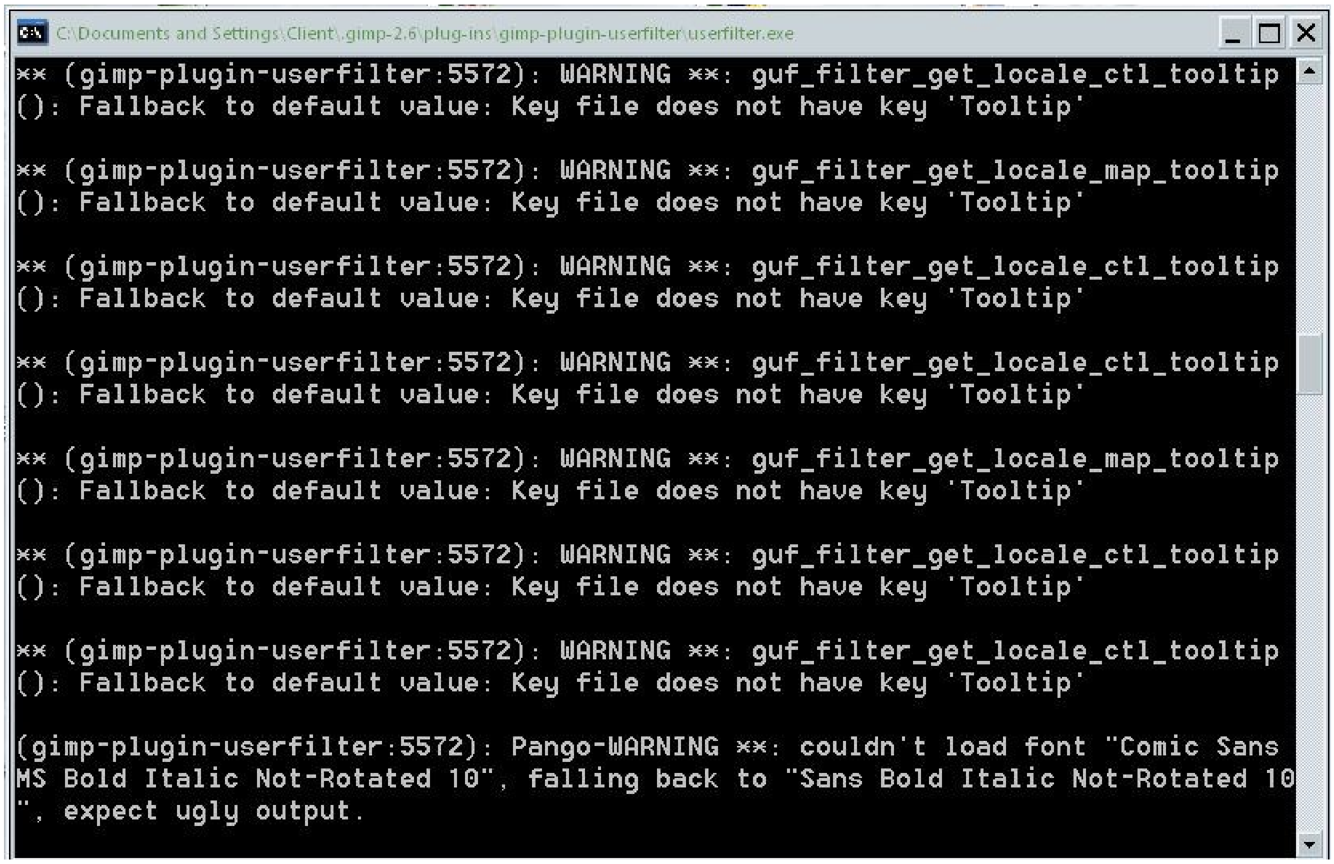
How To Use

BASIC

First do **no** worry for the first thing tou will see when starting the plugin:

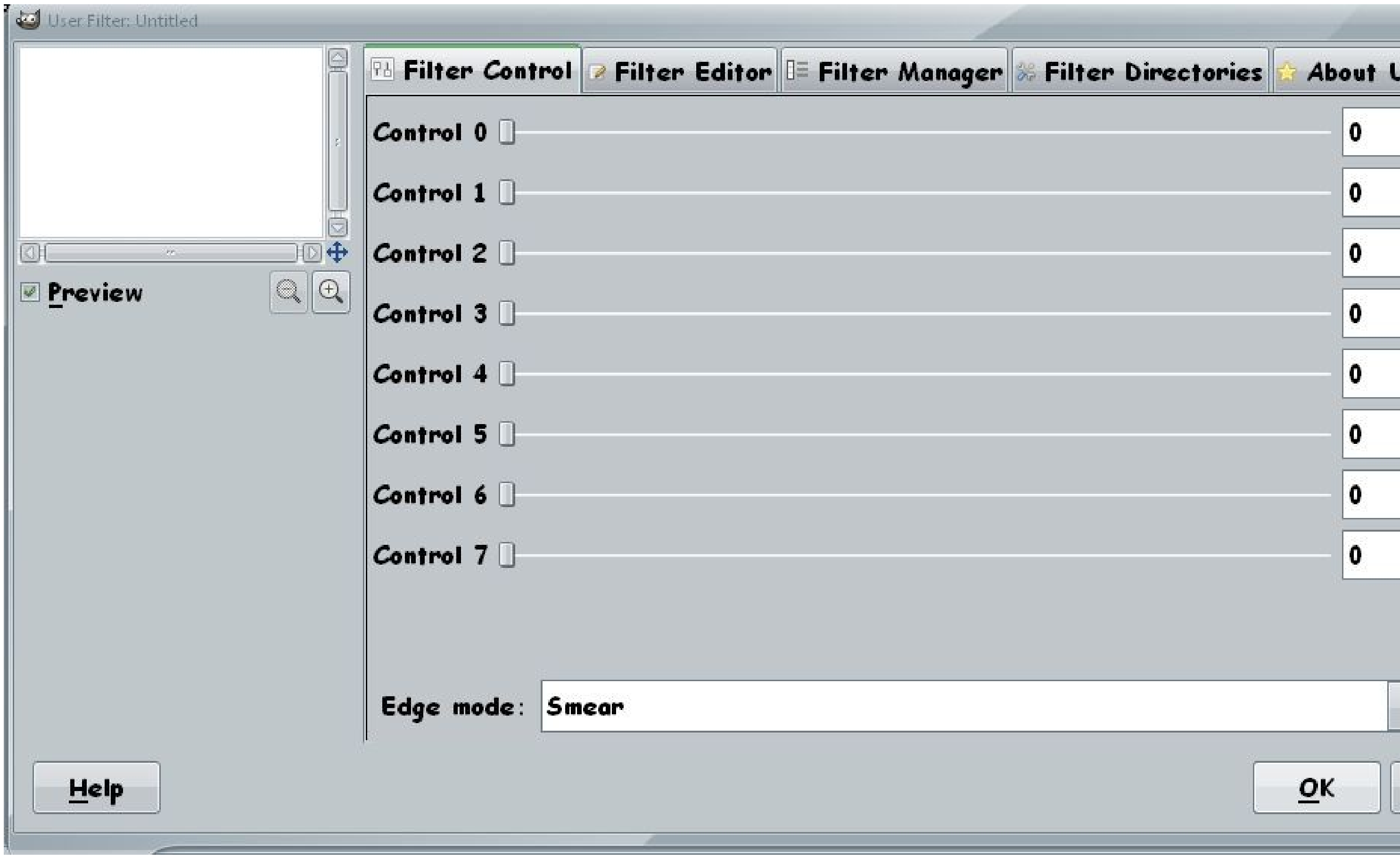
this sort of black DOS window full of criptic warnings


