FreeImage

a free, open source graphics library

Documentation
Library version 3.4.0

Contents

List of tables

List of figures

Introduction

Foreword

Thank you for downloading FreeImage, a free and open source graphics library for Windows, Linux and Mac OS X. FreeImage is widely used and praised for its speed and simplicity. It has been under development for more than 4 years.

FreeImage was created by Floris van den Berg. It was originally developed to provide bitmap loading support to an authoring tool named the Magenta Multimedia Tool. The major parts of the library were designed by Floris, but in its long lifetime, many people have contributed to FreeImage, adding new features and helping to test the library. Without the help of these people, FreeImage wouldn't have been where it is now. Anyone can contribute and post their changes and improvements and have them inserted in the main sources (of course on the condition that developers agree on that the fixes are good). The list of contributors in the FreeImage.h header file is only a small part of all the people that every day provide us with bug reports, suggestions, ideas and source code.

In the middle of the year 2002, Floris stopped the development of the library. Since this date, the FreeImage Project continues to be developed and is maintained by Hervé Drolon.

Purpose of FreeImage

A clear picture about a project is important, because it is that picture that defines which features are implemented and which are not.

FreeImage supports:

- Loading and saving of as many bitmap types as possible
- Easy access to bitmap components, such as palettes and data bits
- Converting bitmap's bit depths from one to another
- Accessing pages in a bitmap when there are multiple, such as in TIFF
- Basic manipulation of bitmaps, such as rotation, flipping and resampling or point operations such as brightness and contrast adjustment
- Alpha compositing and alpha blending

FreeImage does not support:

- Advanced image processing operations such as convolution and transforms
- Bitmap drawing
- Vector graphics

Library reference

Each function name in FreeImage starts with "FreeImage_", for instance FreeImage_Load, FreeImage_Save, FreeImage_Unload ...

A detailed description of each function supported by the FreeImage library is given in the *Function reference* and *Toolkit function reference* chapters. For each entry, the function prototype is shown for C/C++ and the function arguments and explanations are listed. Throughout these chapters, you will see numbers in colored boxes at the top of some functions. These numbers indicate the pixel depth of the input image that the function can operate on. This may be 1-, 4-, 8-, 16-, 24- or 32-bits per pixel. If boxed numbers are not displayed the function operation is independent of the image pixel depth (e.g. for load / save and plugins functions).

Function reference

General functions

The following functions don't have anything to do with the bitmap support provided by FreeImage. They are internal library management functions. That doesn't mean they are not important. Without them you won't be able to load any bitmap at all.

FreeImage_Initialise

```
DLL_API void DLL_CALLCONV FreeImage_Initialise(BOOL load_local_plugins_only
FI_DEFAULT(FALSE));
```

Initialises the library. When the *load_local_plugins_only* parameter is TRUE, FreeImage won't make use of external plugins.

When using the FreeImage DLL, this function is called **automatically** with the load_local_plugins_only parameter set to FALSE. When using FreeImage as a static linked library, you must call this function **exactly once** at the start of your program.

FreeImage_DeInitialise

```
DLL_API void DLL_CALLCONV FreeImage_DeInitialise();
```

Deinitialises the library.

When using the FreeImage DLL, this function is called **automatically**. When using FreeImage as a static linked library, you must call this function **exactly once** at the end of your program to clean up allocated resources in the FreeImage library.

FreeImage_GetVersion

```
DLL_API const char *DLL_CALLCONV FreeImage_GetVersion();
```

Returns a string containing the current version of the DLL.

FreeImage_GetCopyrightMessage

```
DLL_API const char *DLL_CALLCONV FreeImage_GetCopyrightMessage();
```

Returns a string containing a standard copyright message you can show in your program.

FreeImage_SetOutputMessage

```
DLL_API void DLL_CALLCONV FreeImage_SetOutputMessage(FreeImage_OutputMessageFunction
omf);
```

When a certain bitmap cannot be loaded or saved there is usually an explanation for it. For example a certain bitmap format might not be supported due to patent restrictions, or there might be a known issue with a certain bitmap subtype. Whenever something fails in FreeImage internally a log-string is generated, which can be captured by an application driving FreeImage. You use the function FreeImage_SetOutputMessage to capture the log string so that you can show it to the user of the program.

```
/**
FreeImage error handler
@param fif Format / Plugin responsible for the error
@param message Error message
*/
void FreeImageErrorHandler(FREE_IMAGE_FORMAT fif, const char *message) {
  printf("\n*** ");
  printf("%s Format\n", FreeImage_GetFormatFromFIF(fif));
  printf(message);
  printf(" ***\n");
}

// In your main program …

FreeImage_SetOutputMessage(FreeImageErrorHandler);
```

Bitmap management functions

The bitmap management functions in FreeImage are definitely the most used ones. They allow you to allocate new bitmaps, import bitmaps so that they can be edited in memory and export bitmaps to disc. As you will see, the FreeImage bitmap management functions are very easy to use.

Although FreeImage can handle more than 20 bitmap types, there are only 4 bitmap handling functions. A special parameter, an enum named FREE_IMAGE_FORMAT, is used to specify the bitmap format that will be loaded or saved. This enum is defined in the header file FREEIMAGE.H. The following FREE_IMAGE_FORMATS constants are currently available:

FIF	Description
FIF_UNKNOWN	Unknown format (**returned value only, never use it as input value**)
FIF_BMP	Windows or OS/2 Bitmap File (*.BMP)
FIF_CUT	Dr. Halo (*.CUT)
FIF_DDS	DirectDraw Surface (*.DDS)
FIF_GIF	Graphics Interchange Format (*.GIF)
FIF_ICO	Windows Icon (*.ICO)
FIF_IFF	Amiga IFF (*.IFF, *.LBM)
FIF_JNG	JPEG Network Graphics (*.JNG)
FIF_JPEG	Independent JPEG Group (*.JPG)
FIF_KOALA	Commodore 64 Koala format (*.KOA)
FIF_MNG	Multiple Network Graphics (*.MNG)
FIF_PBM	Portable Bitmap (ASCII) (*.PBM)
FIF_PBMRAW	Portable Bitmap (BINARY) (*.PBM)
FIF_PCD	Kodak PhotoCD (*.PCD)
FIF_PCX	PCX bitmap format (*.PCX)
FIF_PGM	Portable Graymap (ASCII) (*.PGM)
FIF_PGMRAW	Portable Graymap (BINARY) (*.PGM)
FIF_PNG	Portable Network Graphics (*.PNG)
FIF_PPM	Portable Pixelmap (ASCII) (*.PPM)
FIF_PPMRAW	Portable Pixelmap (BINARY) (*.PPM)
FIF_PSD	Photoshop (*.PSD)
FIF_RAS	Sun Rasterfile (*.RAS)
FIF_TARGA	Targa files (*.TGA)
FIF_TIFF	Tagged Image File Format (*.TIFF)
FIF_WBMP	Wireless Bitmap (*.WBMP)
FIF_XBM	X11 Bitmap Format (*.XBM)
FIF_XPM	X11 Pixmap Format (*.XPM)

Table 1: FREE_IMAGE_FORMATS constants (FreeImage format identifiers).

As an extension to the FREE_IMAGE_FORMATs, you can register your own bitmap formats. Registering bitmaps can be done manually, by calling one of the plugin management functions (see *Plugin functions*), or automatically by copying a precompiled FreeImage bitmap plugin DLL into the same directory where FREEIMAGE.DLL is residing. When a new bitmap type is registered it is assigned a new, unique plugin identification number that you can pass to the same place that you would pass a FREE_IMAGE_FORMAT.

FreeImage_Allocate

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Allocate(int width, int height, int bpp,
unsigned red_mask FI_DEFAULT(0), unsigned green_mask FI_DEFAULT(0), unsigned blue_mask
FI_DEFAULT(0));
```

If you want to create a new bitmap in memory from scratch, without loading a pre-made bitmap from disc, you use this function. FreeImage_Allocate takes a width and height parameter, and a bpp parameter to specify the bit depth of the image and returns an FIBITMAP. The optional last three parameters (red_mask, green_mask and blue_mask) are used to tell FreeImage the bit-layout of the color components in the bitmap, e.g. where in a pixel the red, green and blue components are stored. To give you an idea about how to interpret the color masks: when red_mask is 0xFF000000 this means that the last 8 bits in one pixel are used for the color red. When green_mask is 0x000000FF, it means that the first 8 bits in a pixel are used for the color green.

FreeImage_Allocate allocates an *empty* bitmap, e.g. a bitmap that is filled completely with zeroes. Zero in a bitmap is usually interpreted as black. This means that if your bitmap is palletised it will contain a completely black palette. You can access, and hence populate the palette by using the function *FreeImage_GetPalette*.

```
FIBITMAP *bitmap = FreeImage_Allocate(320, 240, 32);

if (bitmap) {
    // bitmap successfully created!

    FreeImage_Unload(bitmap);
}
```

FreeImage_Allocate is an alias for FreeImage_AllocateT and can be replaced by this call:
FreeImage_AllocateT(FIT_BITMAP, width, height, bpp, red_mask, green_mask, blue_mask);

FreeImage_AllocateT

1 4 8 16 24 32 64 2x64

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_AllocateT(FREE_IMAGE_TYPE type, int width,
int height, int bpp FI_DEFAULT(8), unsigned red_mask FI_DEFAULT(0), unsigned
green_mask FI_DEFAULT(0), unsigned blue_mask FI_DEFAULT(0));
```

While most imaging applications only deal with photographic images, many scientific applications need to deal with high resolution images (e.g. 16-bit greyscale images), with real valued pixels or even with complex pixels (think for example about the result of a Fast Fourier Transform applied to a 8-bit greyscale image: the result is a complex image).

