# The GIMP

The General Image Manipulation Program
Version 0.5

by Peter Mattis and Spencer Kimball

# The GIMP Copying Conditions

The GIMP is *free*; this means that everyone is free to use it and free to redestribute it on a free basis. The GIMP is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of the GIMP that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of the GIMP, that you receive source code or else can get it if you want it, that you can change the GIMP or use pieces of it in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the GIMP, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the GIMP. f the GIMP is modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will no reflect on our reputation.

The precise conditions of the licenses for the GIMP are found in the General Public Licenses that accompanies it.

# 1 Genesis of the GIMP

The GIMP arose from the ashes of a hideously crafted cs164 (compilers) class project. The setting: early morning. We were both weary from lack of sleep and the terrible strain of programming a compiler in LISP. The limits of our patience had long been exceeded, and yet still the dam held.

And then it happened. Common LISP messily dumped core when it could not allocate the 17 MB it needed to generate a parser for a simple grammar using “jyack”. An unbelieving moment passed, there was one shared look of disgust, and then our project was vapor. We had to write something...ANYTHING...useful. Something in C. Something that did not rely on nested lists to represent a bitmap. Thus, the GIMP was born.

Like the phoenix, glorious, new life sprung out of the burnt remnants of LISP and jyacc. Ideas went flying, decisions were made, the GIMP began to take form.

An image manipulation program was the consensus. A program which would at the very least lessen the necessity of using commercial software under “Windoze” or on the “Macintoy”. A program that would provide the features missing from the other X painting and imaging tools. A program that would help maintain the long tradition of excellent and free UNIX applications.

Six months later, we’ve reached an early BETA stage. We want to release now to start working on compatibility issues and cross platform stability. Also, we feel now that the program is actually usable and would like to see other interested programmers developing plug-ins and various file format support.

# 2 Introduction to the GIMP

The GIMP is designed to provide an intuitive graphical interface to a variety of image editing operations. The following list describes parts of the GIMP's functionality.

**Views**. A window on the screen is essentially a viewport into the image space. This means that multiple windows can look into the same space. By this same reasoning, changing the image in one window causes the change to propogate to all the other windows.

**Undo**. There is no restriction on the number of possible undo/redo operations. However, limited memory means that for practical purposes limiting the size and length of the undo stack is necessary. The GIMP allows these values to be customized by the user. Undo/redo works for almost all operations. The few operations which cannot be undone are normally have semantics that would make undo unwanted. For instance, duplicating an image creates a new image from an old one. Undo-ing this operation would consist of destroying the new image, something the user might not expect.

**Selections**. Selections allow the user to manipulate areas of an image. They are used to mask operations during painting and plug-in usage and to specify the active area when using the transformation tools. See ⟨undefined⟩ [Selections], page ⟨undefined⟩, for more info.

**Crop**. The crop tool allows the quick and easy cropping of an image.

**Painting**. The painting tools provide the ability to edit an image in a variety of ways. Different operations include adding brush strokes, cloning parts of an image, convolving parts of an image, filling parts of an image and adding text.

**Transforming**. The transformation tools operate on selections. They modify both the selection and the pixels within the selection. The currently available transformations are rotate, scale, shear, flip horizontal and flip vertical.

**Plug-ins**. Plug-ins provide a mechanism for easily extending the GIMP in certain areas. Namely, operations that work on images and that do not require interactive user input. Currently, plug-ins fill the role of loading and saving images as well as performing various effects on images. See ⟨undefined⟩ [Plug-ins], page ⟨undefined⟩, for more info.

# 3 Creating, modifying and using selections

A *selection* is defined as the currently active region of an image. Selections can be made with single pixel precision, and can be arbitrarily complex. A GIMP selection is represented graphically by a boundary line drawn around the represented region. The line appears to move, and so is easy to discern even when used with a complex image. This moving lines can at times be distracting, or obscure important image information; they can be toggled on or off via the (/Edit/Selection Toggle) menu item.

