item 1
our output.

Select Item 2
our output

Select Item 3
our output

Now, let's make things a little more colorful for our text.

Consider the next snippet of code.

```
virtual void CustomDraw (HDC hDC, int nX,int nY, int nWidth, int nHeight, float scale, BOOL
bIsHighlighted, UIParameters* pParameters)
{
	int nselector = (int)GetValue(PARAM_SELECTOR);

	if (nselector == 0)
	{
		SetTextColor (hDC, RGB(0, 0, 255));
		TextOut (hDC , nX * scale + 20 * scale , nY * scale + 25 * scale ,"zebra",5);
	}

	if (nselector == 1)
	{
		SetTextColor (hDC, RGB(0, 255, 0));
		TextOut (hDC , nX * scale + 20 * scale , nY * scale + 25 * scale ,"kitty",5);
	}

	if (nselector == 2)
	{
		SetTextColor (hDC, RGB(255, 0, 0));
		TextOut (hDC , nX * scale + 20 * scale , nY * scale + 25 * scale ,"doggy",5);
	}
}
```

As you can see with the above code, each of the text output codes are preceded with a SetTextColor, in RGB format, the first (zebra) is format will be blue, followed by green and finally red. Let's test the routine.

Colors make things a little more friendly and easier on the eyes.

Now, the above code works just fine, however, we want to include a different feature, a color block, to make things easier for the user. This is easily accomplished by setting the background color. The command is SetBkColor. This is dependant on another command, SetBkMode

If we were to set the background to green we would use. `SetBkColor(hDC,RGB(0,255,0));`

Consider the next block of code.

```
virtual void CustomDraw (HDC hDC, int nX,int nY, int nWidth, int nHeight, float scale, BOOL
bIsHighlighted, UIParameters* pParameters)
{
	int nselector = (int)GetValue(PARAM_SELECTOR);
	SetBkMode(hDC,OPAQUE);// allows background highlighting of words

	if (nselector == 0)
	{
		SetTextColor(hDC, RGB(255, 0, 0));//sets the color of the text
		SetBkColor(hDC,RGB(0,255,0));//sets the background of the text
		TextOut (hDC , nX * scale + 20 * scale , nY * scale + 25 * scale ,"hello",5);
		//output text
	}

	if (nselector == 1)
	{
		SetTextColor(hDC, RGB(0, 255, 0));//sets the color of the text
		SetBkColor(hDC,RGB(255,0,0));//sets the background of the text
		TextOut (hDC , nX * scale + 20 * scale , nY * scale + 25 * scale ,"kitty",5);
		//output text
	}

	if (nselector == 2)
	{
		SetTextColor(hDC, RGB(0, 255, 255));//sets the color of the text
		SetBkColor(hDC,RGB(0,0,255));//sets the background of the text
		TextOut (hDC , nX * scale + 20 * scale , nY * scale + 25 * scale ,"doggy",5);
		//output text
	}
}
```

Provides the output

That's all there was to that. There are however many things that can be performed within creating your own Icon, many more time complex than what we performed here. It is important to remember the KISS (or keep it simple and stupid), don't try to create something more complex than needed, otherwise you will spend more time creating the Icon, than it takes creating the filter itself.