A special parameter, an enum named FREE_IMAGE_TYPE, is used to specify the bitmap type of a FIBITMAP. This enum is defined in the header file FREEIMAGE.H. The following FREE_IMAGE_TYPE constants are currently available:

FIT	Description
FIT_UNKNOWN	Unknown format (**returned value only, never use it as input value**)
FIT_BITMAP	Standard image: 1-, 4-, 8-, 16-, 24-, 32-bit
FIT_UINT16	Array of unsigned short: unsigned 16-bit
FIT_INT16	Array of short: signed 16-bit
FIT_UINT32	Array of unsigned long: unsigned 32-bit
FIT_INT32	Array of long: signed 32-bit
FIT_FLOAT	Array of float: 32-bit IEEE floating point
FIT_DOUBLE	Array of double: 64-bit IEEE floating point
FIT_COMPLEX	Array of FICOMPLEX: 2 x 64-bit IEEE floating point

Table 2: FREE_IMAGE_TYPE constants (FreeImage data type identifiers).

When you need to know the data type of a bitmap, you can use the *FreeImage_GetImageType* function.

FreeImage_Load

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Load(FREE_IMAGE_FORMAT fif, const char
*filename, int flags FI_DEFAULT(0));
```

This function decodes a bitmap, allocates memory for it and then returns it as a FIBITMAP. The first parameter defines the type of bitmap to be loaded. For example, when FIF_BMP is passed, a BMP file is loaded into memory (an overview of possible FREE_IMAGE_FORMAT constants is available in Table 1). The second parameter tells FreeImage the file it has to decode. The last parameter is used to change the behaviour or enable a feature in the bitmap plugin. Each plugin has its own set of parameters.

```
FIBITMAP *bitmap = FreeImage_Load(FIF_BMP, "mybitmap.bmp", BMP_DEFAULT);

if (bitmap) {
    // bitmap successfully loaded!

    FreeImage_Unload(bitmap);
}
```

Some bitmap loaders can receive parameters to change the loading behaviour. When the parameter is not available or unused you can pass the value 0 or <TYPE_OF_BITMAP>_DEFAULT (e.g. BMP_DEFAULT, ICO_DEFAULT, etc).

Bitmap type	Flag	Description
ICO	ICO_MAKEALPHA	Convert to 32-bit and create an alpha channel from the AND-mask when loading
JPEG	JPEG_DEFAULT	Loads the file as fast as possible, sacrificing some quality
	JPEG_FAST	Loads the file as fast as possible, sacrificing some quality
	JPEG_ACCURATE	Loads the file with the best quality, sacrificing some speed
PCD	PCD_DEFAULT	A PhotoCD picture comes in many sizes. This flag will load the one sized 768 x 512
	PCD_BASE	This flag will load the one sized 768 x 512
	PCD_BASEDIV4	This flag will load the bitmap sized 384 x 256
	PCD_BASEDIV16	This flag will load the bitmap sized 192 x 128
PNG	PNG_IGNOREGAMMA	Avoid gamma correction
TARGA	TARGA_LOAD_RGB888	If set the loader converts RGB555 and ARGB8888 -> RGB888.
TIFF	TIFF_CMYK	This flag will load CMYK bitmaps as 32-bit separated CMYK.

Table 3: Optionnal decoder constants.

FreeImage_LoadFromHandle

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_LoadFromHandle(FREE_IMAGE_FORMAT fif,
FreeImageIO *io, fi_handle handle, int flags FI_DEFAULT(0));
```

FreeImage has the unique feature to load a bitmap from an arbitrary source. This source might for example be a cabinet file, a zip file or an Internet stream. Handling of these arbitrary sources is not directly handled in the FREEIMAGE.DLL, but can be easily added by using a FreeImageIO structure as defined in FREEIMAGE.H.

FreeImageIO is a structure that contains 4 function pointers: one to read from a source, one to write to a source, one to seek in the source and one to tell where in the source we currently are. When you populate the FreeImageIO structure with pointers to functions and pass that structure to FreeImage_LoadFromHandle, FreeImage will call your functions to read, seek and tell in a file. The handle-parameter (third parameter from the left) is used in this to differentiate between different contexts, e.g. different files or different Internet streams.

The function pointers in FreeImageIO use the stdcall calling convention. This means that the functions pointed to must also use the stdcall calling convention. The calling convention was chosen to be compatible with programming language other than C++, such as Visual Basic.

```
FreeImageIO io;
io.read_proc = ReadProc; // pointer to function that calls fread
io.write_proc = NULL;    // not needed for loading
io.seek_proc = SeekProc; // pointer to function that calls fseek
io.tell_proc = TellProc; // pointer to function that calls ftell

FILE *f = fopen("mybitmap.bmp", "rb");

FIBITMAP *bitmap = FreeImage_LoadFromHandle(FIF_BMP, &io, (fi_handle)f, 0);

fclose(f);

if (bitmap) {
    // bitmap successfully loaded!

    FreeImage_Unload(bitmap);
}
```

FreeImage_Save

```
DLL_API BOOL DLL_CALLCONV FreeImage_Save(FREE_IMAGE_FORMAT fif, FIBITMAP *dib, const
char *filename, int flags FI_DEFAULT(0));
```

This function saves a previously loaded FIBITMAP to a file. The first parameter defines the type of the bitmap to be saved. For example, when FIF_BMP is passed, a BMP file is saved (an overview of possible FREE_IMAGE_FORMAT constants is available in Table 1). The second parameter is the name of the bitmap to be saved. If the file already exists it is overwritten. Note that some bitmap save plugins have restrictions on the bitmap types they can save. For example, the JPEG plugin can only save 24 bit and 8 bit greyscale bitmaps*. The last parameter is used to change the behaviour or enable a feature in the bitmap plugin. Each plugin has its own set of parameters.

* In the FreeImage JPEG plugin, 8 bit palletised bitmaps are transparently converted to 24 bit when saving.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'bitmap'

if (FreeImage_Save(FIF_BMP, bitmap, "mybitmap.bmp", 0)) {
    // bitmap successfully saved!
}
```

Some bitmap savers can receive parameters to change the saving behaviour. When the parameter is not available or unused you can pass the value 0 or <TYPE_OF_BITMAP>_DEFAULT (e.g. BMP_DEFAULT, ICO_DEFAULT, etc).

Bitmap type	Flag	Description
BMP	BMP_DEFAULT	Save without any compression
	BMP_SAVE_RLE	Compress the bitmap using RLE when saving
JPEG	JPEG_DEFAULT	Saves with good quality (75:1)
	JPEG_QUALITYSUPERB	Saves with superb quality (100:1)
	JPEG_QUALITYGOOD	Saves with good quality (75:1)
	JPEG_QUALITYNORMAL	Saves with normal quality (50:1)
	JPEG_QUALITYAVERAGE	Saves with average quality (25:1)
	JPEG_QUALITYBAD	Saves with bad quality (10:1)
PBM, PGM, PPM	PNM_DEFAULT	Saves the bitmap as a binary file
	PNM_SAVE_RAW	Saves the bitmap as a binary file
	PNM_SAVE_ASCII	Saves the bitmap as an ASCII file
TIFF	TIFF_DEFAULT	Save using CCITTFAX4 compression for 1-bit bitmaps and LZW compression for any other bitmaps
	TIFF_CMYK	Stores tags for separated CMYK (use | to combine with TIFF compression flags)
	TIFF_PACKBITS	Save using PACKBITS compression.
	TIFF_DEFLATE	Save using DEFLATE compression (also known as ZLIB compression)
	TIFF_ADOBE_DEFLATE	Save using ADOBE DEFLATE compression
	TIFF_NONE	Save without any compression
	TIFF_CCITTFAX3	Save using CCITT Group 3 fax encoding
	TIFF_CCITTFAX4	Save using CCITT Group 4 fax encoding
	TIFF_LZW	Save using LZW compression

Table 4: Optionnal encoder constants.

FreeImage_SaveToHandle

```
DLL_API BOOL DLL_CALLCONV FreeImage_SaveToHandle(FREE_IMAGE_FORMAT fif, FIBITMAP *dib,
FreeImageIO *io, fi_handle handle, int flags FI_DEFAULT(0));
```

The FreeImageIO structure described earlier to load a bitmap from an arbitrary source can also be used to save bitmaps. Once again, FreeImage does not implement the way the bitmap is saved but lets you implement the desired functionality by populating a FreeImageIO structure with pointers to functions. FreeImage will now call *your* functions to write, seek and tell in a stream.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'bitmap'

FreeImageIO io;
io.read_proc = NULL;       // not needed for saving
io.write_proc = WriteProc; // pointer to function that calls fwrite
io.seek_proc = SeekProc;   // pointer to function that calls fseek
io.tell_proc = TellProc;   // pointer to function that calls ftell

FILE *f = fopen("mybitmap.bmp", "wb");

if (FreeImage_SaveToHandle(FIF_BMP, bitmap, &io, (fi_handle)f, 0)) {
    // bitmap successfully saved!
}

fclose(f);
```

FreeImage_Clone

1 4 8 16 24 32 64 2x64

```
DLL_API FIBITMAP * DLL_CALLCONV FreeImage_Clone(FIBITMAP *dib);
```

Makes an exact reproduction of an existing bitmap.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'dib'

FIBITMAP *clone = FreeImage_Clone(dib);

if (clone) {
    // clone succeeded!