C:\Documents and Settings\Client\.gimp-2.6\plug-ins\gimp-plugin-userfilter\userfilter.exe

```
** (gimp-plugin-userfilter:5572): WARNING **: guf_filter_get_locale_ctl_tooltip
(): Fallback to default value: Key file does not have key 'Tooltip'

** (gimp-plugin-userfilter:5572): WARNING **: guf_filter_get_locale_map_tooltip
(): Fallback to default value: Key file does not have key 'Tooltip'

** (gimp-plugin-userfilter:5572): WARNING **: guf_filter_get_locale_ctl_tooltip
(): Fallback to default value: Key file does not have key 'Tooltip'

** (gimp-plugin-userfilter:5572): WARNING **: guf_filter_get_locale_ctl_tooltip
(): Fallback to default value: Key file does not have key 'Tooltip'

** (gimp-plugin-userfilter:5572): WARNING **: guf_filter_get_locale_map_tooltip
(): Fallback to default value: Key file does not have key 'Tooltip'

** (gimp-plugin-userfilter:5572): WARNING **: guf_filter_get_locale_ctl_tooltip
(): Fallback to default value: Key file does not have key 'Tooltip'

** (gimp-plugin-userfilter:5572): WARNING **: guf_filter_get_locale_ctl_tooltip
(): Fallback to default value: Key file does not have key 'Tooltip'

(gimp-plugin-userfilter:5572): Pango-WARNING **: couldn't load font "Comic Sans
MS Bold Italic Not-Rotated 10", falling back to "Sans Bold Italic Not-Rotated 10
", expect ugly output.
```

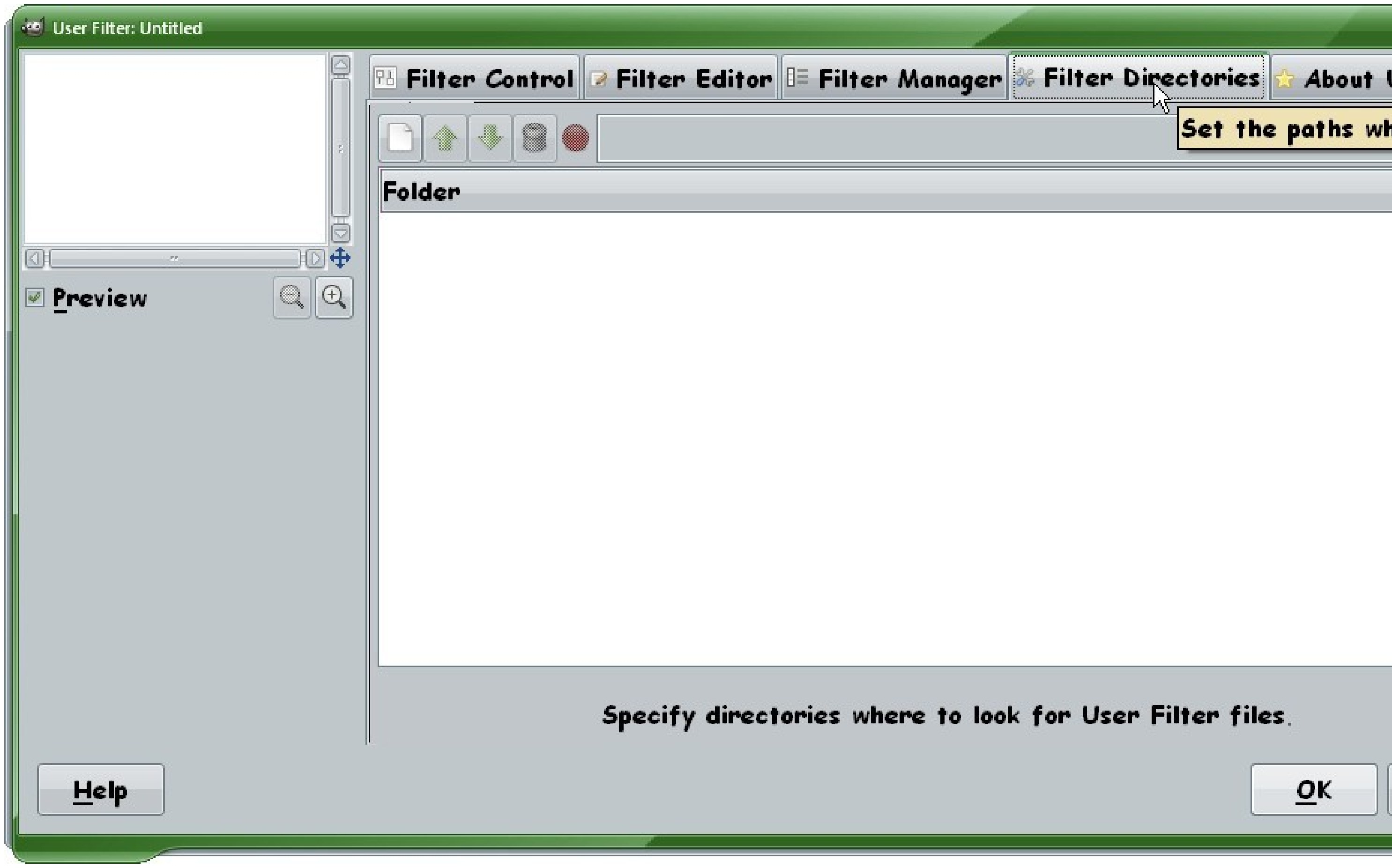
warnings are NOT errors ..You have not to do anything with that you may just minimize
(anyway that window will automatically close with the plugin)
You may minimize it but **NOT** close :that would crash the plugin

Few seconds after that black windows will pop out the real thing:
the plugin interface



Control 0 0
Control 1 0
Control 2 0
Control 3 0
Control 4 0
Control 5 0
Control 6 0
Control 7 0
Edge mode: Smear

You may be tempted to play with that sliders, but would be a delusion, nothing will happen.
First you need to load the plugin with some filters .
As example with some filters from the GUF folder included in the pack that contain 1500 already converted
Filter Factory filters...for that you should click on the "Filter Directories" tab