Tips for Microsoft Visual Studio Express

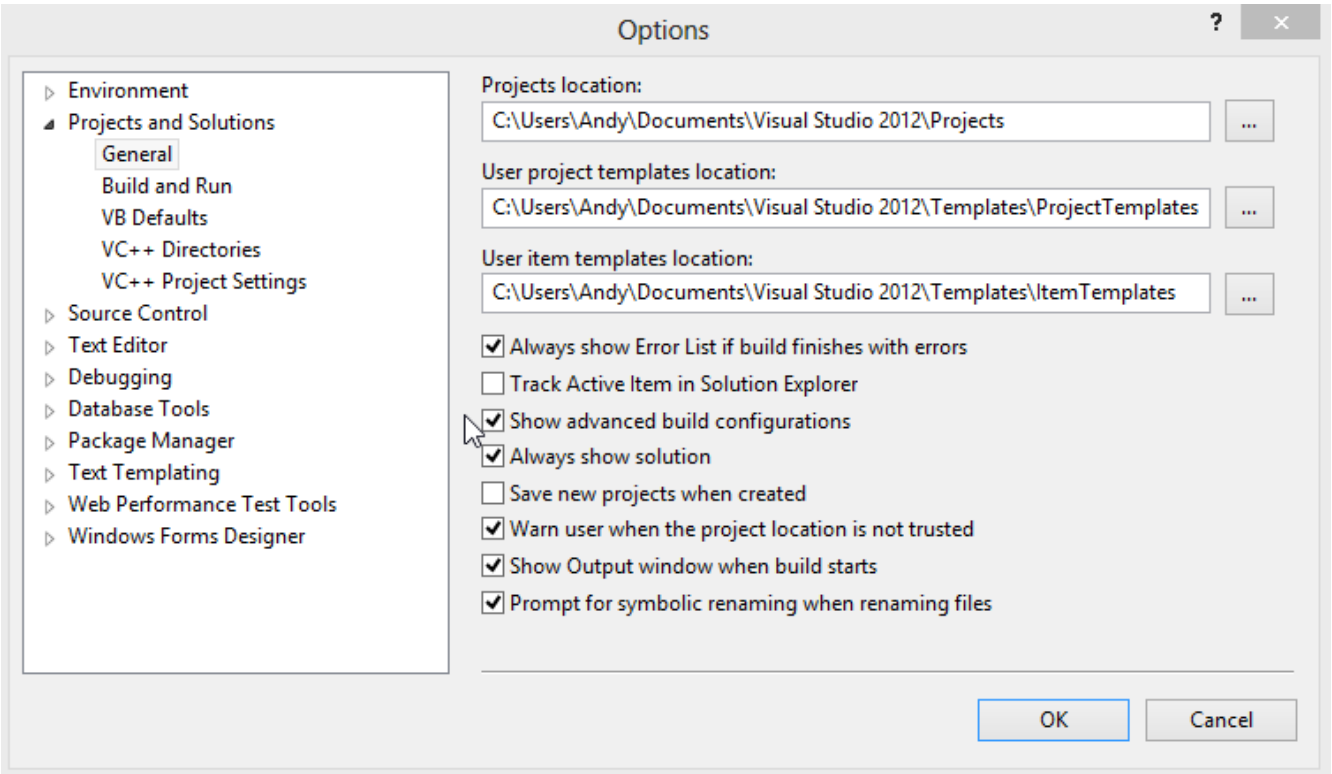
Here are some tip for getting Microsoft Visual Studio working well for your development environment. These Tips are for Visual Studio 2012, but should also apply to other versions as well.

Compiling in 64 Bit

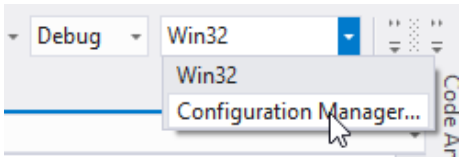
Windows has had 64 bit version of its operating since Windows 2000, though it really never started getting popular since Windows Vista when the prices of 64 Bit processors started getting economical. 64 Bit processing has a number of advantages with the most advertised being more memory access.

Visual Studio Express does not by default compile in 64 bit mode, however it is not difficult to accomplish.

Drill to Tools – Option – Projects and Solutions – General and select the Show Advanced Build Configurations.

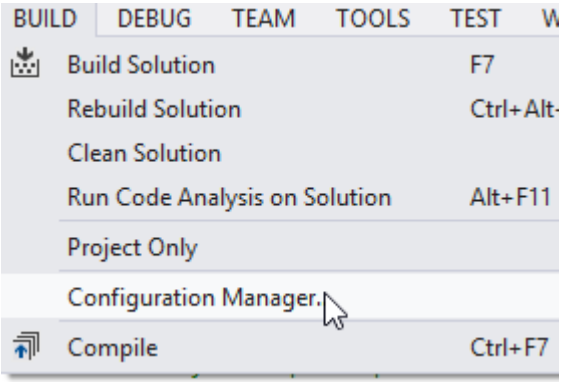


With your project open.