    FreeImage_Unload(clone);
}
```

FreeImage_Unload

1 4 8 16 24 32

```
DLL_API void DLL_CALLCONV FreeImage_Unload(FIBITMAP *dib);
```

Deletes a previously loaded FIBITMAP from memory.

You always need to call this function once you're done with a bitmap, or you will have a memory leak.

Bitmap information functions

Once a bitmap is loaded into memory, you can retrieve all kinds of information from it or access specific parts from the bitmap, such as the pixel bits and the palette.

FreeImage_GetImageType

```
DLL_API FREE_IMAGE_TYPE DLL_CALLCONV FreeImage_GetImageType(FIBITMAP *dib);
```

Returns the data type of a bitmap (see Table 2).

FreeImage_GetColorsUsed

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetColorsUsed(FIBITMAP *dib);
```

Returns the number of colors used in a bitmap. This function returns the palette-size for palletised bitmaps, and 0 for high-colour bitmaps.

There has been some criticism on the name of this function. Some users expect this function to return the actual number of colors being used in a bitmap, while the function actually returns the **size of the palette**. The name of this function originates from a member in BITMAPINFOHEADER named biClrUsed. The function actually returns the content of this member.

FreeImage_GetBPP

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetBPP(FIBITMAP *dib);
```

Returns the size of one pixel in the bitmap in bits. For example when each pixel takes 32-bits of space in the bitmap, this function returns 32. Possible bit depths are 1, 4, 8, 16, 24, 32, 64 and 128.

FreeImage_GetWidth

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetWidth(FIBITMAP *dib);
```

Returns the width of the bitmap in pixels.

FreeImage_GetHeight

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetHeight(FIBITMAP *dib);
```

Returns the height of the bitmap in pixels.

FreeImage_GetLine

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetLine(FIBITMAP *dib);
```

Returns the width of the bitmap in bytes.

There has been some criticism on the name of this function. Some people expect it to return a scanline in the pixel data, while it actually returns the width of the bitmap in bytes. As far as I know the term Line is common terminology for the width of a bitmap in bytes. It is at least used by Microsoft DirectX.

FreeImage_GetPitch

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetPitch(FIBITMAP *dib);
```

Returns the width of the bitmap in bytes, rounded to the next 32-bit boundary, also known as pitch or stride or scan width.

In FreeImage each scanline starts at a **32-bit boundary** for performance reasons.

FreeImage_GetDIBSize

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetDIBSize(FIBITMAP *dib);
```

Returns the size of the DIB-element of a FIBITMAP in memory, i.e. the BITMAPINFOHEADER + palette + data bits.

FreeImage_GetPalette

1 4 8 16 24 32

```
DLL_API RGBQUAD *DLL_CALLCONV FreeImage_GetPalette(FIBITMAP *dib);
```

Returns a pointer to the bitmap's palette. If the bitmap doesn't have a palette (i.e. when the pixel bit depth is greater that 8), this function returns NULL.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'dib'
if(FreeImage_GetBPP(dib) == 8) {
  // Build a greyscale palette
  RGBQUAD *pal = FreeImage_GetPalette(dib);
  for (int i = 0; i < 256; i++) {
    pal[i].rgbRed = i;
    pal[i].rgbGreen = i;
    pal[i].rgbBlue = i;
  }
```

FreeImage_GetDotsPerMeterX

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetDotsPerMeterX(FIBITMAP *dib);
```

Returns the horizontal resolution, in pixels-per-meter, of the target device for the bitmap.

FreeImage_GetDotsPerMeterY

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetDotsPerMeterY(FIBITMAP *dib);
```

Returns the vertical resolution, in pixels-per-meter, of the target device for the bitmap.

FreeImage_GetInfoHeader

1 4 8 16 24 32

```
DLL_API BITMAPINFOHEADER *DLL_CALLCONV FreeImage_GetInfoHeader(FIBITMAP *dib);
```

Returns a pointer to the BITMAPINFOHEADER of the DIB-element in a FIBITMAP.

FreeImage_GetInfo

1 4 8 16 24 32

```
DLL_API BITMAPINFO *DLL_CALLCONV FreeImage_GetInfo(FIBITMAP *dib);
```

Alias for FreeImage_GetInfoHeader that returns a pointer to a BITMAPINFO rather than to a BITMAPINFOHEADER.

FreeImage_GetColorType

1 4 8 16 24 32

```
DLL_API FREE_IMAGE_COLOR_TYPE DLL_CALLCONV FreeImage_GetColorType(FIBITMAP *dib);
```

Investigates the color type of the bitmap by reading the bitmap's pixel bits and analysing them. FreeImage_GetColorType can returns one of the following values:

Value	Description
FIC_MINISBLACK	Monochrome bitmap (1-bit) : first palette entry is black. Palletised bitmap (4 or 8-bit) : the bitmap has a greyscale palette
FIC_MINISWHITE	Monochrome bitmap (1-bit) : first palette entry is white. Palletised bitmap (4 or 8-bit) : the bitmap has an inverted greyscale palette
FIC_PALETTE	Palettized bitmap (1, 4 or 8 bit)
FIC_RGB	High-color bitmap (16, 24 or 32 bit)
FIC_RGBALPHA	High-color bitmap with an alpha channel (32 bit only)
FIC_CMYK	CMYK bitmap (32 bit only)

Table 5: FREE_IMAGE_COLOR_TYPE constants.

To be judged greyscale (i.e. FIC_MINISBLACK), a bitmap must have a palette with these characteristics:
- The red, green, and blue values of each palette entry must be equal,
- The interval between adjacent palette entries must be positive and equal to 1.

The CMYK color model (i.e. FIC_CMYK) is the preferred one, if one needs a picture for the print industry or press. In almost every case, this is done by graphic artists: they take a RGB picture (e.g. from a digital camera) and correct the values as appropriate for the picture (single pixel, brightness, contrast...). Finally, they export an CMYK separated image. This will go directly to a layout program and then to the print machines. Most FreeImage users will never need to use CMYK separated images, because the printer drivers will do the conversion job. But in the professional print, the proofed conversion is essential to get a brilliant print result (where no driver will do something like conversion). That's why printed pictures in some magazines look so much better than our home-made prints.

FreeImage_GetRedMask

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetRedMask(FIBITMAP *dib);
```

Returns a bit pattern describing the red color component of a pixel in a FIBITMAP.

FreeImage_GetGreenMask

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetGreenMask(FIBITMAP *dib);
```

Returns a bit pattern describing the green color component of a pixel in a FIBITMAP.

FreeImage_GetBlueMask

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetBlueMask(FIBITMAP *dib);
```

Returns a bit pattern describing the blue color component of a pixel in a FIBITMAP.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'dib'
unsigned red_mask, green_mask, blue_mask;
red_mask = FreeImage_GetRedMask(dib);
green_mask = FreeImage_GetGreenMask(dib);
blue_mask = FreeImage_GetBlueMask(dib);
if(FreeImage_GetBPP(dib) == 16) {
  if (red_mask == 0x1F) && (green_mask == 0x7E0) && (blue_mask == 0xF800)) {
    // We are in RGB16 565 mode
  } else {
    // We are in RGB16 555 mode
  }
}
```

FreeImage_GetTransparencyCount

1 4 8 16 24 32

```
DLL_API unsigned DLL_CALLCONV FreeImage_GetTransparencyCount(FIBITMAP *dib);
```

Returns the number of transparent colors in a palletised bitmap. When the bitmap is not palletised, FreeImage_GetTransparencyCount always returns 0.

FreeImage_GetTransparencyTable

8

```
DLL_API BYTE * DLL_CALLCONV FreeImage_GetTransparencyTable(FIBITMAP *dib);
```

Returns a pointer to the bitmap's transparency table. Only palletised bitmaps have a transparency table. High-color bitmaps store the transparency values directly in the bitmap bits. FreeImage_GetTransparencyTable returns NULL for these bitmaps.

FreeImage_SetTransparencyTable

8

```
DLL_API void DLL_CALLCONV FreeImage_SetTransparencyTable(FIBITMAP *dib, BYTE *table,
int count);
```

Set the bitmap's transparency table. Only palletised bitmaps have a transparency table. High-color bitmaps store the transparency values directly in the bitmap bits. FreeImage_SetTransparencyTable does nothing for these bitmaps.

```
#include "FreeImage.h"

int main(int argc, char* argv[]) {

 FIBITMAP *hDIB24bpp = FreeImage_Load(FIF_BMP, "test.bmp", 0);
 if (hDIB24bpp) {
  // color-quantize 24bpp (results in a 8bpp bitmap to set transparency)
  FIBITMAP *hDIB8bpp = FreeImage_ColorQuantize(hDIB24bpp, FIQ_WUQUANT);
  // get palette and find bright green
  RGBQUAD  *Palette  = FreeImage_GetPalette(hDIB8bpp);
  BYTE      Transparency[256];
  for (unsigned i = 0; i < 256; i++) {
    Transparency[i] = 0xFF;
    if (Palette[i].rgbGreen >= 0xFE &&
      Palette[i].rgbBlue  == 0x00 &&
      Palette[i].rgbRed   == 0x00) {
      Transparency[i] = 0x00;
    }
  }
  // set the tranparency table
  FreeImage_SetTransparencyTable(hDIB8bpp, Transparency, 256);
  // save 8bpp image as transparent PNG
  FreeImage_Save(FIF_PNG, hDIB8bpp, "test.png", 0);

  FreeImage_Unload(hDIB24bpp);
  FreeImage_Unload(hDIB8bpp);
 }
 return 0;
}
```

FreeImage_SetTransparent

8 32

```
DLL_API void DLL_CALLCONV FreeImage_SetTransparent(FIBITMAP *dib, BOOL enabled);
```

Tells FreeImage if it should make use of the transparency table that may accompany a bitmap. When calling this function with a bitmap whose bitdepth is different from 8- or 32-bit, transparency is disabled whatever the value of the Boolean parameter.

FreeImage_IsTransparent

1 4 8 16 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_IsTransparent(FIBITMAP *dib);
```

Returns TRUE when the transparency table is enabled (8-bit images) or when the input dib contains alpha values (32-bit images). Returns FALSE otherwise.