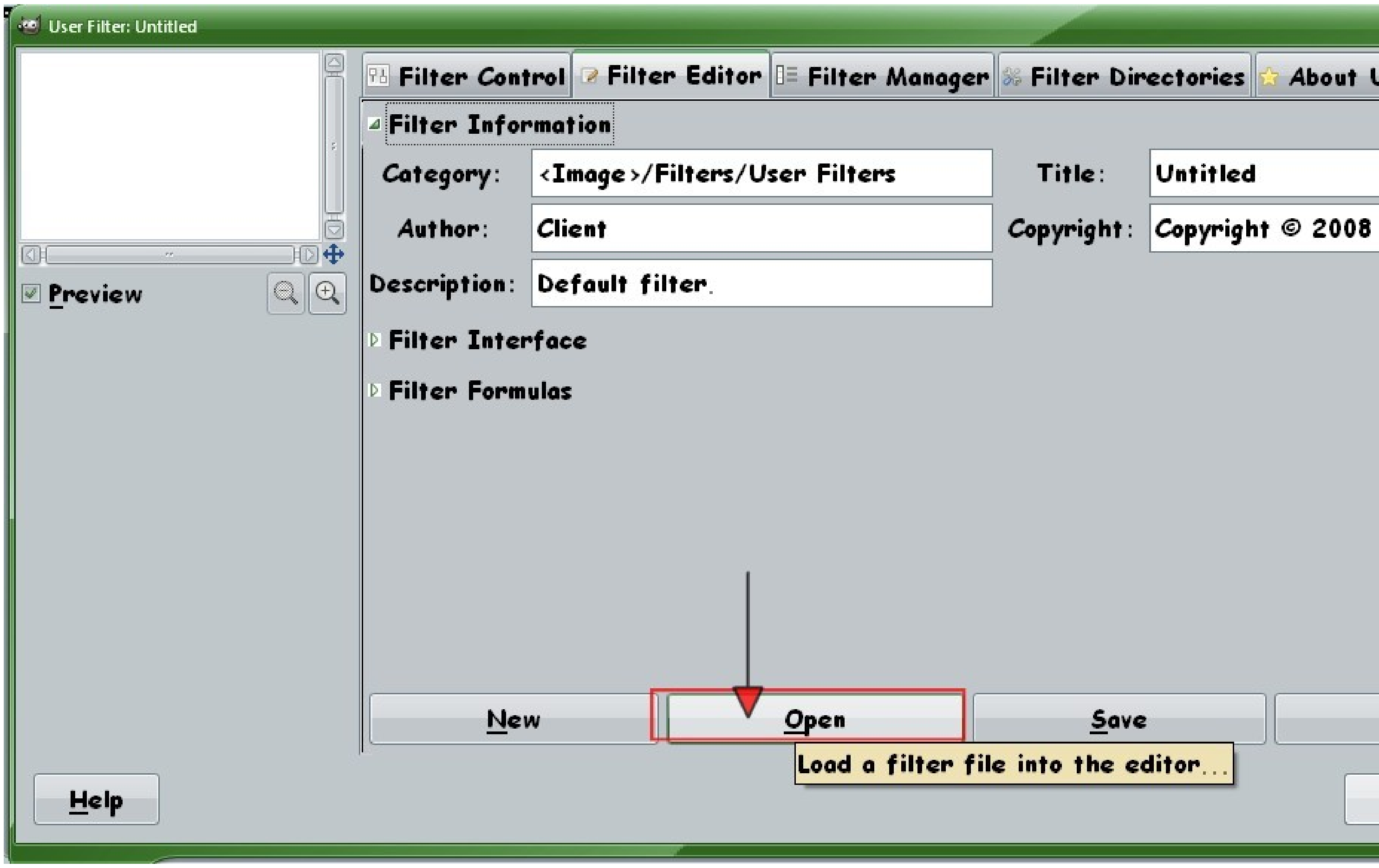


Specify directories where to look for User Filter files.

This dialog works exactly as the Gimp preference dialog explained in HOW TO INSTALL :
first a click on the icon to the right to create a new path,
then one to the icon most on the left (in the screenshoot is covered by the tip) to search.
..and is done

This work only for already converted filter, (Guf) for the other you have 2 possibilities

Firs possibility is click on the "Filter Editor" tab and there use the "Open" button, as explained below



Category: <Image>/Filters/User Filters

Title: Untitled

Author: Client

Copyright: Copyright © 2008

Description: Default filter.

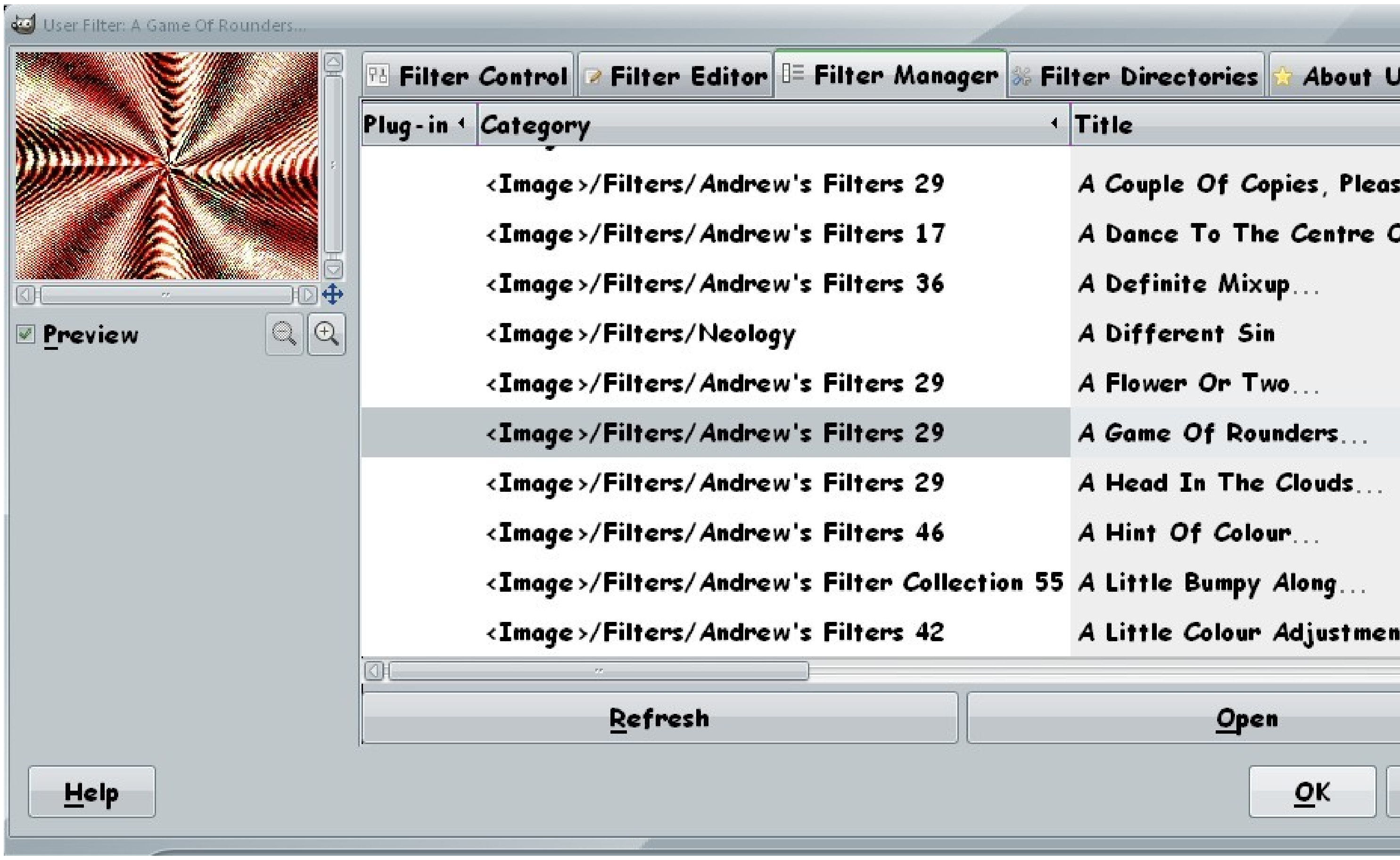
Load a filter file into the editor...