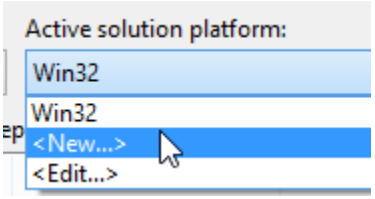


Where you see Win32 (processor selector), click on the down arrow.
Select Configuration Manager

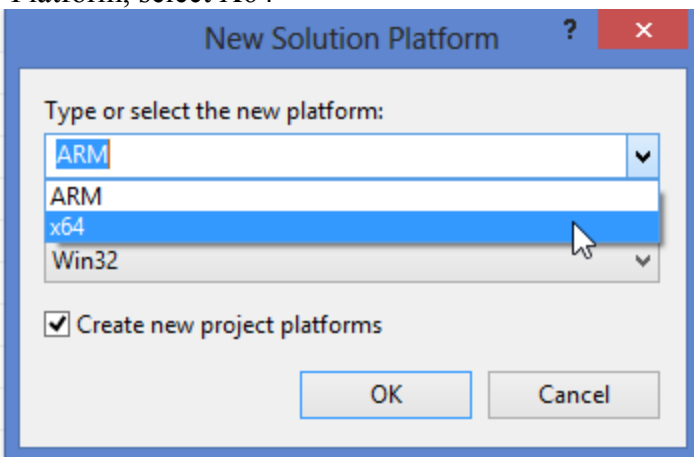
Alternately, drill to Build – Configuration Manager



On the Active Solution Platform, select New



In the Type of select new Platform, select X64



Copy setting from Win32, press OK

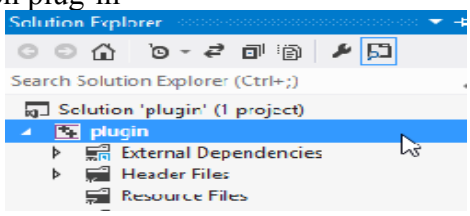
Line numbers

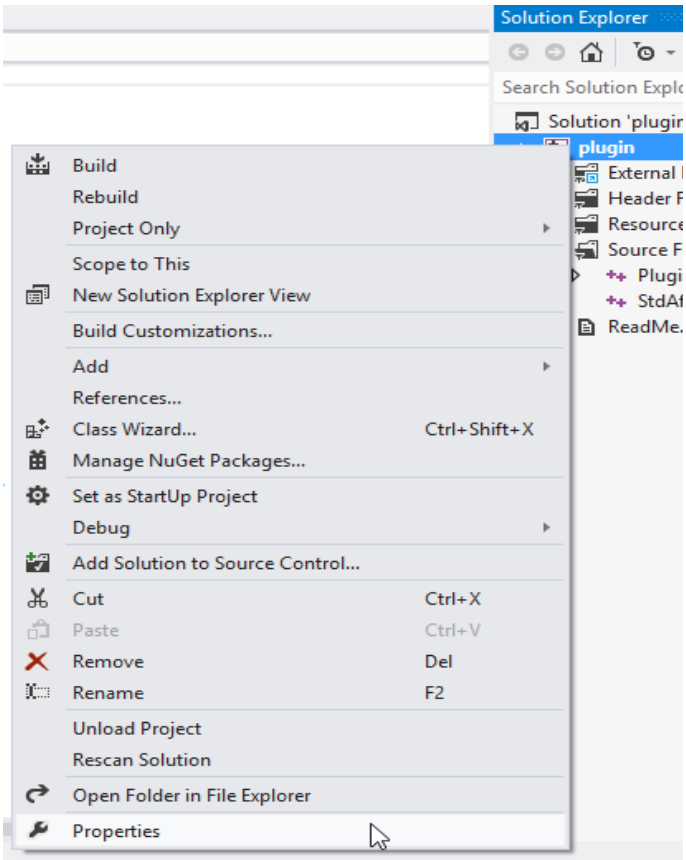
It is sometime nice to have line numbers in your program to keep track of where you are

Drill to Tools – Options – Text Editor – C/C++ – General. Under Display you just check Line Numbers

Change the name of the output

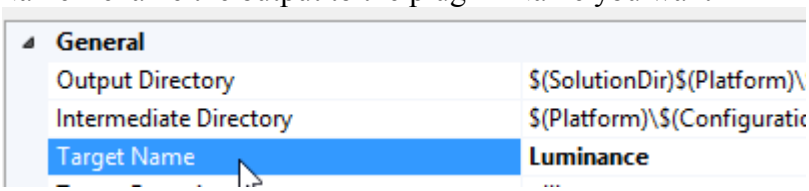
In Solution Explorer, right click on plug-in



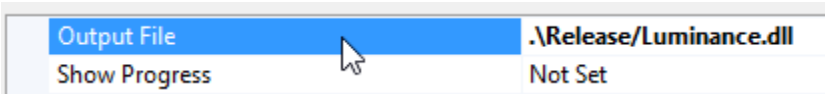


Select Configuration Properties

In General – Target Name Rename the output to the plug-in Name you want



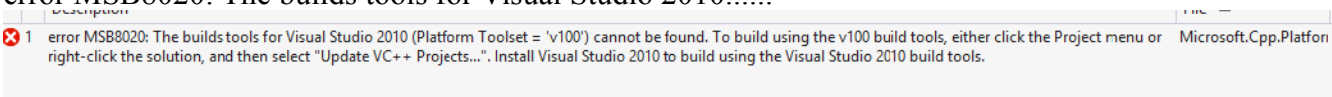
Drill to Linker – General – Output File and rename your plug-in to the name you want with a .dll extension.