FreeImage_HasBackgroundColor

8 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_HasBackgroundColor(FIBITMAP *dib);
```

Returns TRUE when the image has a file background color, FALSE otherwise.

FreeImage_GetBackgroundColor

8 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_GetBackgroundColor(FIBITMAP *dib, RGBQUAD
*bkcolor);
```

Retrieves the file background color of an image. Returns TRUE if successful, FALSE otherwise. For 8-bit images, the color index in the palette is returned in the rgbReserved member of the bkcolor parameter.

FreeImage_SetBackgroundColor

8 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_SetBackgroundColor(FIBITMAP *dib, RGBQUAD
*bkcolor);
```

Set the file background color of an image. When saving an image to PNG, this background color is transparently saved to the PNG file.

When the bkcolor parameter is NULL, the background color is removed from the image.

Filetype functions

The following functions retrieve the FREE_IMAGE_FORMAT from a bitmap by reading up to 16 bytes and analysing it.

Note that for some bitmap types no FREE_IMAGE_FORMAT can be retrieved. This has to do with the bit-layout of the bitmap-types, which are sometimes not compatible with FreeImage's file-type retrieval system. The unidentifiable formats are: CUT, MNG, PCD, TARGA and WBMP. However, these formats can be identified using the *FreeImage_GetFIFFromFilename* function.

FreeImage_GetFileType

```
DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFileType(const char *filename, int size FI_DEFAULT(0));
```

Orders FreeImage to analyze the bitmap signature. The function then returns one of the predefined FREE_IMAGE_FORMAT constants or a bitmap identification number registered by a plugin. The size parameter is currently not used and can be set to 0.

Because not all formats can be identified by their header (some images don't have a header or one at the end of the file), FreeImage_GetFileType may return FIF_UNKNOWN whereas a plugin is available for the file being analysed. In this case, you can use *FreeImage_GetFIFFromFilename* to guess the file format from the file extension, but this last function is slower and less accurate.

FreeImage_GetFileTypeFromHandle

```
DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFileTypeFromHandle(FreeImageIO *io, fi_handle handle, int size FI_DEFAULT(0));
```

Uses the FreeImageIO structure as described in the topic *Bitmap management functions* to identify a bitmap type. Now the bitmap bits are retrieved from an arbitrary place.

Pixel access functions

The pixel access functions provide you with an easy way to read, write and work pixel-by-pixel with FIBITMAP data.

FreeImage is able to work not only with standard bitmap data (e.g. 1-, 4-, 8-, 16-, 24- and 32-bit) but also with scientific data such as 16-bit greyscale images, or images made up of long, double or complex values (often used in signal and image processing algorithms). An overview of the supported data types is given in Table 2.

In FreeImage, FIBITMAP are based on a coordinate system that is upside down relative to usual graphics conventions. Thus, the scanlines are stored upside down, with the first scan in memory being the bottommost scan in the image.

Bit Formats

In a FIBITMAP the format of the bits are defined by a pixel's bit depth that can be read via a call to *FreeImage_GetBPP*. Possible bit depths include 1-, 4-, 8-, 16-, 24-, 32-, 64- and 128-bit. All formats share the following rules:

- Every scanline is DWORD-aligned. The scanline is buffered to alignment; the buffering is set to 0.
- The scanlines are stored upside down, with the first scan (scan 0) in memory being the bottommost scan in the image.

Each format has the following specifics:

- 1-bit DIBs are stored using each bit as an index into the color table. The most significant bit is the leftmost pixel.
- 4-bit DIBs are stored with each 4 bits representing an index into the color table. The most significant nibble is the leftmost pixel.
- 8-bit DIBs are the easiest to store because each byte is an index into the color table.
- 24-bit DIBs have every 3 bytes representing a color, using the same ordering as the RGBTRIPLE structure.
- 32-bit DIB have every 4 bytes representing a color associated to a alpha value (used to indicate transparency), using the same ordering as the RGBQUAD structure.
- Non standard image types such as short, long, float or double do not have a color table. Pixels are stored in a similar way as 8-bit DIB.
- Complex image types are stored in a similar way as 24- or 32bit DIB, using the same ordering as the FICOMPLEX structure.

Color model

A color model is an abstract mathematical model describing the way colors can be represented as tuples of numbers, typically as three or four values or color components (e.g. RGB and CMYK are color models). FreeImage mainly uses the RGB[A] color model to represent pixels in memory.

However, the pixel layout used by this model is OS dependant. Using a byte by byte memory order to label the pixel layout, then FreeImage uses a BGR[A] pixel layout under a Little Endian processor (Windows, Linux) and uses a RGB[A] pixel layout under a Big Endian processor (Mac OS X or any Big Endian Linux / Unix). This choice was made to ease the use of FreeImage with graphics API.

This subtle difference is however transparent to the user. In order to make pixel access OS independent, FreeImage defines a set of macros used to set or get individual color components in a 24- or 32-bit DIB.

Channel	Pixel position	Associated mask
Red	FI_RGBA_RED	FI_RGBA_RED_MASK
Green	FI_RGBA_GREEN	FI_RGBA_GREEN_MASK
Blue	FI_RGBA_BLUE	FI_RGBA_BLUE_MASK
Alpha	FI_RGBA_ALPHA	FI_RGBA_ALPHA_MASK

Table 6: Pixel access macros and associated masks.

When accessing to individual color components of a 24- or 32-bit DIB, you should always use FreeImage macros or RGBTRIPLE / RGBQUAD structures in order to write OS independent code.

The following sample shows how to use these macros when working with a 32-bit dib:

```
 // Allocate a 32-bit dib
 FIBITMAP *dib = FreeImage_Allocate(512, 512, 32, FI_RGBA_RED_MASK,
FI_RGBA_GREEN_MASK, FI_RGBA_BLUE_MASK);

 // Calculate the number of bytes per pixel (3 for 24-bit or 4 for 32-bit)
 int bytespp = FreeImage_GetLine(dib) / FreeImage_GetWidth(dib);

 for(unsigned y = 0; y < FreeImage_GetHeight(dib); y++) {
   BYTE *bits = FreeImage_GetScanLine(dib, y);

  for(unsigned x = 0; x < FreeImage_GetWidth(dib); x++) {
    // Set pixel color to green with a transparency of 128
    bits[FI_RGBA_RED]   = 0;
    bits[FI_RGBA_GREEN] = 255;
    bits[FI_RGBA_BLUE]  = 0;
    bits[FI_RGBA_ALPHA] = 128;

    // jump to next pixel
    bits += bytespp;
  }
 }
```

FreeImage_GetBits

1 4 8 16 24 32 64 2x64

```
DLL_API BYTE *DLL_CALLCONV FreeImage_GetBits(FIBITMAP *dib);
```

Returns a pointer to the data-bits of the bitmap. It is up to you to interpret these bytes correctly, according to the results of FreeImage_GetBPP, FreeImage_GetRedMask, FreeImage_GetGreenMask and FreeImage_GetBlueMask.

FreeImage_GetScanLine

1 4 8 16 24 32 64 2x64

```
DLL_API BYTE *DLL_CALLCONV FreeImage_GetScanLine(FIBITMAP *dib, int scanline);
```

Returns a pointer to the start of the given scanline in the bitmap's data-bits.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'image'

unsigned x, y;

FREE_IMAGE_TYPE type = FreeImage_GetImageType(image);

// test pixel access
switch(image_type) {
  case FIT_BITMAP:
    if(FreeImage_GetBPP(image) == 8) {
      for(y = 0; y < FreeImage_GetHeight(image); y++) {
        BYTE *bits = (BYTE *)FreeImage_GetScanLine(image, y);
        for(x = 0; x < FreeImage_GetWidth(image); x++) {
          bits[x] = 128;
        }
      }
    }
    break;
  case FIT_UINT16:
    for(y = 0; y < FreeImage_GetHeight(image); y++) {
      unsigned short *bits = (unsigned short *)FreeImage_GetScanLine(image, y);
      for(x = 0; x < FreeImage_GetWidth(image); x++) {
        bits[x] = 128;
      }
    }
    break;
  case FIT_INT16:
    for(y = 0; y < FreeImage_GetHeight(image); y++) {
      short *bits = (short *)FreeImage_GetScanLine(image, y);
      for(x = 0; x < FreeImage_GetWidth(image); x++) {
        bits[x] = 128;
      }
    }
    break;
  case FIT_UINT32:
    for(y = 0; y < FreeImage_GetHeight(image); y++) {
      unsigned long *bits = (unsigned long *)FreeImage_GetScanLine(image, y);
      for(x = 0; x < FreeImage_GetWidth(image); x++) {
        bits[x] = 128;
      }
    }
    break;
  case FIT_INT32:
    for(y = 0; y < FreeImage_GetHeight(image); y++) {
      long *bits = (long *)FreeImage_GetScanLine(image, y);
      for(x = 0; x < FreeImage_GetWidth(image); x++) {
        bits[x] = 128;
      }
    }
    break;
  case FIT_FLOAT:
    for(y = 0; y < FreeImage_GetHeight(image); y++) {
      float *bits = (float *)FreeImage_GetScanLine(image, y);
      for(x = 0; x < FreeImage_GetWidth(image); x++) {
        bits[x] = 128;
      }
    }
    break;
  case FIT_DOUBLE:
    for(y = 0; y < FreeImage_GetHeight(image); y++) {
      double *bits = (double *)FreeImage_GetScanLine(image, y);
      for(x = 0; x < FreeImage_GetWidth(image); x++) {
        bits[x] = 128;
      }
    }
    break;
  case FIT_COMPLEX:
    for(y = 0; y < FreeImage_GetHeight(image); y++) {
      FICOMPLEX *bits = (FICOMPLEX *)FreeImage_GetScanLine(image, y);
      for(x = 0; x < FreeImage_GetWidth(image); x++) {
        bits[x].r = 128;
        bits[x].i = 128;
      }
    }
    break;
}
```

FreeImage_GetPixelIndex

1 4 8

```
DLL_API BOOL DLL_CALLCONV FreeImage_GetPixelIndex(FIBITMAP *dib, unsigned x, unsigned
y, BYTE *value);
```

Get the pixel index of a palettized image at position (x, y), including range check (slow access). Parameter x is the pixel position in horizontal direction, and parameter y is the pixel position in vertical direction. The function returns TRUE on success, and returns FALSE otherwise (e.g. for RGB[A] images).