You can load existing filters from this panel, too, regardless whether they are located in your User Filter data directories or not. The User Filter plug-in recognizes four different types of files that can be loaded:

1. *.*guf* – the native file format of the User Filter plug-in
2. *.*8bf* – a (Windows) binary filter for Photoshop
3. *.*afs* – a source file for Filter Factory or Filter Foundry
4. *.*txt* – a source file for the Plugin Commander

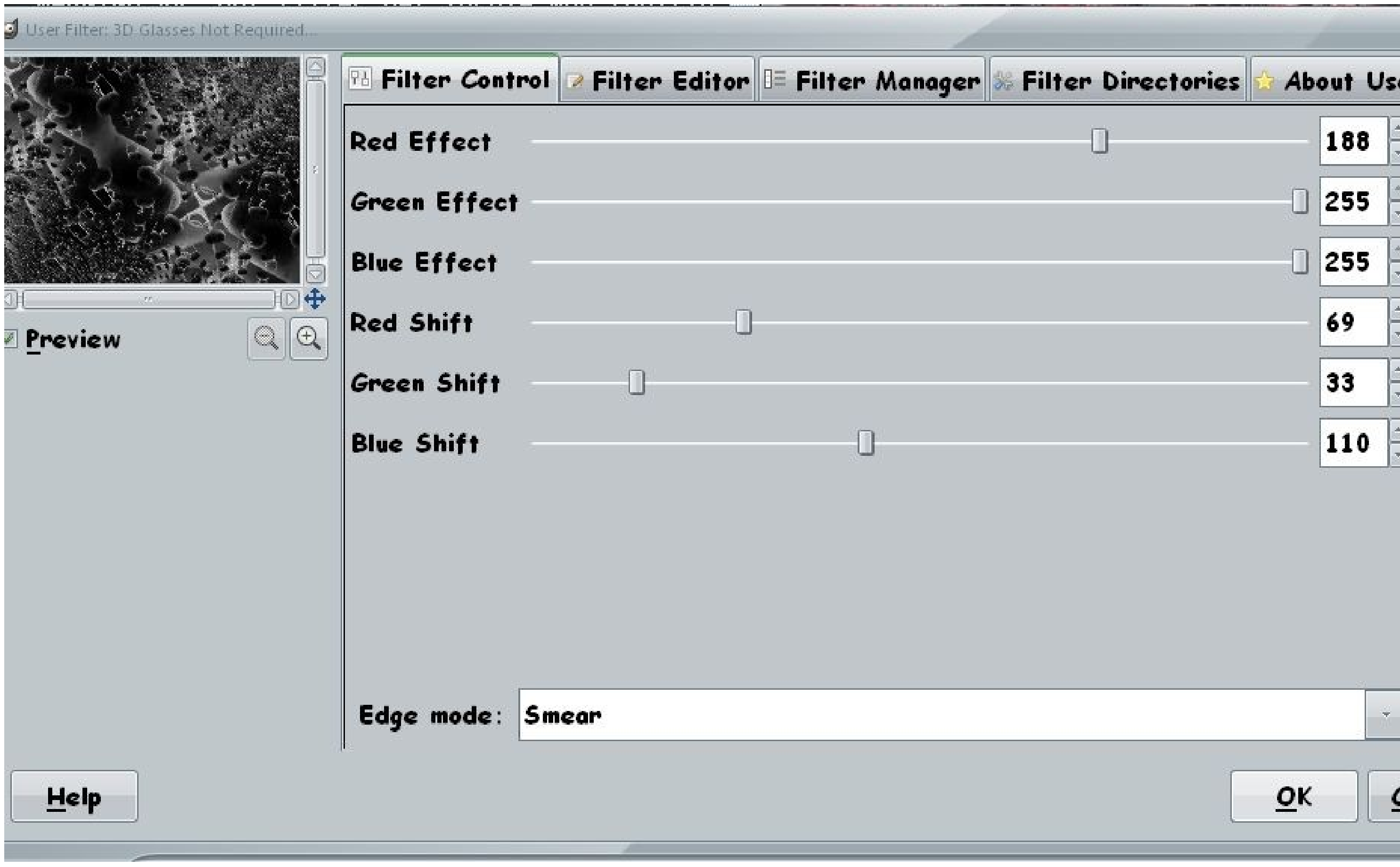
when saving filters, you should always save them to the native file format of the User Filter plug-in that is .guf

once loaded filters are dispayed in the Filter Manager ...Note that clicking on the filter name you may get a
preview of the default filter effect


User Filter: A Game Of Rounders...

Plug-in	Category	Title
	<Image>/Filters/Andrew's Filters 29	A Couple Of Copies, Pleas
	<Image>/Filters/Andrew's Filters 17	A Dance To The Centre C
	<Image>/Filters/Andrew's Filters 36	A Definite Mixup...
	<Image>/Filters/Neology	A Different Sin
	<Image>/Filters/Andrew's Filters 29	A Flower Or Two...
	<Image>/Filters/Andrew's Filters 29	A Game Of Rounders...
	<Image>/Filters/Andrew's Filters 29	A Head In The Clouds...
	<Image>/Filters/Andrew's Filters 46	A Hint Of Colour...
	<Image>/Filters/Andrew's Filter Collection 55	A Little Bumpy Along...
	<Image>/Filters/Andrew's Filters 42	A Little Colour Adjustmen

Then the filtrel controls are available from the Filter Control tab
You may note the difference with image 2, this is the same tab but now the controls are all active



Red Effect 188
Green Effect 255
Blue Effect 255
Red Shift 69
Green Shift 33
Blue Shift 110
Edge mode: Smear

This should be sufficent to start use the Filter Factory Converter
More may be found   in the FAQ written by the plugin developer and reported here below

There is anorther and much quicker way to load and convert Factory Filters then do one by one from the Filters Editor tab
 That allow batch conversion , is quite easy

BUT since that function is not available directly from the plugin interface but only
from (included) command line utilities is discussed on the next Charpter "ADVANCED"

There also you will find everything on edit and create new filters, the specific of GUF language and links for tutorials and guide
to the Filter Factory language

Filter Factory Converter (User Filter) FAQ

================================

Q: What does this plug-in do ?