Converting from one Visual Studio version to another

There are a number of different errors that can occur during the conversion process and this document is not aimed to help with everything, however if I find a problem, I will post it. Remember google is your friend, however some other resources are stackoverflow.com and microsoft visual studio forums.

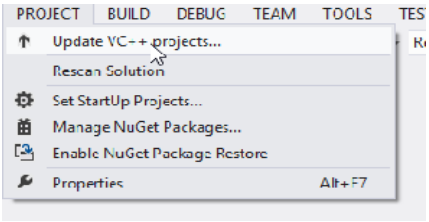
error MSB8020: The builds tools for Visual Studio 2010.....



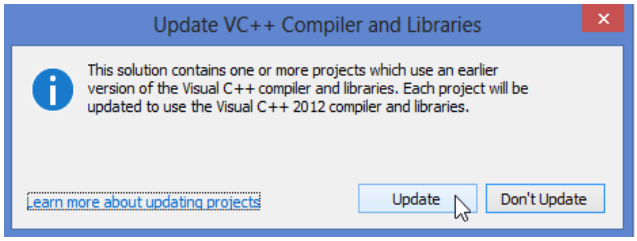
The error is fully explained if you copy the error and paste to notepad.

error MSB8020: The builds tools for Visual Studio 2010 (Platform Toolset = 'v100') cannot be found. To build using the v100 build tools, either click the Project menu or right-click the solution, and then select "Update VC++ Projects..."

This means that the project was built in one version of Visual Studio and needs conversion to be built in another version. This happens if you are going from VS2008 → 2010 → 2012



In VS 2012, go to Project – Update VC++ projects



Select Update. A few seconds later, your project should be ready to compile.

Tips for Programming Photo-Reactor Plug-ins

These Tips are in no particular order, nor are they guaranteed to be of any particular use for everything.

Working in 0-1 floating point as opposed to Integer 0-255.

A vast majority of Algorithms available online are programmed in 0-1 space with floating points, however there are enough programmed in the 0-255 space that one needs to pay careful attention to the algorithm. Placing the image in the 0-1 space required that the variable be either a double or floating point.

How do you place and image in 0-1 space from 0-255 space.

Easy way

```
float temp_red   = (float) nR / 255.0;
float temp_green = (float) nG / 255.0;
float temp_blue  = (float) nB / 255.0;
```

However, I prefer to do it slightly differently these days

```
float colorspace = 255.0;

float temp_red = (nR / colorspace);
float temp_green = (nG / colorspace);
float temp_blue = (nB /colorspace);
```

What is really the difference between the two? They both get the job done the same way.

The difference is easy expandability. Currently this program supports only 8 bit RGB images, however in the future there could be support for 16 bit images. Separating the divisor 255 allows you to easily place an If statement to accommodate for 16 bit images such as

```
if (bitdepth == 8)
{
        float colorspace = 255.0;
}
else
{
        float colorspace = 32768.0;
}
float temp_red   = (nR / colorspace);
float temp_green = (nG / colorspace);
float temp_blue  = (nB / colorspace);
```

Once you are in the 0-1 space, eventually you will need to come back to the 0-255 color space. We just reverse the formula.

```
Int Red   = temp_red   * colorspace;
Int Green = temp_green * colorspace;
Int Blue  = temp_blue  * colorspace;

pBGRA_out[nIdx+CHANNEL_R] = Red;
pBGRA_out[nIdx+CHANNEL_G] = Green;
pBGRA_out[nIdx+CHANNEL_B] = Blue;
```

Clamping Values

After programming a test filter, I received a note from Media Chance:

I would normally throw CLAMP255 on the line Lum = luma * colorspace; like Lum = CLAMP255(luma...) just for good measure. We are talking about unsigned char buffers and over or under means the result color gets huge visible jump. Not saying it is the case here but with the rounding and compiler optimization a max number can easily jump from 255 to 256

This is an extremely good point and could be a easy to create bug for a plug-in by not adhering to this. Luckily, built into the SDK is a easy to use Macro that does just that.

```
CLAMP255
```

Usage is simple

```
output = CLAMP255(input)
```

Using our sample from above

```
Int Red   = CLAMP255(temp_red   * colorspace);
Int Green = CLAMP255(temp_green * colorspace);
Int Blue  = CLAMP255(temp_blue  * colorspace);
```

This will assure that no value will exceed 255, if it tries, it will be clamped down to 255.