Selections are created and modified using the suite of selection tools which include: rectangular, elliptical, free hand, fuzzy, bezier, and intelligent. Each successive use of a tool replaces the current selection with a new one. However, the current selection can be changed instead of replaced by holding down either the ***shift*** or ***control*** keys. The ***shift*** key has the effect of taking the union of the current selection and the new selection. The ***control*** key takes the difference.

While a selection is active in an image window, operations only affect the portion of the image which is inside the selection. Effects filters, for example, only change the pixels defined by the active selection. The various painting tools work in the same fashion.

## 3.1 Editing Selections

Intuitively, clicking with the mouse inside a selection allows the selection to be dragged to a new location. This functionality is only present when one of the selection tools is the active tool. The GIMP represents a selection being dragged by its outline or its bounding box. The bounding box is useful when the outlines become very complex and the host machine is slow. The default is to use the selection's outline, but bounding boxes can be enabled through a command line switch or an X resource.

Selected regions can be precisely positioned using the keyboard's cursor keys. Holding down the ***shift*** key has the effect of increasing the step size of each movement.

Standard editing functions are available for use on selections. In a basic sense, cut, copy, & paste work on selected image areas just like they might work on selected text. Cutting a selected region removes both the selection and the associated image region, replacing the former pixel colors with the background color. The cut selection is now stored in a global buffer, from which the cut selection can be pasted any number of times. Copying has the same effect as cut, but does not remove the selection being copied. Successive copies or cuts replace the former contents of the global buffer.

Paste copies the selection in the global buffer to the current image, replacing the image's selection, if one existed. It positions the new selection so that its center corresponds to the center of the former selection. If there was no fomer selection, the pasted selection is centered with respect to the entire image.

A selection that is pasted or moved is considered to be *floating*. That is, the contents of the image under it are preserved. Moving a floating selection to a new location restores the image previously hidden to its former state. A floating selection can be anchored by a invoking the (/Edit/Selection Anchor) menu item. This effectively replaces the contents of the image underneath with the contents of the floating selection, and does away with the “floating” status. Modifying a floating selection’s boundary with selection tools has the same effect.

The transparency of floating selections can be modified through the (Edit/Paste Opacity) menu item. This invokes a dialog with a slider that control the percentage of the floating buffer’s opacity. 100% corresponds to no transparency or full opacity. The lower the opacity is set to, the more transparent the floating buffer becomes until at 0%, the floating buffer is invisible and only the image under it can be seen.

## 3.2 Advanced selection usage

There is support in GIMP selections for an alpha value at each pixel in the selected area. That is, each pixel can have a value from 0 to 255 associated with it. This extra data is used by the GIMP in more complex operations such as anti-aliasing and image composition.

Cutting a selection from an image uses this alpha value to determine how much of a former pixel should remain behind and how much should be cut. If the alpha value is 255, then the entire pixel will be removed, and only the background color will remain. If the alpha value is 0, then nothing will be cut and the pixel on the image will remain the same. If the alpha value is 127, then half of the pixel’s previous color will be cut, and what remains will be a 50/50 combination of the background color and the former pixel color.

A floating selection uses the alpha value to *blend* the color of the pixel it represents with the color underneath that pixel. The effect is transparency in the floating selection. This essentially is how floating selections can achieve anti-aliasing.

A selection’s alpha channel can be edited through the menu item: (/Edit/Selection To Mask), which represents the current selection as an editable grayscale image. White in the image corresponds to an alpha value of 255. Black in the image similarly corresponds to an alpha value of 0. Shades of gray represent all of the intermediate values. This image is identical in every respect to a standard grayscale image, and can be modified in the same ways.

The grayscale image created by (/Edit/Selection To Mask) can be converted back into a selection via the (/Edit/Selection From Mask) menu item. This call pops up a dialog with an option menu requesting the mask source image. It then translates the shades of gray into a new selection with the corresponding alpha values.

Because the global buffer discussed previously can only hold one selection, named selection buffers are supported by the GIMP. Instead of the usual (/Edit/Cut) or (/Edit/Copy), using (/Edit/Cut Named) or (/Edit/Copy Named) pops up a dialog box asking for the name of the new buffer. In this fashion, an arbitrary number of selections can be saved and recalled at convenient times. (/Edit/Paste Named) calls a dialog box which allows any of the previously saved buffers to be pasted.