FreeImage_GetPixelColor

16 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_GetPixelColor(FIBITMAP *dib, unsigned x, unsigned
y, RGBQUAD *value);
```

Get the pixel color of a 16-, 24- or 32-bit image at position (x, y), including range check (slow access). Parameter x is the pixel position in horizontal direction, and parameter y is the pixel position in vertical direction. The function returns TRUE on success, and returns FALSE otherwise (e.g. for palettized images).

FreeImage_SetPixelIndex

1 4 8

```
DLL_API BOOL DLL_CALLCONV FreeImage_SetPixelIndex(FIBITMAP *dib, unsigned x, unsigned
y, BYTE *value);
```

Set the pixel index of a palettized image at position (x, y), including range check (slow access). Parameter x is the pixel position in horizontal direction, and parameter y is the pixel position in vertical direction. The function returns TRUE on success, and returns FALSE otherwise (e.g. for RGB[A] images).

FreeImage_SetPixelColor

16 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_SetPixelColor(FIBITMAP *dib, unsigned x, unsigned
y, RGBQUAD *value);
```

Set the pixel color of a 16-, 24- or 32-bit image at position (x, y), including range check (slow access). Parameter x is the pixel position in horizontal direction, and parameter y is the pixel position in vertical direction. The function returns TRUE on success, and returns FALSE otherwise (e.g. for palettized images).

Conversion functions

The following functions make it possible to convert a bitmap from one bit depth to another.

Under a Little Endian OS (Windows, Linux on PC), bitmaps are always stored in memory as blue first, then green then red, then alpha (BGR[A] convention). Under a Big Endian OS, FreeImage uses the RGB[A] convention. However, these portability considerations are transparently handled by the conversion functions, so that you can later save converted bitmaps in an OS independent manner.

FreeImage_ConvertTo8Bits

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo8Bits(FIBITMAP *dib);
```

Converts a bitmap to 8 bits. If the bitmap was a high-color bitmap (16, 24 or 32-bit) or if it was a monochrome or greyscale bitmap (1 or 4-bit), the end result will be a greyscale bitmap, otherwise (1 or 4-bit palletised bitmaps) it will be a palletised bitmap. A clone of the input bitmap is returned for 8-bit bitmaps.

When creating the greyscale palette, the greyscale intensity of a result pixel is based on red, green, and blue levels of the corresponding source pixel using the following formula:
grey = 0.299 x R + 0.587 x G + 0.114 x B
The values 0.299, 0.587 and 0.114 represent the relative red, green, and blue intensities.

FreeImage_ConvertTo16Bits555

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo16Bits555(FIBITMAP *dib);
```

Converts a bitmap to 16 bits, where each pixel has a color pattern of 5 bits red, 5 bits green and 5 bits blue. One bit in each pixel is unused.

FreeImage_ConvertTo16Bits565

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo16Bits565(FIBITMAP *dib);
```

Converts a bitmap to 16 bits, where each pixel has a color pattern of 5 bits red, 6 bits green and 5 bits blue.

FreeImage_ConvertTo24Bits

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo24Bits(FIBITMAP *dib);
```

Converts a bitmap to 24 bits. A clone of the input bitmap is returned for 24-bit bitmaps.

FreeImage_ConvertTo32Bits

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertTo32Bits(FIBITMAP *dib);
```

Converts a bitmap to 32 bits. A clone of the input bitmap is returned for 32-bit bitmaps.

FreeImage_ColorQuantize

24

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ColorQuantize(FIBITMAP *dib,
FREE_IMAGE_QUANTIZE quantize);
```

Quantizes a high-color 24-bit bitmap to an 8-bit palette color bitmap. The quantize parameter specifies the color reduction algorithm to be used:

Parameter	Quantization method
FIQ_WUQUANT	Xiaolin Wu color quantization algorithm
FIQ_NNQUANT	NeuQuant neural-net quantization algorithm by Anthony Dekker

Table 7: FREE_IMAGE_QUANTIZE constants.

References

Wu, Xiaolin, Efficient Statistical Computations for Optimal Color Quantization. In Graphics Gems, vol. II, p. 126-133. [Online] http://www.csd.uwo.ca/faculty/wu/

Dekker A. H., Kohonen neural networks for optimal color quantization. Network: Computation in Neural Systems, Volume 5, Number 3, Institute of Physics Publishing, 1994. [Online] http://members.ozemail.com.au/~dekker/NEUQUANT.HTML

FreeImage_Threshold

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Threshold(FIBITMAP *dib, BYTE T);
```

Converts a bitmap to 1-bit monochrome bitmap using a threshold T between [0..255]. The function first converts the bitmap to a 8-bit greyscale bitmap. Then, any brightness level that is less than T is set to zero, otherwise to 1. For 1-bit input bitmaps, the function clones the input bitmap and builds a monochrome palette.

FreeImage_Dither

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Dither(FIBITMAP *dib, FREE_IMAGE_DITHER
algorithm);
```

Converts a bitmap to 1-bit monochrome bitmap using a dithering algorithm. For 1-bit input bitmaps, the function clones the input bitmap and builds a monochrome palette.

The algorithm parameter specifies the dithering algorithm to be used. The function first converts the bitmap to a 8-bit greyscale bitmap. Then, the bitmap is dithered using one of the following algorithms:

Parameter	Dithering method
FID_FS	Floyd & Steinberg error diffusion algorithm
FID_BAYER4x4	Bayer ordered dispersed dot dithering (order 2 – 4x4 -dithering matrix)
FID_BAYER8x8	Bayer ordered dispersed dot dithering (order 3 – 8x8 -dithering matrix)
FID_CLUSTER6x6	Ordered clustered dot dithering (order 3 - 6x6 matrix)
FID_CLUSTER8x8	Ordered clustered dot dithering (order 4 - 8x8 matrix)
FID_CLUSTER16x16	Ordered clustered dot dithering (order 8 - 16x16 matrix)

Table 8: FREE_IMAGE_DITHER constants.

References

Ulichney, R., Digital Halftoning. The MIT Press, Cambridge, MA, 1987.

Hawley S., Ordered Dithering. Graphics Gems, Academic Press, 1990.

FreeImage_ConvertFromRawBits

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertFromRawBits(BYTE *bits, int width, int
height, int pitch, unsigned bpp, unsigned red_mask, unsigned green_mask, unsigned
blue_mask, BOOL topdown FI_DEFAULT(FALSE));
```

Converts a raw bitmap somewhere in memory to a FIBITMAP. The parameters in this function are used to describe the raw bitmap. The first parameter is a pointer to the start of the raw bits. The width and height parameter describe the size of the bitmap. The pitch defines the total width of a scanline in the source bitmap, including padding bytes that may be applied. The bpp parameter tells FreeImage what the bit depth of the bitmap is. The red_mask, green_mask and blue_mask parameters tell FreeImage the bit-layout of the color components in the bitmap. The last parameter, topdown, will store the bitmap top-left pixel first when it is TRUE or bottom-left pixel first when it is FALSE.

When the source bitmap uses a 32-bit padding, you can calculate the pitch using the following formula:
int pitch = ((((bpp * width) + 31) / 32) * 4);

FreeImage_ConvertToRawBits

1 4 8 16 24 32

```
DLL_API void DLL_CALLCONV FreeImage_ConvertToRawBits(BYTE *bits, FIBITMAP *dib, int
pitch, unsigned bpp, unsigned red_mask, unsigned green_mask, unsigned blue_mask, BOOL
topdown FI_DEFAULT(FALSE));
```

Converts a FIBITMAP to a raw piece of memory. The layout of the memory is described in the passed parameters, which are the same as in the previous function. The last parameter, topdown, will store the bitmap top-left pixel first when it is TRUE or bottom-left pixel first when it is FALSE.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'dib'

// convert a bitmap to a 32-bit raw buffer (top-left pixel first)
// --------------------------------------------------------------
FIBITMAP *src = FreeImage_ConvertTo32Bits(dib);
FreeImage_Unload(dib);

// Allocate a raw buffer
int width = FreeImage_GetWidth(src);
int height = FreeImage_GetHeight(src);
int scan_width = FreeImage_GetPitch(src);
BYTE *bits = (BYTE*)malloc(height * scan_width);

// convert the bitmap to raw bits (top-left pixel first)
FreeImage_ConvertToRawBits(bits, src, scan_width, 32,
                           FI_RGBA_RED_MASK, FI_RGBA_GREEN_MASK, FI_RGBA_BLUE_MASK, TRUE);
FreeImage_Unload(src);

// convert a 32-bit raw buffer (top-left pixel first) to a FIBITMAP
// ----------------------------------------------------------------
FIBITMAP *dst = FreeImage_ConvertFromRawBits(bits, width, height, scan_width,
                32, FI_RGBA_RED_MASK, FI_RGBA_GREEN_MASK, FI_RGBA_BLUE_MASK, FALSE);
```