Another Case for 0-1 space

Any time you need to gather statistics on an image, it usually starts with counting cumulative sum or perhaps gathering an ever increasing number. If you are doing a small 640x480 image, this will probably not get out of hand, however these days we deal with images that have millions of pixels.

In C++ (and other languages as well), we have variables that have a finite bit length and thus a finite range that they can be. So for example

int = –2,147,483,648 to 2,147,483,647
uint = 0 to 4,294,967,295
short int = -32768 to 32767
float = 3.4E +/- 38 (7 digits)
double = 1.7E +/- 308 (15 digits)
bool = 0 to 1

As you can see, floating point and double can grow rather large, but not matter how large the number is, most likely it will be exceeded at some point.

So lets, go back to the 640x480 image, small image, however what if all the pixels were white. The sum of all the pixels would be

(640*480) * 255 = 78,336,000
That a rather large value for such a small image

lets try some other image sizes
(1024 * 768) * 255 = 200,540,160
2 Mega Pixel (1600 * 1200) * 255 = 489,600,000
4 Mega Pixel (2272 *1704) * 255 = 987,229,440
10 Mega Pixel (3648 * 2736) * 255 = 2,545,136,640

As you can see, image sizes can really grow fairly fast and thus the sum can also grow fast. Now in reality, no one will have a 10 Mega Pixel pure white image (unless you are into that sort of thing). None the less, it make more sense from a software point of view to reduce the image into the 0-1 space, perform your calculations and then resize the numbers as you are calculating the pixel value itself.

Double are more accurate than floats, but also take more memory to declare, and you may not notice a difference anyway.

Forward and Backward compatibility of the Plug-in SDK

The plugin SDK use C++ polymorphism and so there is very little reason the base will need to change as new things can be added with backward and forward compatibility. Basically it mimics COM functionality without the burden on the developer to actually mess with COM or any of the COM registration problems.

Memory Usage of Photo-Reactor Plug-ins

Declaring variables inside the SDK are fine if you declare them statically.

I.e.
float m_fVariable
or
float m_fVariable[255]

If you declare a variable dynamically, you will need to delete the variable
I.e.
float* pVar = new float[255]

If you use alloc or memalloc, you will need to delete or free the variable

Tips for Image Processing

**This is a great place for others to contribute, please message your formulas to me andydansby on the Mediachance boards or post codes/formulas to the mediachance message-boards and I will post the formula here along with the posters name. If you are sending a code snippet, please label your variables with a full name and comment as much as possible. Try to work in the 0-1 space rather than 0-255.

It is often best, but not always necessary to include math functions in your source code, inclusion is easy, simply place:

```
#include <math.h>
```

at the very top of your code beside the other includes

Here are some code Snippets for various image processing calculations. Make sure you have

```
#include <math.h>
```

at the very top beside the other includes

In your Virtual Void routine for single input plug-ins use

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

For dual input plug-ins use

```
virtual void Process_Data2 (BYTE* pBGRA_out, BYTE* pBGRA_in1, BYTE* pBGRA_in2, int nWidth, int nHeight,
UIParameters* pParameters)
```

You can run multiple loops to determine statistics of your image

I.e

```
for (int y = 0; y< nHeight; y++)
{
	for (int x = 0; x< nWidth; x++)
	{
		int nIdx = x*4+y*4*nWidth;
		//determine statistics
	}
}

for (int y = 0; y< nHeight; y++)
{
	for (int x = 0; x< nWidth; x++)
	{
		int nIdx = x*4+y*4*nWidth;
		//perform action
	}
}
```