## 3.3 The selection tools

The GIMP currently supports six different selection tools. In order of appearance in the tool box they are: rectangle, ellipse, free, fuzzy, bezier, and intelligent. Each tool has its advantages and disadvantages–no one tool is clearly superior to the others. In general, a combination of tools proves to be the optimal strategy for complex selection tasks. Following is a more complete description of each tool, including strengths and weaknesses as well as features.

### 3.3.1 Rectangle and ellipse selection tools

These tools are self-explanatory. Rectangles and ellipses can be placed in an image by clicking in the image window with the left mouse button. The width and height are specified by dragging the mouse with the button still pressed. Holding down the 'Shift' and 'Control' keys simultaneously keeps the width and height equal, effectively producing squares or circles. As with all selection tools, holding down 'Shift' when the rectangle or ellipse is initially placed adds the resulting shape to the current selection; 'Control' subtracts from the current selection.

### 3.3.2 Free selection tool

Free select allows an arbitrarily complex curve to be drawn with the mouse which is then scan converted into a polygon and incorporated into the current selection. Free select is especially good for "touching up" selections–either subtracting extraneous pixels or adding a few missing pixels. Selecting large complex shapes is possible, but time consuming and inaccurate.

### 3.3.3 Fuzzy selection tool

Fuzzy select or "magic wand" is initiated by specifying a seed pixel with the left mouse button. The resulting selection will include the seed pixel and contiguous pixels of similar color. Essentially, the selection flows outward from the seed pixel until it reaches pixels which are sufficiently dissimilar in color. The threshold value which determinates what is "sufficient" can be dynamically set by dragging the mouse with the left button still pressed. Moving left or up decreases the threshold, shrinking the selection. Moving right or down increases the threshold, enlarging the selection. When the proposed selection is satisfactory, it can be combined with the current selection by releasing the left mouse button. Subsequent use of the fuzzy tool will start with the final threshold value of the previous use.

Fuzzy selections are well suited to selecting objects of near constant color. Unfortunately, objects of this nature seldom appear in images. When used in conjunction with other tools, however, fuzzy select proves invaluable. It is especially effective in subtracting background pixels from roughly drawn boundaries.

### 3.3.4 Bezier selection tool

Bezier select allows the interactive creation of closed bezier curves. The curves are made up of a series of segments and the segments are in turn made up of a series of points. Clicking the left mouse button when this tool is active places an anchor point. Clicking and dragging places an anchor point and allows the specification of the next control point on the curve which is then placed when the button is released. While moving a control point, the anchor point can be moved by holding down the 'Control' key. The continuity between curve segments can be controlled by using the 'Shift' key. Holding down the 'Shift' key while a control point is being moved causes the continuity between to segments to be disrupted. Releasing the 'Shift' key causes it to be restored. (This allows for the creation of sharp corners on the curve). Control points are displayed as squares and anchor points as circles. The active point (whether it is a control or anchor point) is displayed filled while all other points or unfilled.

The curve is changed into a selection by clicking inside of it. Similar to other selection tools, holding down 'Shift' adds to the current selection while holding down 'Control' subtracts from the current selection.

The bezier selection tool is particularly well suited for selecting curved regions of an image, such as the outline of a teapot. Conversely, it is inadequate for selecting regions which contain a lot of variation.

### 3.3.5 Intelligent selection tool

Intelligent selection or "intelligent scissors" is a computationaly expensive tool that often provess well worth the extra CPU cycles. This tool is specifically for selecting well definined objects in an image. Selections are created by placing control points at appropriate positions along the edges of the desired object. The intelligence of the tool is its ability to discern a path between each set of adjacent control points that corresponds closely to the object's edge. Clicking the left mouse button on the image displays a target cursor specifying where the control point will be placed when the button is released. Dragging the mouse with the button still pressed moves the target cursor as well. Instead of following the mouse cursor exactly, the target cursor "jumps" to the nearest perceived edge in the image. This behavior is useful because good results using this tool are achieved with control points that lie on strong edges. It can be disabled by holding down the 'Shift' key while dragging the mouse.