A: This plug-in will enable you to use filters generated with Photoshop Filter Factory. These filters come in various file formats: text (.txt, .ffl, .afs) and binary (.8bf). There is a catch, however, in the fact that not all .8bf files are Filter Factory filters and thus are not understood by this plug-in.
You can also write your own Filter Factory style filters with this plug-in and share them with users of the original Filter Factory.

Q: I already have the Photoshop filter interface plug-in pspi installed. So what is this plug-in good for then ?

A: The pspi plug-in will also be able to utilize the .8bf plug-ins generated by Filter Factory. But it has its drawbacks, since (a) the initialization takes up a lot time, (b) the menu gets clustered with filters (if you have quite some), and (c) every Filter Factory binary plug-in is at least 48K in size - for the "Andrew's Filters" collection this sums up to about 58M on your disc whereas the same collection as .guf will take only about 1500K.
Furthermore, there are also people who are not working on x86 based machines. They would not be able to run pspi - but they can use this plug-in whatsoever.
Filter Factory binary filters have also no means of localization. When you import those filters in User Filter, however, they can be localized by adding the locale data for the control tags that are defined by the filter. In addition, you also get a filter description (which is used as a tooltip in the Gimp's menu) and tooltips for the filter controls (all of which also can easily be adopted to the current locale).
You can also translate the filters into native code Gimp plug-ins using the uf2c(1) tool. The size of the binary (with statically linked userfilter library) will then be somewhat smaller than a Filter Factory .8bf file. It will also execute faster than the same filter run by the byte-code interpreter of User Filter.

Q: But I like the idea of being able to access the filters from the filters menu of the image.

A: You can also do that with User Filter. In the Filter Manager
dialog of the plug-in, you can select (or deselect) the filters
you want in your Gimp menu (you will have to restart Gimp though,
to take those changes into effect). You are still able to access
any other filter through the manager panel or directly load them
from the editor panel of the plug-in.
The filters need to be in a writable directory though to make this
happen (i.e. you won't be able to change the settings for filters
in a system-wide data directory or in the data directory of another
user).
You can also translate the filter descriptions into native code
plug-ins for the Gimp using uf2c(1). These will also install in
the Gimp's menu once you've moved them into your plug-in directory.
You need to make sure, however, that for the filter to be used in
the stand-alone mode or as a binary plug-in, the "Category" of the
filter needs to be set to a correct Gimp menu path.

Q: I checked the "Plug-in" column of a filter in the manager panel, but the filter does not show up in the Gimp's menu, even when I re-start the program.

A: You need to set a correct menu path for the filter in the "Category"
entry of the editor panel. If you fail to comply, The Gimp cannot
install the filter in its menu.

Q: I understand that the formulas for the filters get interpreted by the plug-in for every pixel in the image. This must be very slow. The .8bf Filter Factory plug-ins are surely quicker, because they are Windows executables, even if they load slower with pspi.

A: The .8bf Filter Factory binary plug-ins do just the same as this
plug-in: they first byte-compile the formulas and then interpret
the byte-compiled code.

Q: I understand that the original userfilter just worked on .8bf, .txt, .afs, and .ffl files. Why is there another file format now and what is it good for ?

A: The original file formats are a mess. They are unstructured and
awful to parse. You can also damage them by editing the text
format files in a normal text editor program, because this could
destroy the way the end-of-line is handled in the files (some are

formatted in the Mac way, others as DOS text etc.) which is important for the file load routines.
Last, but not least, it is not possible to expand those formats. User Filter is not Filter Factory (although it might act like it and can understand its files). It already has a number of features that cannot be stored in the old file formats (e.g. a meta description of what a filter does, the tooltips, or the locale data), but which are of interest to The Gimp. Also, future extensions (like new functions, mathematical constants etc.) cannot be handled by the old files, since there is no way to check the compatibility of a filter with the version of the plug-in that is in use. The new file format has a feature version information tag that is used for this purpose.

Q: I have used User Filter in the past with Gimp v1.x.x - now I cannot find the tons of filters that used to work with it although I have installed them correctly in the userfilter data directory.

A: Things have changed. The original file formats were a mess and induced several bugs. It has therefore been decided that the plug-in will only accept its new native file format as a proper filter description when looking for (new) filters in its data directories.
You can still import the old filters (in the editor panel) or convert them with one of the command line utilities (8bf2guf(1), afs2guf(1), ffl2guf(1), and txt2guf(1)).

Q: But it is soooo slow - this is not usable for my monster size images.

A: Actually, it is quite fast - about 30% faster than the original (v0.8) plug-in - but since the byte code compiler can do only little optimization, speed of the filter still depends highly upon the quality of the code written and the complexity of the formula.
To get around this, there is a converter uf2c(1) which will produce a C source for a standalone GIMP plug-in from a filter description. This source can then be translated with an optimizing C compiler using gimptool(1) and put in the GIMP plug-ins directory.
<future-version>
For your convenience, there is also a wrapper uftool(1) that automates this process (translate filter into C, apply gimptool) for you.
A standalone plug-in produced by this tool should be significantly faster (although not quite as fast as a plug-in hand-written in C) than the byte-compiled filter.
</future-version>
Still, in most cases, a slow-running filter means that it contains code,

that could be optimized by the author.

Q: I have translated a filter with uf2c(1). Now, I want to extend it (add funtionality and/or locale data), but I seem to have lost the original filter file. Is there a way to decompile a plug-in created with uf2c ?

A: Yes and no. There is no way to decompile a plug-in created with an optimizing C compiler (which is what you did). However, the filter code is still in the binary (it is needed for the control value presets, the labels / tooltips, and locale data; the formulas are not needed, but small enough to keep, so their extra payload does not really enlarge the memory footprint of a compiled plug-in significantly). You will be able to extract this data using the strings(1) command or your favourite data dumper, unless the plug-in source was generated using the --strip option of uf2c(1) (which removes comments and function code from the filter data).

Q: I downloaded a FF filter from the internet and it does not work.

A: Some FF filters use undocumented features (I have seen some using a value "t" for instance, which does not show up in the FF documentation). Others include invalid characters in the code portion of the filter which might confuse the parser.
If you happen to have a filter that results in syntax errors, it is most likely that it is because of one of the two reasons stated above. You then can correct the formula in the editor panel and save the corrected filter.
There are also different versions of FF (Windows and Macintosh), that use different return values for certain trigonometric functions.
User Filter uses the specification of the Macintosh variant of FF, so filters that have been written for the Windows version might need to be tweaked a little to give the intended result.
If else the formulas compile alright but the filter does not work as expected, it is probably a bug in the code generator or the optimizer of User Filter treats. This should be, however, not very likely.