To normalize your image from 0-255 to 0-1 floating point and back again

```
{
	float fcolorspace = 255.0;
	float fred = 0.0;
	float fgreen = 0.0;
	float fblue = 0.0;

	int red = 0;
	int green = 0;
	int blue = 0;

	for (int y = 0; y< nHeight; y++)
	{
		for (int x = 0; x< nWidth; x++)
		{
			int nIdx = x*4+y*4*nWidth;

			//normalize image to 0 - 1 space
			{
				fred = fred + pBGRA_in[nIdx+CHANNEL_R] / fcolorspace;
				fgreen = fgreen + pBGRA_in[nIdx+CHANNEL_G] / fcolorspace;
				fblue = fblue +  pBGRA_in[nIdx+CHANNEL_B] / fcolorspace;
			}

			//normalize image to 0 - 255 space
			{
				red = (fred * fcolorspace);
				green = (fgreen * fcolorspace);
				blue = (fblue * fcolorspace);
			}
		}
	}
}
```

To find the brightest pixel in an image in each of the red, green and blue channels (while working in floating point).

```
{
        float fcolorspace = 255.0;
        float fred = 0.0;
        float fgreen = 0.0;
        float fblue = 0.0;

        float maxred = 0.0;
        float maxgreen = 0.0;
        float maxblue = 0.0;

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;

                        //normalize image to 0 - 1 space
                        {
                                fred = fred + pBGRA_in[nIdx+CHANNEL_R] / fcolorspace;
                                fgreen = fgreen + pBGRA_in[nIdx+CHANNEL_G] / fcolorspace;
                                fblue = fblue +  pBGRA_in[nIdx+CHANNEL_B] / fcolorspace;
                        }

                        //to find max brightness in each color
                        {
                                if (fred > maxred) { maxred = fred; }
                                if (fgreen > maxgreen) { maxgreen = fgreen; }
                                if (fblue > maxblue) { maxblue = fblue; }
                                // brightest is found in maxred, maxgreen, maxblue
                        }
                }
        }
}
```

To find the darkest pixel in an image in each of the red, green and blue channels (while working in floating point).

```
{
        float fcolorspace = 255.0;
        float fred = 0.0;
        float fgreen = 0.0;
        float fblue = 0.0;

        float minred = 2.0;
        float mingreen = 2.0;
        float minblue = 2.0;

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;

                        //normalize image to 0 - 1 space
                        {
                                fred = fred + pBGRA_in[nIdx+CHANNEL_R] / fcolorspace;
                                fgreen = fgreen + pBGRA_in[nIdx+CHANNEL_G] / fcolorspace;
                                fblue = fblue +  pBGRA_in[nIdx+CHANNEL_B] / fcolorspace;
                        }

                        //to find min brightness in each color
                        {
                                if (fred < maxred) { minred = fred; }
                                if (fgreen < maxgreen) { mingreen = fgreen; }
                                if (fblue < maxblue) { minblue = fblue; }
                        }
                }
        }
}
```

To find the average pixel level in an image in each of the red, green and blue channels (while working in floating point).

```
{
        float fcolorspace = 255.0;
        float fred = 0.0;
        float fgreen = 0.0;
        float fblue = 0.0;

        float redsum = 0.0;
        float greensum = 0.0;
        float bluesum = 0.0;

        float redcount = 0.0;
        float greencount = 0.0;
        float bluecount = 0.0;

        float normalizedred = 0.0;
        float normalizedgreen = 0.0;
        float normalizedblue = 0.0;

        float pixelcount = 0.0;

        int averagered = 0.0;
        int averagegreen = 0.0;
        int averageblue = 0.0;

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;

                        //normalize image to 0 - 1 space
                        {
                                fred = fred + pBGRA_in[nIdx+CHANNEL_R] / fcolorspace;
                                fgreen = fgreen + pBGRA_in[nIdx+CHANNEL_G] / fcolorspace;
                                fblue = fblue +  pBGRA_in[nIdx+CHANNEL_B] / fcolorspace;
                        }

                        {
                                redsum += fred;
                                greensum +=  fgreen;
                                bluesum = fblue;
                        }
                        pixelcount ++;
                }
        }
        // at this point redsum,greensum and bluesum are the summation of all the pixel levels of the entire image
        // pixelcount are the number of pixels in the image.

        normalizedred = red/pixelcount;
        normalizedgreen = green/pixelcount;
        normalizedblue = blue/pixelcount;

        averagered = (normalizedred * fcolorspace);
        averagegreen = (normalizedgreen * fcolorspace);
        averageblue = (normalizedblue * fcolorspace);

        // final output
        // normalizedred, normalizedgreen and normalizedblue are the average pixel level of the entire image in floating point 0-1
        //averagered, averagegreen and averageblue  are the average pixel level of the entire image in interger 0 - 255
}
```