FreeImage_ConvertToStandardType

1 4 8 16 24 32 64 2x64

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertToStandardType(FIBITMAP *src, BOOL
scale_linear FI_DEFAULT(TRUE));
```

Converts an image of any type to a standard 8-bit greyscale image. When the scale_linear parameter is TRUE, conversion is done by scaling linearly each pixel value from [min, max] to an integer value between [0..255], where min and max are the minimum and maximum pixel values in the image. When scale_linear is FALSE, conversion is done by rounding each pixel value to an integer between [0..255]. Rounding is done using the following formula:

dst_pixel = (BYTE) MIN(255, MAX(0, q)) where int q = int(src_pixel + 0.5);

The function returns the converted 8-bit greyscale image. For standard images, a clone of the input image is returned.

FreeImage_ConvertToType

1 4 8 16 24 32 64 2x64

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_ConvertToType(FIBITMAP *src, FREE_IMAGE_TYPE
dst_type, BOOL scale_linear FI_DEFAULT(TRUE));
```

Converts an image of any type to type dst_type. When dst_type is equal to FIT_BITMAP, the function calls FreeImage_ConvertToStandardType. Otherwise, conversion is done using standard C language casting convention. When a conversion is not allowed, a NULL value is returned and an error message is thrown (it can be catched using *FreeImage_SetOutputMessage*). The following conversions are currently allowed by the library (other conversions may be added easily if needed):

➔	FIT_BITMAP	FIT_UINT16	FIT_INT16	FIT_UINT32	FIT_INT32	FIT_FLOAT	FIT_DOUBLE	FIT_COMPLEX
FIT_BITMAP	X	X	X	X	X	X	X	X
FIT_UINT16	X	X						X
FIT_INT16	X		X					X
FIT_UINT32	X			X				X
FIT_INT32	X				X			X
FIT_FLOAT	X					X		X
FIT_DOUBLE	X						X	X
FIT_COMPLEX								X

Table 9: Bitmap type conversions allowed by FreeImage.

ICC profile functions

Whenever an ICC profile is available in a bitmap file it is transparently loaded and stored in the FIBITMAP. On the other side, whenever an ICC profile is stored in a FIBITMAP, it is transparently stored in the bitmap file when saving, provided the output FREEIMAGE_FORMAT supports ICC profiles (a plugin can be asked for ICC profile support using *FreeImage_FIFSupportsICCProfiles*).

FreeImage defines a structure called FIICCPROFILE, that is used to access this ICC profile. The structure can then be used with any color management engine to perform bitmap transformations between two ICC profiles.

If the FIICCPROFILE is flagged with FIICC_COLOR_IS_CMYK the bitmap is a representation of a CMYK separation. Together with color management this information is important, because the profile data and the bitmap must reside in the same color model (e.g. RGB or CMYK).

In almost all cases, the bitmap is loaded as an RGB representation. It may depend on special flags to FreeImage_Load, whether the original color representation is preserved or not.

```
// load a bitmap from file, enforce to preserve the
// CMYK separated data from TIFF (no RGB conversion done)

FIBITMAP *bitmap = FreeImage_Load (FIF_TIFF, name, TIFF_CMYK);

if (bitmap) {

// test for RGB or CMYK colour space

  if ((FreeImage_GetICCProfile(bitmap)->flags &
       FIICC_COLOR_IS_CMYK) == FIICC_COLOR_IS_CMYK)

    // we are in CMYK colour space

  else
    // we are in RGB colour space

}
```

FreeImage_GetICCProfile

1 4 8 16 24 32

```
DLL_API FIICCPROFILE *DLL_CALLCONV FreeImage_GetICCProfile(FIBITMAP *dib);
```

Retrieves a pointer to the FIICCPROFILE data of the bitmap. This function can also be called safely, when the original format does not support profiles.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'bitmap'

// retrieve a pointer to FIICCPROFILE structure

FIICCPROFILE *profile = FreeImage_GetICCProfile(bitmap);

If (profile->data) {

  // profile data present

}
```

FreeImage_CreateICCProfile

1 4 8 16 24 32

```
DLL_API FIICCPROFILE *DLL_CALLCONV FreeImage_CreateICCProfile(FIBITMAP *dib, void
*data, long size);
```

Creates a new FIICCPROFILE block from ICC profile data previously read from a file or built by a color management system. The profile data are attached to the bitmap. The function returns a pointer to the FIICCPROFILE structure created.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'bitmap'

DWORD size = _filelength(fileno(hProfile));

// read profile data from file and zero-terminate

if (size && (data = (void *)malloc(size + 1))) {
  size = fread(data, 1, size, hProfile);
  *(data + size) = 0;

  // attach retrieved profile data to bitmap

  FIICCPROFILE *profile = FreeImage_CreateICCProfile (bitmap, data, size);

  free (data);
}
```

FreeImage_DestroyICCProfile

1 4 8 16 24 32

```
DLL_API void DLL_CALLCONV FreeImage_DestroyICCProfile(FIBITMAP *dib);
```

This function destroys an FIICCPROFILE previously created by FreeImage_CreateICCProfile. After this call the bitmap will contain no profile information. This function should be called to ensure that a stored bitmap will not contain any profile information.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'bitmap'

// destroy profile possibly present

FreeImage_DestroyICCProfile(bitmap);

// store profile-less bitmap

FreeImage_Save (FIF_TIFF, bitmap, name, flags);
```

Plugin functions

Through average use you won't probably notice it, FreeImage is plugin driven. Each bitmap loader/saver is in fact a plugin module that is linked inside the integrated plugin manager. You won't notice it, until you decide to write your own plugins.

Almost every plugin in FreeImage is incorporated directly into the DLL. The reason why this is done this way is a mixture of evolution and design. The first versions of FreeImage (actually, about the whole first year of its existence) it had no notion of plugins. This meant that all bitmap functionality was available only from the main DLL. In the second year Floris decided to create plugins, because he wanted to support some bitmaps formats that have license restrictions on them, such as GIF. In fear that he would put all its bitmap loaders/savers in tiny DLLs that would splatter the hard drive, his most important 'customer' strongly encouraged him to keep as much bitmap formats in one DLL as possible. He took his word for it and it lead to the design you see here today.

The actual plugin system evolved from something very simple to a very flexible mechanism that he now often reuses in other software. At this moment it's possible to have plugins in the main FREEIMAGE.DLL, in external DLLs, and even directly in an application that drives FreeImage.

FreeImage_GetFIFCount

```
DLL_API int DLL_CALLCONV FreeImage_GetFIFCount();
```

Retrieves the number of FREE_IMAGE_FORMAT identifiers being currently registered. In FreeImage FREE_IMAGE_FORMAT became, through evolution, synonymous with plugin.

FreeImage_SetPluginEnabled

```
DLL_API int DLL_CALLCONV FreeImage_SetPluginEnabled(FREE_IMAGE_FORMAT fif, BOOL
enable);
```

Enables or disables a plugin. A disabled plugin cannot be used to import and export bitmaps, nor will it identify bitmaps. When called, this function returns the previous plugin state (TRUE / 1 or FALSE / 0), or –1 if the plugin doesn't exist.

FreeImage_IsPluginEnabled

```
DLL_API int DLL_CALLCONV FreeImage_IsPluginEnabled(FREE_IMAGE_FORMAT fif);
```

Returns TRUE when the plugin is enabled, FALSE when the plugin is disabled, -1 otherwise.

FreeImage_GetFIFFromFormat

```
DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFIFFromFormat(const char *format);
```

Returns a FREE_IMAGE_FORMAT identifier from the format string that was used to register the FIF.

FreeImage_GetFIFFromMime

```
DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFIFFromMime(const char *mime);
```

Returns a FREE_IMAGE_FORMAT identifier from a MIME content type string (MIME stands for Multipurpose Internet Mail Extension).

```
FREE_IMAGE_FORMAT fif = FreeImage_GetFIFFromMime("image/png");
If(fif != FIF_UNKNOWN) {
  assert(fif == FIF_PNG);
}
```

FreeImage_GetFormatFromFIF

```
DLL_API const char *DLL_CALLCONV FreeImage_GetFormatFromFIF(FREE_IMAGE_FORMAT fif);
```

Returns the string that was used to register a plugin from the system assigned FREE_IMAGE_FORMAT.

FreeImage_GetFIFExtensionList

```
DLL_API const char *DLL_CALLCONV FreeImage_GetFIFExtensionList(FREE_IMAGE_FORMAT fif);
```

Returns a comma-delimited file extension list describing the bitmap formats the given plugin can read and/or write.