ADVANCED

1.2 User Filter Functional Language

Any filter consists of a set of functional expressions that are applied for each pixel in an image (or a selection in the image respectively). The plug-in will apply these expressions in sequence R,G,B,A – this means, that you can store the results of certain subexpressions and re-use them in the next channel's functional expression.

The User Filter plug-in has been designed to be backward compatible with Filter Factory / Filter Foundry. It will understand the same functional language that is understood by these Photoshop plug-ins. This means that you can not only import filters that have been created with or for those plug-ins, but you can also use any tutorial or manual from these plug-ins for your work with User Filter as well. However, since there exist two versions of FF that differ from each other with regards to the values returned by certain functions and some maximum values, you should consult the section about portability in order to get the intended results with filters that have been written for FF.

Functional expressions follow a notation that closely resembles the syntax found in the “C” programming language.

1.2.1 Constants

The User Filter functional language has two kinds of constants: “true” constants, i.e. those that have a predefined value, image environment dependent constants, i.e. constants that have different values for every image, and pixel environment dependent constants, i.e. constants that have different values for every pixel within the image. Any of the named constants are often called “variables” in manuals and tutorials for Filter Factory / Filter Foundry, but since they cannot be modified by any of the functional expressions, we rather call them constants here.

Name	**Description**
`Integer literals`	Numeric values can be written in octal, decimal, and hexadecimal in the same notation as in the “C” programming language, i.e. any number starting with a *0* will be treated as an octal value, unless it also contains the digits *8* or *9*; any number starting with *0x* will be treated as a hexadecimal number.
`rmin, gmin, bmin, amin, cmin`	Minimum value that can be applied to the “red”, “green”, “blue”, “alpha”, and “current” color channel (*0*).
`rmax, gmax, bmax, amax, cmax`	Maximum value that can be applied to the “red”, “green”, “blue”, “alpha”, and “current” color channel (*255*). *Note, that this value will change from a true constant into an image dependent constant when wider color channels will get supported*[1].
`R, G, B, A, C`	Range of channel values such that $R = \|rmin...rmax\|$, $G = \|gmin...gmax\|$, $B = \|bmin...bmax\|$, $A = \|amin...amax\|$, and $C = \|cmin...cmax\|$.
`r, g, b, a, c`	Value of the current pixel in the “red”, “green”, “blue”, alpha”, and the “current” color channel, where $rmin <= r <= rmax$, $gmin <= g <= gmax$, $bmin <= b <= bmax$, $amin <= a <= amax$, and $cmin <= c <= cmax$ respectively.
`imin, umin, vmin`	Minimum color values in YUV color space, where $imin = 0$, $umin = -56$, and $vmin = -78$.
`imax, umax, vmax`	Maximum color values in YUV color space, where $imax = 255$, $umax = 56$, and $vmax = 78$.
`i, u, v`	Current pixel value in YUV color space, where $imin <= i <= imax$, $umin <= u <= umax$, and $vmin <= v <= vmax$.
`xmin, ymin`	Minimum horizontal and vertical coordinates (*0*).
`xmax, ymax`	Maximum horizontal and vertical coordinates. This depends on the dimensions of the image.
`X, Y`	Range of horizontal and vertical image coordinates (width and height of the image), such that $X = \|xmin...xmax\|$, and $Y = \|ymin...ymax\|$.
`x, y`	Current horizontal and vertical coordinate of the pixel that is processed, where $xmin <= x <= xmax$ and $ymin <= y <=ymax$.
`dmin`	Minimum angle within the image (*0*).
`dmax`	Maximum angle within the image (*1023*)[2].

1

`D`	Range of angles within the image, so that $D = \lvert dmin \ldots dmax \rvert = 1024$.
`d`	Angle of the current pixel from the center of the image, where $dmin <= d <= dmax$.
`mmin`	Minimum distance (“magnitude”) of the current pixel from the center of the image (*0*).
`mmax`	Maximum distance (“magnitude”) of the current pixel from the center of the image, where $mmax = (M = (\sqrt{(X^2 + Y^2)}) / 2) - 1$.
`M`	Range of magnitudes of the current pixel from the center of the image, where $M = \lvert mmin \ldots mmax \rvert = (\sqrt{(X^2 + Y^2)}) / 2$.
`m`	Distance (“magnitude”) of the current pixel from the center of the image where $mmin <= m <= mmax$.
`zmin`	Minimum channel index (*0*).
`zmax`	Maximum channel index, *depending on the color mode of the image, where 0 = Gray < GrayA < RGB < RGBA = 3.*
`Z`	Range of channel indexes within one pixel, so that $Z = \lvert zmin \ldots zmax \rvert$.
`z`	Channel index for the current expression, where $0 = R < G < B < A = 3$.

Any of the “true” constants are treated like integer literals by the byte-code generator of User Filter and the plug-in translator uf2c. There is no difference between writing the constant name or its integer representation except where it comes to portability. The byte-code optimizer of User Filter will also reduce the code tree for *any constant subexpression, except where pixel dependent constants are used*.

Unlike as it is documented in Filter Factory, using *i*, *u*, or *v* in your functions is *not* slower than using *r*, *g*, or *b*.

1.2.2 Variables

There are no true variables in the User Filter functional language (since you cannot “assign” values). It is possible, however, to store up to 256 different values in an indexed temporary store (and retrieve these values later). See the “put (y, i)” and “get (i)” functions for more information.

1.2.3 Operators

The operators are used in the functional language of User Filter as are used in the “C” language and with the same semantical effects (except for the assignment operators, which are not available here).

All arithmetic operations are done using 32-bit integer values. There are neither floating point nor fixed point operations.

All logical operations consider the value *0* being *false,* all other values being *true*. If a function or operation returns the value of *true*, it will effectively return the numeric value *1*.

Operator	**Syntax**	**Description**
`,`	*`expression , expression`*	*Sequence*. The comma operator is used to chain multiple expressions into a single expression. Execution of code is from left to right – both subexpressions are evaluated and the result value of the right expression is returned.
`?:`	*`expression ? expression : expression`*	*Conditional*. The conditional operator evaluates the expression before *?* and if the result is not *0* returns the result of the first subexpression, else it returns the result of the second subexpression. Only one of the two conditional subexpressions is evaluated.
`&&`	*`expression && expression`*	*Logical and.* Returns *0* (false) if one of the two subexpressions is *0* (false), else *1* (true).
`||`	*`expression || expression`*	*Logical or.* Returns *1* (true) if one of the two subexpressions is *1* (true), else *0* (false). The logical operators are shortcut operators, i.e. if the result of the first subexpression would also be the result of both subexpressions, only the first subexpression gets evaluated (e.g. *1* || *0* or *0 && 1*).
`&`	*`expression & expression`*	*Bitwise and*. For any bit that is set in both expressions the bit is also set in the result.
`^`	*`expression ^ expression`*	*Bitwise exclusive or*. Returns a set bit for any bit that is set in the first or the second subexpression, but not in both.
`|`	*`expression | expression`*	*Bitwise inclusive or*. Returns a set bit for any bit that is set in one of the two subexpressions. The bitwise operators are full-evaluation operators, i.e. both subexpressions are