To find the Standard Deviation in an image in each of the red, green and blue channels (while working in floating point).

```
{
        float fcolorspace = 255.0;
        float fred = 0.0;
        float fgreen = 0.0;
        float fblue = 0.0;

        float redsum = 0.0;
        float greensum = 0.0;
        float bluesum = 0.0;

        float redcount = 0.0;
        float greencount = 0.0;
        float bluecount = 0.0;

        float normalizedred = 0.0;
        float normalizedgreen = 0.0;
        float normalizedblue = 0.0;

        float pixelcount = 0.0;

        int averagered = 0.0;
        int averagegreen = 0.0;
        int averageblue = 0.0;

        //standard deviation
        float sred;
        float sgreen;
        float sblue;

        float sdevred;
        float sdevgreen;
        float sdevblue;

        int stddevR;
        int stddevG;
        int stddevB;

        for (int y = 0; y< nHeight; y++)
        {
                for (int x = 0; x< nWidth; x++)
                {
                        int nIdx = x*4+y*4*nWidth;

                        //normalize image to 0 - 1 space
                        {
                                fred = fred + pBGRA_in[nIdx+CHANNEL_R] / fcolorspace;
                                fgreen = fgreen + pBGRA_in[nIdx+CHANNEL_G] / fcolorspace;
                                fblue = fblue +  pBGRA_in[nIdx+CHANNEL_B] / fcolorspace;
                        }

                        {
                                redsum += fred;
                                greensum +=  fgreen;
                                bluesum = fblue;
                        }

                        pixelcount ++;
                }
        }
```

```
	// at this point redsum,greensum and bluesum are the summation of all the pixel levels of the entire image
	// pixelcount are the number of pixels in the image.

	normalizedred = red/pixelcount;
	normalizedgreen = green/pixelcount;
	normalizedblue = blue/pixelcount;

	averagered = (normalizedred * fcolorspace);
	averagegreen = (normalizedgreen * fcolorspace);
	averageblue = (normalizedblue * fcolorspace);

	// final output
	// normalizedred, normalizedgreen and normalizedblue are the average pixel level of the entire image in floating point 0-1
	//averagered, averagegreen and averageblue  are the average pixel level of the entire image in interger 0 - 255

	for (int y = 0; y< nHeight; y++)
	{
		for (int x = 0; x< nWidth; x++)
		{
			int nIdx = x*4+y*4*nWidth;

			//normalize to 0 - 1
			{
				fred = pBGRA_in[nIdx+CHANNEL_R] / 255.0;
				fgreen = pBGRA_in[nIdx+CHANNEL_G] / 255.0;
				fblue = pBGRA_in[nIdx+CHANNEL_B] / 255.0;
			}

			sred += (red - normalizedred) * (red - normalizedred);
			sgreen += (green - normalizedgreen) * (green - normalizedgreen);
			sblue += (blue - normalizedblue) * (blue - normalizedblue);

		}
	}

	sred /= (nHeight * nWidth);
	sgreen /= (nHeight * nWidth);
	sblue /= (nHeight * nWidth);

	sdevred = sqrt(sred);
	sdevgreen = sqrt(sgreen);
	sdevblue = sqrt(sblue);

	stddevR = sdevred * fcolorspace;
	stddevG = sdevgreen * fcolorspace;
	stddevB = sdevblue * fcolorspace;

	// at this point sdevred, sdevgreen and sdevblue are the standard deviation in floating point 0-1 of the entire image
	// stddevR, stddevG and  are the standard deviation in in interger 0 - 255 of the entire image
}
```

Formulas

Placing your image in the 0-1 color space.

For each color you will want a floating point variable i.e.

```
float red;
float green;
float blue;
```

You will also need to know the color space of the image, i.e., 0-255 (or 8 bpp (bits per pixel)) or 0-65535 (16bpp).

formula for normalizing to 0-1 color space

floatingcolor = input / colorspace

Placing your image in the viewable color space

normalizedcolor = input * colorspace

Blending Mode Formulas

Sources www.pegtop.net , http://illusions.hu/ , http://www.wikipedia.org/

These formulas assume the 0-1 space. Top is the upper image or blend layer. Bottom is the lower image or base layer
On certain formulas you will see + .0001, this is to help against a divide by 0 error. Highlighted areas are math functions requiring the math.h header.