To finish the selection, specify the last control point as the original control point, which connects the boundary. Control points can be modified by dragging them to new positions. New control points can be inserted by clicking on the curves between any set of existing control points. Clicking inside the connected boundary results in the final selection. As with other tools, holding down 'Shift' adds to the current selection while holding down 'Control' subtracts from the current selection.

Though smarter than most other selection tools, intelligent select is stupid compared to the average human, so it is not surprising that it is prone to foolish errors. By adding additional control points–explicitly defining where the edge of the object is–this tool can be shown the error of its ways. Intelligent select is especially good at selecting objects with strong, well defined edges. When these conditions do not exist, the bezier selection tool is a better choice. Some errors that the tool makes are very difficult to correct by adding additional control points. In these cases, using free selection or fuzzy selection to "touch up" the intelligent selection makes sense.

# 4 The transformation tools

# 5 The painting tools

# 6 Extending the GIMP

Plug-ins are external programs which offer an easy method of extending parts of the GIMP's functionality. Basically, they can be used to operate on images perform such useful tasks as loading and saving images to disk as fell as applying various effects filters. Plug-ins are easy to write, easy to use and a vary nice solution to the problem of bloating the main program with a lot of code that rarely gets used.

## 6.1 How to Use Plug-ins

Plug-ins, as part of their functionality, must be easily accessible from within the GIMP. This is accomplished by putting them within easy reach in the image popup menu which is accessed by clicking the right mouse button within the image. The user need only select the appropriate plug-in from the choices for it to run.

Actually, the above is a simplification of the actual mechanism. The GIMP uses a configuration file that is parsed at startup to determine which plug-ins should be accessible to the user. The configuration file allows the user to specify a hierarchy of menus in which the plug-ins lie. The configuration file also specifies a search path to look through in order to find the actual location of the plug-in binaries.

File plug-ins work slightly differently than other plug-ins. They are accessed semi-invisibly when the user opens or saves a file using the (file/open) or (file/save) commands. The correct file plug-in is determined by examining the extension used to save or open the file. (ie. If a file "temp.jpg" is opened then the "jpeg" file plug-in will be used). The configuration file described above provides a mapping between extensions and file plug-ins. The user can also specify a particular file plug-in to use when opening and saving files which overrides the extension mechanism. This allows a file to be opened or saved which doesn't have an extension.

Many of the plug-ins provide simple dialog interfaces to parameters that affect their operation. These dialogs share the common theme that clicking the "ok" button has the effect of applying the plug-in and clicking the "cancel" button has the effect of canceling the plug-in.

## 6.2 The Standard Plug-ins

The GIMP is distributed with an assortment of plug-ins to extend its functionality. Some of these plug-ins, the file plug-ins for instance, should be viewed as integral parts of the GIMP since using the GIMP would be difficult without them.

Plug-ins are broken into 3 categories. File plug-ins load and save files to disk. Filter plug-ins perform effects on an image. These effects modify the image and are restricted to take place within the current selection. And lastly, channel operations operate on entire images at a single time and perform such operations as duplicate and composite. Below is a list of the plug-ins that come with the GIMP and a short description of their operation.

add - channel operation. Adds together two images by adding corresponding pixels and creating a new image.

bleed - filter. Acts as a low pass filter on the rgb components of an rgb image.

blend - channel operation. Blends together two images by a specified percentage.

blur - filter. Blurs an image using a 3x3 convolution kernel.

compose - channel operation. Composes an rgb image out of three grayscale images. It interpret the grayscale images as the rgb components of the new image or as hsv (hue/saturation/value) components of the new image. Used in conjunction with 'decompose'.

composite - channel operation. Composites three images together and creates a new image. The three images are two source images and a mask image.

decompose - channel operation. Decomposes an rgb image into three grayscale images. The grayscale images can be specified as the rgb channels or the hsv channels of the original image.

difference - channel operation. Takes the absolute value of the difference between two images and creates a new image.