		evaluated.
`==`	*`expression == expression`*	*Equal to*. Returns *1* (true) if the results of the left and right subexpression are equal, else *0* (false).
`!=`	*`expression != expression`*	*Not equal to*. Returns *1* (true) if the results of the left and right subexpression differ, else *0* (false).
`<`	*`expression < expression`*	*Lesser than*. Returns *1* (true) if the results of the left subexpression is lesser than the result of the right subexpression, else *0* (false).
`<=`	*`expression <= expression`*	*Lesser than or equal to*. Returns *1* (true) if the results of the left subexpression is lesser than or equal to the result of the right subexpression, else *0* (false).
`>`	*`expression > expression`*	*Greater than*. Returns *1* (true) if the results of the left subexpression is greater than the result of the right subexpression, else *0* (false).
`>=`	*`expression >= expression`*	*Greater than or equal to*. Returns *1* (true) if the results of the left subexpression is greater than or equal to the result of the right subexpression, else *0* (false).
`<<`	*`expression << expression`*	*Bitwise shift left*. Shifts the bits of the result of the left subexpression result of the right subexpression times left. This operation is equivalent to an n-times multiplication of the left subexpression with *2*.
`>>`	*`expression >> expression`*	*Bitwise shift right.* Shifts the bits of the result of the left subexpression result of the right subexpression times right. This operation is equivalent to an n-times division of the left subexpression by *2*.
`*`	*`expression * expression`*	*Multiplication*. Returns the result of multiplication the left subexpression by the right subexpression.
`/`	*`expression / expression`*	*Division*. Returns the result of the left subexpression divided by the right subexpression. If the right subexpression

		returns *0*, the operation will return *1*.
%	*expression % expression*	*Modulo division*. Returns the remainder of the left subexpression divided by the right subexpression. If the right subexpression returns *0*, the operation will also return *0*.
+	*expression + expression*	*Addition*. Returns the value of the left subexpression plus the value of the right subexpression.
-	*expression - expression*	*Subtraction*. Returns the value of the left subexpression minus the value of the right subexpression.
!	*! expression*	*Logical not*. Returns *1* (true) if the result of the right expression is *0* (false), else *0*.
~	~ expression	*Bitwise not*. Inverts all bits in the result of the right expression.
-	- expression	*Negation*. Returns the negative value of the result of the right expression.

1.2.4 Functions

Function	**Description**
ctl (i)	Value of slider *i*, where *0 <= i <= 7* and *0 <= ctl (i) <= 255*.
val (i, a, b)	Value of slider *i*, where *0 <= i <= 7* and *0 <= ctl (i) <= 255* mapped onto the range *[a...b]* so that *a <= val (i, a, b) <= b*.
map (i, n)	Return entry *n* from mapping table *i*, where *0 <= n <= 255* and *0 <= i <= 3*. Each mapping table uses a pair of sliders, *ctl (2 * i)* and *ctl(2 * i + 1)* for the high and the low values of the mapping table formula. The entries of the mapping table are computed so that *if n <= ctl (2 * i + 1) then 0 else if n >= ctl (2 * i) then 255 else if ctl (2 * i + 1) < n < ctl (2 * i) then (n - ctl (2 * i + 1)) * 255 / (ctl(2 * i) – ctl(2 * i + 1)) else 0*.
src (x, y, z)	Return the channel value of the pixel at the cartesian coordinates *x, y* in color channel *z*, where *xmin <= x <= xmax*, *ymin <= y <= ymax*, and *zmin <= z <= zmax*. For off-image coordinates the channel value depends upon the setting of the edge-mode control[3]. Using this function to read the channel value at the

3

	current pixel coordinates is very inefficient; instead use *r*, *g*, *b*, *a*, or *c*.
`rad (d, m, z)`	Return the channel value of the pixel at the polar coordinates *d, m* in color channel *z*, where *dmin <= d <= dmax*, *mmin <= m <= mmax*, and *zmin <= z <= zmax*. For off-image coordinates the channel value depends upon the setting of the edge-mode control[4].
`cnv (m11, m12, m13, m21, m22, m23, m31, m32, m33, d)`	The convolver function works similar to the Convolution Matrix filter of The Gimp with the difference that the matrix is a little smaller. Every pixel of the current color channel and its neighboring pixels are evaluated to get the result. Please refer to the documentation of the Convolution Matrix filter for the description of this function.
`min (a, b)`	Return the lesser of *a* and *b*, so that *if a < b then a else b*.
`max (a, b)`	Return the greater of *a* and *b*, so that *if a > b then a else b*.
`abs (a)`	Return the absolute value of *a*, so that *if a < 0 then -a else a*.
`add (a, b, c)`	Return the sum of *a* and *b*, or *c*, whichever is lesser, so that *add (a, b, c) = min (a+b, c)*.
`dif (a, b)`	Return the absolute value of the difference of *a* and *b*, so that *dif (a, b) = abs (a-b)*.
`sub (a, b, c)`	Return the difference of *a* and *b*, or *c*, whichever is greater, so that *sub (a, b, c) = max (dif(a, b), c)*.
`rnd (a, b)`	Return a random number in range of *[a...b]*, so that *a <= rnd (a, b) <= b*. The random number generator is seeded with a value computed from the image that is currently processed, thus guaranteeing that the applying the same filter on two identical images will produce the same result. The control panel of the plug-in will show a special control to set the random number seed to a different value when the filter is using the random number function.
`mix (a, b, n, d)`	Return the mixture of *a* and *b* by fraction *n / d*, so that $\text{if } d = 0 \text{ then } 0 \text{ else } a * n / d + b * (d - n) / d$.
`scl (a, il, ih, ol, oh)`	Scale *a* from input range *[il...ih]* to output range *[ol...oh]*.
`sqr (x)`	Return the square root of *x*.
`sin (d)`	Sine function of *d* where *dmin <= d <= dmax* and the result is *-D <= sin (d) <= D*[5].
`cos (d)`	Cosine function of *d* where *dmin <= d <= dmax* and the result is *-D <= cos (d) <= D*[6].
`tan (d)`	Bounded tangent function of d where *-dmax/4 <= d <= dmax/4* and the result is

4
5
6

	-D <= tan (d) <= D[7].
`r2x (d, m)`	Return the *x* displacement of the pixel that is *m* units away from the center of the image at an angle of *d*.
`r2y (d, m)`	Return the *y* displacement of the pixel that is *m* units away from the center of the image at an angle of *d*.
`c2d (x, y)`	Return the angle of the pixel at cartesian coordinate *x, y* from the center of the image.
`c2m (x, y)`	Return the distance of the pixel at cartesian coordinate *x, y* from the center of the image.
`put (v, i)`	Store a value *v* in the temporary store at index *i*, where *0 <= i <= 255*. The return value is so that *if 0 <= i <= 255 then v else 0*.
`get (i)`	Fetches a value from the temporary store at index *i*, so that *if 0 <= i <= 255 then value else 0*.