```
/**
Builds a series of string pairs that specify filters you can apply to load a file.
The filter string is to be used by a 'File Open' dialog box
(GetOpenFileName or CFileDialog).
@param szFilter Input and output parameter. szFilter is an array of char whose length
should be 1024 or more.
@return Returns the number of supported import formats
*/
int GetOpenFilterString(char *szFilter) {
  int i, iCount;
  char Filter[1024];
  char *token;

  // Build a string for 'All image files'
  Filter[0] = '\0';
  for(i = 0; i < FreeImage_GetFIFCount(); i++) {
    if(FreeImage_FIFSupportsReading((FREE_IMAGE_FORMAT)i)) {
      strcat(Filter, FreeImage_GetFIFExtensionList((FREE_IMAGE_FORMAT)i));
      strcat(Filter, ",");
    }
  }
  Filter[strlen(Filter)-1] = '\0';
  strcpy(szFilter, "All image files|");
  token = strtok(Filter, ",");
  while(token != NULL) {
    strcat(szFilter, "*.");
    strcat(szFilter, token);
    strcat(szFilter, ";");
    // get next token
    token = strtok(NULL, ",");
  }
  szFilter[strlen(szFilter)-1] = '|';

  // Build a string for 'All files'
  strcat(szFilter, "All Files (*.*)|*.*|");

  // Build a string for each format
  Filter[0] = '\0';
  iCount = 0;
  for(i = 0; i < FreeImage_GetFIFCount(); i++) {
    if(FreeImage_FIFSupportsReading((FREE_IMAGE_FORMAT)i)) {
      // Description
      sprintf(Filter, "%s (%s)|", FreeImage_GetFIFDescription((FREE_IMAGE_FORMAT)i),
      FreeImage_GetFIFExtensionList((FREE_IMAGE_FORMAT)i));
      strcat(szFilter, Filter);
      // Extension(s)
      strcpy(Filter, FreeImage_GetFIFExtensionList((FREE_IMAGE_FORMAT)i));
      token = strtok(Filter, ",");
      while(token != NULL) {
        strcat(szFilter, "*.");
        strcat(szFilter, token);
        strcat(szFilter, ";");
        // get next token
        token = strtok(NULL, ",");
      }
      szFilter[strlen(szFilter)-1] = '|';
      iCount++;
    }
  }
  strcat(szFilter, "|");

  return iCount;
}
```

FreeImage_GetFIFDescription

```
DLL_API const char *DLL_CALLCONV FreeImage_GetFIFDescription(FREE_IMAGE_FORMAT fif);
```

Returns a descriptive string that describes the bitmap formats the given plugin can read and/or write.

FreeImage_GetFIFRegExpr

```
DLL_API const char * DLL_CALLCONV FreeImage_GetFIFRegExpr(FREE_IMAGE_FORMAT fif);
```

Returns a regular expression string that can be used by a regular expression engine to identify the bitmap. FreeImageQt makes use of this function.

FreeImage_GetFIFFromFilename

```
DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_GetFIFFromFilename(const char
*filename);
```

This function takes a filename or a file-extension and returns the plugin that can read/write files with that extension in the form of a FREE_IMAGE_FORMAT identifier.

```
/** Generic image loader
@param lpszPathName Pointer to the full file name
@param flag Optional load flag constant
@return Returns the loaded dib if successful, returns NULL otherwise
*/
FIBITMAP* GenericLoader(const char* lpszPathName, int flag) {
  FREE_IMAGE_FORMAT fif = FIF_UNKNOWN;
  // check the file signature and deduce its format
  // (the second argument is currently not used by FreeImage)
  fif = FreeImage_GetFileType(lpszPathName, 0);
  if(fif == FIF_UNKNOWN) {
    // no signature ?
    // try to guess the file format from the file extension
    fif = FreeImage_GetFIFFromFilename(lpszPathName);
  }
  // check that the plugin has reading capabilities ...
  if((fif != FIF_UNKNOWN) && FreeImage_FIFSupportsReading(fif)) {
    // ok, let's load the file
    FIBITMAP *dib = FreeImage_Load(fif, lpszPathName, flag);
    // unless a bad file format, we are done !
    return dib;
  }
  return NULL;
}
```

FreeImage_FIFSupportsReading

```
DLL_API BOOL DLL_CALLCONV FreeImage_FIFSupportsReading(FREE_IMAGE_FORMAT fif);
```

Returns TRUE if the plugin belonging to the given FREE_IMAGE_FORMAT can be used to load bitmaps, FALSE otherwise.

FreeImage_FIFSupportsWriting

```
DLL_API BOOL DLL_CALLCONV FreeImage_FIFSupportsWriting(FREE_IMAGE_FORMAT fif);
```

Returns TRUE if the plugin belonging to the given FREE_IMAGE_FORMAT can be used to save bitmaps, FALSE otherwise.

```
/** Generic image writer
@param dib Pointer to the dib to be saved
@param lpszPathName Pointer to the full file name
@param flag Optional save flag constant
@return Returns true if successful, returns false otherwise
*/
bool GenericWriter(FIBITMAP* dib, const char* lpszPathName, int flag) {
  FREE_IMAGE_FORMAT fif = FIF_UNKNOWN;
  BOOL bSuccess = FALSE;
  if(dib) {
    // try to guess the file format from the file extension
    fif = FreeImage_GetFIFFromFilename(lpszPathName);
    if(fif != FIF_UNKNOWN ) {
      // check that the plugin has sufficient writing
      // and export capabilities ...
      WORD bpp = FreeImage_GetBPP(dib);
      if(FreeImage_FIFSupportsWriting(fif) &&
         FreeImage_FIFSupportsExportBPP(fif, bpp)) {
        // ok, we can save the file
        bSuccess = FreeImage_Save(fif, dib, lpszPathName, flag);
        // unless an abnormal bug, we are done !
      }
    }
  }
  return (bSuccess == TRUE) ? true : false;
}
```

FreeImage_FIFSupportsExportType

```
DLL_API BOOL DLL_CALLCONV FreeImage_FIFSupportsExportType(FREE_IMAGE_FORMAT fif,
FREE_IMAGE_TYPE type);
```

Returns TRUE if the plugin belonging to the given FREE_IMAGE_FORMAT can save a bitmap in the desired data type, returns FALSE otherwise. Currently, TIFF is the only plugin able to save *all* non-standard images. The PNG plugin is able to save unsigned 16-bit images.

FreeImage_FIFSupportsExportBPP

```
DLL_API BOOL DLL_CALLCONV FreeImage_FIFSupportsExportBPP(FREE_IMAGE_FORMAT fif, int
bpp);
```

Returns TRUE if the plugin belonging to the given FREE_IMAGE_FORMAT can save a bitmap in the desired bit depth, returns FALSE otherwise.

```
/**
Builds a series of string pairs that specify filters you can apply to save a file.
The filter string is to be used by a 'File Save As' dialog box
(GetSaveFileName or CFileDialog).
@param szFilter Input and output parameters. szFilter is an array of char whose length
should be 1024 or more.
@param bpp The bit depth of the image to be saved.
@return Returns the number of supported export formats
*/
int GetSaveAsFilterString(char *szFilter, WORD bpp) {
  int i, iCount;
  char Filter[1024];
  char *token;
  szFilter[0] = '\0';
  iCount = 0;
  // Build a string for each format
  for(i = 0; i < FreeImage_GetFIFCount(); i++) {
    if(FreeImage_FIFSupportsExportBPP((FREE_IMAGE_FORMAT)i, bpp)) {
      // Handle the special case of PNM files
      strcpy(Filter, FreeImage_GetFormatFromFIF((FREE_IMAGE_FORMAT)i));
      if((bpp == 1) && (!strncmp(Filter, "PGM", 3) || !strncmp(Filter, "PPM", 3)))
        continue;
      if((bpp == 8) && (!strncmp(Filter, "PBM", 3) || !strncmp(Filter, "PPM", 3)))
        continue;
      if((bpp == 24) && (!strncmp(Filter, "PGM", 3) || !strncmp(Filter, "PBM", 3)))
        continue;
      // Description
      sprintf(Filter, "%s (%s)|", FreeImage_GetFIFDescription((FREE_IMAGE_FORMAT)i),
      FreeImage_GetFIFExtensionList((FREE_IMAGE_FORMAT)i));
      strcat(szFilter, Filter);
      // Extension(s)
      strcpy(Filter, FreeImage_GetFIFExtensionList((FREE_IMAGE_FORMAT)i));
      token = strtok(Filter, ",");
      while(token != NULL) {
        strcat(szFilter, "*.");
        strcat(szFilter, token);
        strcat(szFilter, ";");
        // get next token
        token = strtok(NULL, ",");
      }
      szFilter[strlen(szFilter)-1] = '|';
      iCount++;
    }
  }
  strcat(szFilter, "|");
  return iCount;
}
```

FreeImage_FIFSupportsICCProfiles

```
DLL_API BOOL DLL_CALLCONV FreeImage_FIFSupportsICCProfiles(FREE_IMAGE_FORMAT fif);
```

Returns TRUE if the plugin belonging to the given FREE_IMAGE_FORMAT can load or save an ICC profile, FALSE otherwise.

```
// determine, whether profile support is present

if (FreeImage_FIFSupportsICCProfiles(FIF_TIFF)) {

  // profile support present

}
```

FreeImage_RegisterLocalPlugin

```
DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_RegisterLocalPlugin(FI_InitProc
proc_address, const char *format FI_DEFAULT(0), const char *description FI_DEFAULT(0),
const char *extension FI_DEFAULT(0), const char *regexpr FI_DEFAULT(0));
```

Registers a new plugin to be used in FreeImage. The plugin is residing directly in the application driving FreeImage. The first parameter is a pointer to a function that is used to

initialise the plugin. The initialization function is responsible for filling in a Plugin structure and storing a system-assigned format identification number used for message logging.