duplicate - channel operation. Duplicates the new image and creates a new image.

edge - filter. Performs edge detection on the input image. The edge detection technique used is to take the Pythagorean sum of two Sobel gradient operators at 90 degrees to each other.

enhance - filter. Enhances edges in the input image.

flip_horz - filter. Flips the image horizontally.

flip_vert - filter. Flips the image vertically.

gamma - filter. Gamma corrects the image.

gbrush - file. Saves or loads an image in the 'gbrush' format used for brushes.

gif - file. Saves or loads an image in the 'gif' format.

grayify - filter. Replaces the rgb components in an rgb image by their grayscale equivalent.

invert - filter. Inverts an image.

jpeg - file. Saves or loads an image in the 'jpeg' format.

multiply - channel operation. Multiplies two images together and creates a new image.

offset - channel operation. Offsets an image by a specified amount and creates a new image.

pixelize - filter.

png - file. Saves or loads an image in the 'png' format.

relief - filter. Performs a Laplacian convolution on the input image. This effect is sometimes known as emboss.

rotate - channel operation. Rotates the input image and creates a new image. Blank areas are filled with the background color.

scale - channel operation. Scales an image and creates a new image.

shift - filter. Displaces each row of pixels by a random user specified amount.

spread - filter. Displaces each pixel by a random user specified amount.

subtract - channel operation. Takes the difference between two images and creates a new image.

tiff - file. Saves or loads an image in the 'tiff' format.

to-color - channel operation. Converts a grayscale or indexed color image to an rgb color image.

to-gray - channel operation. Converts an rgb color or indexed color image to a grayscale image.

to-indexed - channel operation. Converts an rgb color or grayscale image to an indexed color image.

## 6.3 Programing for Plug-ins

Plug-ins are simply ordinary programs that are run by the GIMP. They communicate with the GIMP using a simple message passing system that currently works using a combination of unix pipes and shared memory.

The message passing system allows plug-ins to access services provided by the GIMP, such as creating and displaying an image. It is, however, too low level to be used for everyday programming. The solution is a high level library that provides another layer on top of the message passing system. It is this library and its usage that will be described.

### 6.3.1 Initializing the plug-in library

Initialization of the GIMP library is necessary so the library can setup its communication channels with the GIMP. The GIMP passes in information to be used to setup these channels on the command line when the plug-in is run. All that is needed to initialize the library, however, is to call the `gimp_init` function with the *argc* and *argv* arguments.

`int` **gimp_init** (`int` *argc*, `char **`*argv*) Function

This function initializes the GIMP library. The arguments *argc* and *argv* are simply the *argc* and *argv* values passed into `main` upon invocation.

```
int
main (argc, argv)
     int argc;
     char **argv;
{
  if (gimp_init (argc, argv))
    {
      /* do some stuff */
    }
}
```

### 6.3.2 Getting and operating on images

Several functions are provided by the GIMP library for manipulating images. These functions all lie at the lowest level of operation. They include, getting images and information about images, creating images and displaying images.

Many of the following functions operate on the `Image` data type. This is an opaque data type that contains such information as the images size and active area. This information is intended to be accessed through the appropriate accessor functions.

`void` **gimp_free_image** (`Image` *image*) Function

This function tells the GIMP library to free its link to *image*. The image is not actually destroyed since the GIMP will retain its link.

`void` **gimp_destroy_image** (`Image` *image*) Function

This function tells the GIMP library to free its link to *image* and to tell the GIMP to also free its link. The result is that *image* will be destroyed.

`Image` **gimp_new_image** (`char *`*title*, `long` *width*, `long` *height*, `ImageType` *type*) Function

This function creates a new image with the given title, width, height and type. The new image is returned in the opaque type `Image`. Use the `gimp_image_*` functions to get information out of the image.

*type* should be specified as one of `RGB_IMAGE`, `GRAY_IMAGE` or `INDEXED_IMAGE` to create a new RGB image, grayscale image or indexed color image respectively.