Normal	=	Top
Darken	=	min(Top,Bottom)
Lighten	=	max(Top,Bottom)
Addition	=	min((Top + Bottom), 1)
Subtract	=	max((Top – Bottom),0)
Multiply	=	(Top * Bottom) / 1
Divide	=	1 * Top / (Bottom + .00001);
Difference	=	abs(Top - Bottom)
Exclusion	=	0.5 - 2*(Top-0.5)*(Bottom-0.5)
Alternate Exclusion	=	Top + Bottom – (Top * Bottom / .5)
Negation	=	1 – abs(1-Bottom - Top)
Screen	=	(((1 - Bottom) * (1 - Top)) / 1);
Texture Illusions.hu	=	max(min(Bottom+Top)-0.5),1),0)
Parallel Illusions.hu	=	min(max(2/(1/Top+1/Bottom)),0,1)
Average	=	(Top + Bottom) / 2
Geometric Mean	=	sqrt(Top * Bottom)
Overlay	=	if Bottom < .5 (1 - 2 * (1 - Bottom) * (1 - Top) / 1) else (2 * Bottom * Top / 1)
Hard Light	=	if Bottom < .5 (2 * Bottom * Top / 1) else (1 - 2 * (1 - Bottom) * (1 - Top) / 1)

Soft Light	=	if (Top > 0.5)
		(1 - (1 - Bottom) * (1 - (Top - .5)))
		else
		(Bottom * (Top + .5))

Photoshop soft light	=	if (Top < .5)
		(2 * Bottom * Top + Bottom * Bottom * (1 - 2 * Top)):
		else
		sqrt(Bottom) * (2 * Top - 1) + (2 * Bottom) * (1 - Top)

Alternate Soft Light Pegtop	=	(1 - 2 * Bottom) * pow(Top,2) + 2 * Bottom * Top
Alternate Soft Light Illusions.hu	=	pow(Bottom,pow(2.0f,(2.0f*(0.f,.5f-Top))))
Andys Softlight	=	(((1 - Top) * ((Top * Bottom) / 1)) + (Top * (1 - ((1 - Top) * (1 - Bottom)) / 1))) / 1
Color Dodge	=	Bottom / (1 - Top)
Linear Dodge	=	Bottom + Top
Soft Dodge	=	((Top + Bottom) < 1) .5 * Bottom / (1 - Top) else 1 - .5 * (1 - Top) / Bottom
Color Burn	=	1 - (1 - Bottom) / Top
Linear Burn	=	Bottom + Top - 1

Soft Burn	=	if ((Top + Bottom) < 1)
		.5 * Top / (1 - Bottom)
		else
		1 - .5 * (1 - Bottom) / Top

Reflect	=	(Bottom * Bottom) / (1 - Top) + .00001
Glow	=	(Top * Top) / (1 - Bottom) + .00001
Freeze	=	1 - sqrt(1 - Bottom) / Top + .00001
Heat	=	1 - sqrt(1 - Top) / Bottom + .00001
Stamp	=	Bottom + 2 * Top - 1
interpolation	=	.5 - .25 * cos(3.14 * Bottom) - .25 * cos(3.14 * Top)

Vivid Light	=	if (Top > 0.5)
		1 - (1 - Bottom) / Top
		else
		Bottom / (1 - Top)

Linear Light	=	if (Top > 0.5)
		(Bottom + 2 * (Top - .5))
		else
		(Bottom + 2 * Top - 1)

Pin Light	=	if (Top > 0.5)
		(max(Bottom , 2 * (Top - .5)))
		else
		(min(Bottom , 2 * Top))

Grain Extract	=	Top - Bottom + .5
Grain Merge	=	Top + Bottom - .5;

Opacity Formula
Assuming a mixing value of 0 to 100 % and color space = 0-1
output = Bottomimage * MixingValue / 1.0 + Topimage * (1.0 - MixingValue) / 1.0;

Apply SRGB Gamma curve
srgb image = pow(image,2.2)

Remove SRGB Gamma curve
image = pow(srgb image,(1.0/2.2));

Extract Weighted Luminance
```
luma = (0.299 * red) + (0.587 * green) + (0.114 * blue);
```

Histogram Stretching (for contrast)
First find the minimum and maximum pixel values of the entire image. Store each in a separate variable. Then go through each pixel value again.

```
Histogram Stretch = (currentpixelvalue - minimumpixelvalue) * 1 / (maximumpixelvalue - minimumpixelvalue) +
.0001;
```

(the .0001 is to prevent divide by 0 error)