1.2.5 Portability

The User Filter plug-in has been written with backwards compatibility with Filter Factory for Photoshop in mind. That means that, as a rule of thumb, any filter that has been written with Filter Factory (or Filter Foundry) will also work with User Filter and (with some limitations) vice versa. There are, however, certain issues that needs to be taken care of when it comes to using an filter written for FF with User Filter or writing a filter with User Filter that should be used with FF.

- Many filters written for FF use integer representation of certain values instead of the constants provided. This is not only bad style, but leads to problems. User Filter uses the constant values also used by the Macintosh version of Filter Factory, which in this aspect differs from the Windows variant. When importing a filter, it should be made certain, that the right constants are used in the filter code.

- Filter Factory only supported 8-bit wide color channels, as does the current version of User Filter. The latter will change, however, when The Gimp will support wider color channels. This means, that the constants for the maximum channel values will differ depending on the image being processed. Many filters that have been written for FF do not use the constants for maximum channel width, but use integer literals instead. This needs to be corrected in order to make these filters work properly with future versions of User Filter and images that have wider color channels.

- Filters written with User Filter can also be exported to FF filters and used with FF with the same results, except for the following cases:

    - The filter depends upon the pixel-fetcher edge-mode option being different from FF's default behavior (“smear”). In this case, the filter can be used in FF, but will produce different results for processing off-image pixels than when used in User Filter.

7

- The filter source code exceeds 1024 characters in length per channel. In this case, it is not possible to export this filter to a file format that can be read by FF, since this is a limitation of FF that is not present in User Filter.

- The filter source code utilizes language extensions of User Filter. In this case, it is not possible to export the filter to a file format understood by FF, since FF does not know anything about these language extensions[8].

Please note, that many authors of Filter Factory filters did not understand the difference between constants like X (which is the number of values in a range) and xmax (which is the maximum value of a given range). There are many cases where the maximum value if off by 1 due to this misunderstanding. Most of the cases, you will not recognize this, because FF's mode for handling off-image values defaults to “smear”, but in other cases you may see a 1-pixel wide artifact at the edge of the image after processing it with a filter.

1.3 User Filter External Tools

A set of external tools is provided with the User Filter plug-in which are meant as helper tools for importing existing Filter Factory / Filter Foundry filters into the native filter file format of the User Filter plug-in and to translate filter files into native plug-ins for The Gimp.

1.3.1 The 8bf2guf, afs2guf, ffl2guf, and txt2guf Command Line Utilities

These programs can be used to quickly convert many Filter Factory filters into the native filter file format of the User Filter plug-in. They have been named, so that they match the filter filename extensions of the filter file types they have been written for, i.e. use *8bf2guf* to convert **.8bf* filters, *afs2guf* to convert **.afs* filters and so on.

Since you can also load the FF filter file formats (except the **.ffl* format, which is not a single filter, but instead a collection of filters) from within the editor panel of the User Filter plug-in, these external utility programs are meant for converting a larger number of filters at a time. For each filter processed by these tools, a single filter file will be created in **.guf* file format. Where necessary, these tools will also fill the additional entries in the **.guf* file with information (like tool-tips, filter description, version and date information). However, filter conversion is not perfect (the least it is for the proprietary **.8bf* binary format) and User Filter can use a lot more information than Filter Factory or Filter Foundry are able to, so you will most likely need to hand-tune the filter information in the editor panel of the plug-in to make them look nice and integrate seamlessly in The Gimp (if you want to register them as stand-alone filters in The Gimp's menu). Please refer also to the section about portability of this document for further things that you might need to take care of in order to make the filters work as intended.

Please refer to the man-pages (8bf2guf(1), afs2guf(1), ffl2guf(1), txt2guf(1), 8bf(5), afs(5), ffl(5), guf(5)) for more details.

1.3.2 The uf2c Translator

This program will translate a filter into the “C” source code required to produce a native plug-in for The Gimp. These plug-ins will use the same user interface already known from the User Filter plug-in itself when running a filter in stand-alone mode. As a rule of thumb, translated filters will execute faster than those

8

that are run from the User Filter plug-in itself, since the latter will be interpreted byte-code while translated filters are machine code. However, do not expect too much of it – this will not turn a stock car into an F1 racer. Also, the User Filter plug-in is able to reduce the code tree significantly more than it is possible with a translated version of the same filter. This is due to the fact that all user interface derived values can be treated as constants prior to final application of the filter, while only compile-time optimization can be performed on the translated filter. The speed gained from translating a filter with uf2c will be quite noticeable for very complex filters though or for filters that use a lot of computation base on the current pixel coordinates.

Please refer to the man-pages (uf2c(1), guf(5)) for more details.

Please note, that many of the existing filters (most of them written for Filter Factory) use integer representations of the maximum channel values. This will lead to problems when wider color channels are used. Whenever a filter that has been written for Filter Factory is to be used with User Filter, it should be checked that the correct constant is used.

Please note, that Filter Factory for Adobe Photoshop unfortunately used two different values for the return results of the trigonometric functions (sin, cos, tan) in the Windows (512) and the Macintosh (1024) versions of the program. The User Filter plug-in for The Gimp uses the value also used by the Macintosh version of Filter Factory. This may lead to different results when using filters that have been written for the Windows variant of Filter Factory. When importing a filter, this should be checked and corrected whenever necessary.

This is an Extension to the functionality of Filter Factory / Filter Foundry. Use the default edge-mode (“smear”) for fully backwards compatible filters.

This is an Extension to the functionality of Filter Factory / Filter Foundry. Use the default edge-mode (“smear”) for fully backwards compatible filters.

Please note, that Filter Factory for Adobe Photoshop unfortunately used two different values for the return results of the trigonometric functions (sin, cos, tan) in the Windows (512) and the Macintosh (1024) versions of the program. The User Filter plug-in for The Gimp uses the value also used by the Macintosh version of Filter Factory. This may lead to different results when using filters that have been written for the Windows variant of Filter Factory. When importing a filter, this should be checked and corrected wherever necessary.

Please note, that Filter Factory for Adobe Photoshop unfortunately used two different values for the return results of the trigonometric functions (sin, cos, tan) in the Windows (512) and the Macintosh (1024) versions of the program. The User Filter plug-in for The Gimp uses the value also used by the Macintosh version of Filter Factory. This may lead to different results when using filters that have been written for the Windows variant of Filter Factory. When importing a filter, this should be checked and corrected wherever necessary.

Please note, that Filter Factory for Adobe Photoshop unfortunately used two different values for the return results of the trigonometric functions (sin, cos, tan) in the Windows (512) and the Macintosh (1024) versions of the program. The User Filter plug-in for The Gimp uses the value also used by the Macintosh version of Filter Factory. This may lead to different results when using filters that have been written for the Windows variant of Filter Factory. When importing a filter, this should be checked and corrected wherever necessary.

Currently, User Filter does not have any language extensions, but several extensions are planned, like access to the current foreground and background colors, access to another image or layer as alternative image sources, etc.