`Image` **gimp_get_input_image** (`long` *id*) Function

`gimp_get_input_image` gets an input image with the given ID. Every image in the main GIMP application has a unique ID value assigned to it at creation. Values are assigned starting from 1. This function can be used to get any valid image in the GIMP. However, this action is not recommended since it is impossible to tell which ID's are valid.

The main usage is to get the default image which is specified by an ID value of 0. (This is used since no actual image in the GIMP will have an ID of 0).

Input images are defined to be read-only. That is, the programmer should not modify the image. (This is enforced by the library by only read-only to the shared memory segment).

`Image` **gimp_get_output_image** (`long` *id*) Function

`gimp_get_output_image` is analogous to `gimp_get_input_image` except that it gets an output image. An output image is defined to be read-write. Meaning that data can be read to and written from. It may seem strange, but getting both the input image and output image for the same ID actually returns two different images and two different data segments. Writing to the input image does not affect the output image and vice-versa.

`void` **gimp_display_image** (`Image` *image*) Function

This function displays an image. A new view is created in the process. Therefore, it is not necessary to display an image that is already displayed. (This function is used mainly by file plug-ins which must tell the GIMP to display a newly created image).

`void` **gimp_update_image** (`Image` *image*) Function

This function tells the GIMP to redraw an image. The image is redrawn in every view.

`char*` **gimp_image_name** (`Image` *image*) Function

This function returns an images name.

`long` **gimp_image_width** (`Image` *image*) Function

This function returns an images width.

`long` **gimp_image_height** (`Image` *image*) Function

This function returns an images height.

`long` **gimp_image_channels** (`Image` *image*) Function

This function returns an images channels. RGB images have 3 channels. Grayscale and indexed color images have 1 channel.

`ImageType` **gimp_image_type** (`Image` *image*) Function

This function returns the type of an image. The type is one of `RGB_IMAGE`, `GRAY_IMAGE`, `INDEXED_IMAGE`, or `UNKNOWN_IMAGE`. If `UNKNOWN_IMAGE` is returned an error has occurred within the library. (ie. This should never happen).

`void` **gimp_image_area** (`Image` *image*, `int *`*x1*, `int *`*y1*, `int *`*x2*, `int *`*y2*) Function

This function returns the active area of operation for a plug-in. The area is returned as a rectangle defined by the upper-left corner (*x1*, *y1*) and the lower-right corner (*x2*, *y2*). This is useful for effects filters which have their results masked by the current selection in an image. By only operating within the active area a slow running plug-in can dramatically increase its speed. (Even a fast running plug-in can dramatically increase its speed since the user may have only selected a small portion of the image).

It should be noted that a plug-in should run the same whether or not it uses the information about the active image area. That is, operating on the entire image and operating on the active area should have the same end effect from the users point of view.

`void*` **gimp_image_data** (`Image` *image*) Function

This function returns the actual pixel data for the image. For RGB images the data is laid out as a series of RGB triplets where each triplet is 3 bytes. For grayscale and indexed color images, each pixel value is a single byte corresponding to a grayscale value or pixel index respectively.

`void*` **gimp_image_cmap** (`Image` *image*) Function

This function returns the colormap associated with an image. It is only valid for indexed color images. The colormap is a series of RGB triplets where each triplet is 3 bytes. The size of the colormap is always 768 bytes, however, only the first `gimp_image_colors (`*image*`)` triplets are valid.

`long` **gimp_image_colors** (`Image` *image*) Function

This function returns the number of colors in an indexed color image. It is only valid for indexed color images.

`void` **gimp_set_image_colors** (`Image` *image*, `void *`*cmap*, `long` *ncolors*) Function

This function sets an images colormap and the number of colors associated with it. It is only valid for indexed color images.

`void` **gimp_foreground_color** (`unsigned char *`*red*`, unsigned char *`*green*`, unsigned char *`*blue*) Function

This function returns the current foreground color in the GIMP application.

`void` **gimp_background_color** (`unsigned char *`*red*`, unsigned char *`*green*`, unsigned char *`*blue*) Function

This function returns the current background color in the GIMP application.

### 6.3.3 Creating and using dialogs

Many plug-ins allow the user to specify various parameters that affect their operation. It is recommended that these paramaters be modifiable through a dialog. There are 2 major methods of creating such a dialog. The first is to use X11 and Motif directly. This method has the advantage that arbitrarily complex dialogs and interfaces can be created. However, it has the disadvantage that the plug-in must be linked with the X11 and Motif libraries. In addition, if the GIMP is ever ported to Windows or the Mac, your plug-in will have to be modified to deal with the new GUI (Graphical User Interface).

The second option is to use a simple dialog interface supported by the GIMP library and the GIMP. It allows the creation of dialogs which have items such as text fields, scales, check buttons, radio buttons, labels and frames. Callbacks can be specified for any active item and the GIMP library will transparently handle receiving such callbacks from the GIMP. It has the disadvantage in that the programmer is somewhat limited in what can be done. However, most (if not all) plug-ins need fairly simple interfaces that are better handled by a simple API (Applications Programmer Interface), than by a complex one such as X11 and Motif.

It is this second option which is used by the standard plug-ins that come with the GIMP. To use the dialog facilities a few items should be explained. First, the dialog is actually created by the main GIMP application. Second, the ID values returned by many of the dialog creation routines are simply integer values used by the library to communicate with the GIMP. An ID value is guaranteed to be unique for any item within a particular dialog and all dialog ID values are guaranteed to be unique. However, items in different dialogs may have the same ID value.

**GimpItemCallbackProc** Data Type

This data type is a function pointer that is used by the GIMP library to handle callbacks. It accepts three arguments. The ID of the item which has initiated the callback, a pointer to user specified client data and a pointer to callback specific call data.

`int` **gimp_new_dialog** (`char *`*title*) Function

`int` **gimp_show_dialog** (`int` *dialog_ID*) Function

`void` **gimp_close_dialog** (`int` *dialog_ID*`, int` *return_val*) Function

`int` **gimp_ok_item_id** (`int` *dialog_ID*) Function

`int` **gimp_cancel_item_id** (`int` *dialog_ID*) Function `int` **gimp_new_row_group** (`int` *dialog_ID*, `int` *parent_ID*, `int` *group_type*, `char` **title*) Function

`int` **gimp_new_column_group** (`int` *dialog_ID*, `int` *parent_ID*, `int` *group_type*, `char` **title*) Function

`int` **gimp_new_push_button** (`int` *dialog_ID*, `int` *parent_ID*, `char` **title*) Function

`int` **gimp_new_check_button** (`int` *dialog_ID*, `int` *parent_ID*, `char` **title*) Function

`int` **gimp_new_radio_button** (`int` *dialog_ID*, `int` *parent_ID*, `char` **title*) Function

`int` **gimp_new_image_menu** (`int` *dialog_ID*, `int` *parent_ID*, `char` *constraint*, `char` **title*) Function

`int` **gimp_new_scale** (`int` *dialog_ID*, `int` *parent_ID*, `long` *min*, `long` *max*, `long` *start_value*, `long` *precision*) Function

`int` **gimp_new_frame** (`int` *dialog_ID*, `int` *parent_ID*, `char` **title*) Function

`int` **gimp_new_label** (`int` *dialog_ID*, `int` *parent_ID*, `char` **title*) Function

`int` **gimp_new_text** (`int` *dialog_ID*, `int` *parent_ID*, `char` **title*) Function

`void` **gimp_change_item** (`int` *dialog_ID*, `int` *item_ID*, `long` *data_size*, `void` **data*) Function

`void` **gimp_show_item** (`int` *dialog_ID*, `int` *item_ID*) Function

`void` **gimp_hide_item** (`int` *dialog_ID*, `int` *item_ID*) Function

`void` **gimp_delete_item** (`int` *dialog_ID*, `int` *item_ID*) Function

`void` **gimp_add_callback** (`int` *dialog_ID*, `int` *item_ID*, `GimpItemCallbackProc` *callback*, `void` **callback_data*) Function

### 6.3.4 Miscellaneous library functions

### 6.3.5 An example of a filter plug-in

### 6.3.6 An example of a file plug-in

### 6.3.7 An example of a channel operation