```
static int s_format_id;

void stdcall
Init(Plugin *plugin, int format_id) {
    s_format_id = format_id;

    // pointer to a function that returns a type-string
    // for the bitmap. For example, a plugin that loads
    // BMPs returns the string "BMP".

    plugin->format_proc = Format;

    // pointer to a function that returns a descriptive
    // string for the bitmap type. For example, a plugin
    // that loads BMPs may return "Windows or OS/2 Bitmap"

    plugin->description_proc = Description;

    // pointer to a function that returns a comma delimited
    // list of possible file extension that are valid for
    // this plugin. A JPEG plugin would return "jpeg,jif,jfif"

    plugin->extension_proc = Extension;

    // pointer to a function that is used to load the bitmap

    plugin->load_proc = Load;

    // pointer to a function that is used to save the bitmap

    plugin->save_proc = Save;

    // pointer to a function that will try to identify a
    // bitmap by looking at the first few bytes of the bitmap.

    plugin->validate_proc = Validate;
}
```

FreeImage_RegisterExternalPlugin

```
DLL_API FREE_IMAGE_FORMAT DLL_CALLCONV FreeImage_RegisterExternalPlugin(const char
*path, const char *format FI_DEFAULT(0), const char *description FI_DEFAULT(0), const
char *extension FI_DEFAULT(0), const char *regexpr FI_DEFAULT(0));
```

Registers a new plugin to be used in FreeImage. The plugin is residing in a DLL. Functionally this function is the same as FreeImage_RegisterLocalPlugin, but now FreeImage calls an Init function in a DLL instead of a local function in an application. The Init function must be called "Init" and must use the stdcall calling convention.

Multipage functions

FreeImage features a set of functions that can be used to manipulate pages in a multi-page bitmap format. Currently TIFF and ICO formats are supported for this. The multi-page API makes it possible to access and change pages in a multi-bitmap, delete pages and change the order of pages. All of this is offered with a minimum implementation in a plugin and low requirement of memory through a sophisticated, compressing cache mechanism.

In the multipage API, whenever a 'page' parameter is needed by a function, it is always 0-based.

FreeImage_OpenMultiBitmap

```
DLL_API FIMULTIBITMAP * DLL_CALLCONV FreeImage_OpenMultiBitmap(FREE_IMAGE_FORMAT fif,
const char *filename, BOOL create_new, BOOL read_only, BOOL keep_cache_in_memory
FI_DEFAULT(FALSE));
```

Opens a multi-paged bitmap.

The first parameter tells FreeImage the bitmap-type of bitmap to be opened. Currently FIF_TIFF and FIF_ICO are supported. The second parameter specifies the name of the bitmap. When the third parameter is TRUE, it means that a new bitmap will be created rather than an existing one being opened. When the fourth parameter is TRUE the bitmap is opened read-only. The last parameter is one purely for performance. When it is TRUE, all gathered bitmap data in the page manipulation process is kept in memory, otherwise it is lazily flushed to a temporary file on the hard disk in 64 Kb blocks. Note that depending on the amount of manipulation being performed and the size of the bitmap, the temporary data can become quite large. It's advised to lazily flush to disc.

FreeImage_CloseMultiBitmap

```
DLL_API BOOL DLL_CALLCONV FreeImage_CloseMultiBitmap(FIMULTIBITMAP *bitmap, int flags
FI_DEFAULT(0));
```

Closes a previously opened multi-page bitmap and, when the bitmap was not opened read-only, applies any changes made to it.

The flags parameter is used to change the behaviour or enable a feature in the bitmap plugin. Each plugin has its own set of parameters (see Table 4). Some bitmap savers can receive parameters to change the saving behaviour. When the parameter is not available or unused you can pass the value 0 or <TYPE_OF_BITMAP>_DEFAULT (e.g. TIFF_DEFAULT, ICO_DEFAULT, etc).

FreeImage_GetPageCount

```
DLL_API int DLL_CALLCONV FreeImage_GetPageCount(FIMULTIBITMAP *bitmap);
```

Returns the number of pages currently available in the multi-paged bitmap.

FreeImage_AppendPage

```
DLL_API void DLL_CALLCONV FreeImage_AppendPage(FIMULTIBITMAP *bitmap, FIBITMAP *data);
```

Appends a new page to the end of the bitmap.

FreeImage_InsertPage

```
DLL_API void DLL_CALLCONV FreeImage_InsertPage(FIMULTIBITMAP *bitmap, int page,
FIBITMAP *data);
```

Inserts a new page before the given position in the bitmap. Page has to be a number smaller than the current number of pages available in the bitmap.

FreeImage_DeletePage

```
DLL_API void DLL_CALLCONV FreeImage_DeletePage(FIMULTIBITMAP *bitmap, int page);
```

Deletes the page on the given position.

FreeImage_LockPage

```
DLL_API FIBITMAP * DLL_CALLCONV FreeImage_LockPage(FIMULTIBITMAP *bitmap, int page);
```

Locks a page in memory for editing. The page can now be saved to a different file or inserted into another multi-page bitmap. When you are done with the bitmap you have to call FreeImage_UnlockPage to give the page back to the bitmap and/or apply any changes made in the page.

It is forbidden to use FreeImage_Unload on a locked page.

FreeImage_UnlockPage

```
DLL_API void DLL_CALLCONV FreeImage_UnlockPage(FIMULTIBITMAP *bitmap, FIBITMAP *page,
BOOL changed);
```

Unlocks a previously locked page and gives it back to the multi-page engine. When the last parameter is TRUE, the page is marked changed and the new page data is applied in the multi-page bitmap.

FreeImage_MovePage

```
DLL_API BOOL DLL_CALLCONV FreeImage_MovePage(FIMULTIBITMAP *bitmap, int target, int
source);
```

Moves the source page to the position of the target page. Returns TRUE on success, FALSE on failure.

FreeImage_GetLockedPageNumbers

```
DLL_API BOOL DLL_CALLCONV FreeImage_GetLockedPageNumbers(FIMULTIBITMAP *bitmap, int
*pages, int *count);
```

Returns an array of page-numbers that are currently locked in memory. When the pages parameter is NULL, the size of the array is returned in the count variable. You can then allocate the array of the desired size and call FreeImage_GetLockedPageNumbers again to populate the array.

Compression functions

FreeImage uses many Open Source third party libraries in order to load or save complex image file formats. Among these libraries, some of them, such as the ZLib library, deal with compression / decompression of memory buffers. Since this feature may be useful in many applications and not only for image compression, FreeImage provides an interface to the main functionalities of these libraries.

Currently, only ZLib compression is supported. Other compression algorithms may be added with future releases of FreeImage.

FreeImage_ZlibCompress

```
DLL_API DWORD DLL_CALLCONV FreeImage_ZLibCompress(BYTE *target, DWORD target_size,
BYTE *source, DWORD source_size);
```

Compresses a source buffer into a target buffer, using the ZLib library. Upon entry, target_size is the total size of the destination buffer, which must be at least 0.1% larger than source_size plus 12 bytes. The function returns the actual size of the compressed buffer, or returns 0 if an error occurred.

```
BYTE *data = NULL;
DWORD original_size = 0;
// ...
data = (BYTE*)malloc(original_size * sizeof(BYTE));
// ...

// compress data
DWORD compressed_size = (DWORD)((double) original_size + (0.1 * (double)
original_size) + 12);
BYTE *compressed_data = (BYTE*)malloc(compressed_size * sizeof(BYTE));
compressed_size = FreeImage_ZLibCompress(compressed_data, compressed_size, data,
original_size);

// write data to disk
fwrite(&original_size, sizeof(DWORD), 1, stream);
fwrite(&compressed_size, sizeof(DWORD), 1, stream);
fwrite(compressed_data, sizeof(BYTE), compressed_size, stream);

free(compressed_data);
```

FreeImage_ZlibUncompress

```
DLL_API DWORD DLL_CALLCONV FreeImage_ZLibUncompress(BYTE *target, DWORD target_size,
BYTE *source, DWORD source_size);
```

Decompresses a source buffer into a target buffer, using the ZLib library. Upon entry, target_size is the total size of the destination buffer, which must be large enough to hold the entire uncompressed data. The size of the uncompressed data must have been saved previously by the compressor and transmitted to the decompressor by some mechanism outside the scope of this compression library. The function returns the actual size of the uncompressed buffer, or returns 0 if an error occurred.

```
BYTE *data = NULL;
DWORD original_size = 0, compressed_size = 0;
// ...
// read data from disk
fread(&original_size, sizeof(DWORD), 1, stream);
fread(&compressed_size, sizeof(DWORD), 1, stream);
data = (BYTE*)malloc(original_size * sizeof(BYTE));
compressed_data = (BYTE*)malloc(compressed_size * sizeof(BYTE));
fread(compressed_data, sizeof(BYTE), compressed_size, stream);

// decompress data
DWORD size = 0;
size = FreeImage_ZLibUncompress(data, original_size, compressed_data,
compressed_size);
assert(size == original_size);
free(compressed_data);
```

Helper functions

FreeImage_IsLittleEndian

```
DLL_API BOOL DLL_CALLCONV FreeImage_IsLittleEndian();
```

This function returns TRUE if the platform running FreeImage uses the Little Endian convention (Intel processors) and returns FALSE if it uses the Big Endian convention (Motorola processors).

FreeImage_LookupX11Color

```
DLL_API BOOL DLL_CALLCONV FreeImage_LookupX11Color(const char *szColor, BYTE *nRed,
BYTE *nGreen, BYTE *nBlue);
```

Converts a X11 color name into a corresponding RGB value. Upon entry, szColor is the color name. On output, nRed, nGreen and nBlue are the color components in the range [0..255]. The function returns TRUE if successful, FALSE otherwise.

```
BYTE red, green, blue;
BOOL bResult;
bResult = FreeImage_LookupX11Color("papaya whip", &red, &green, &blue);
if(bResult) {
  assert((red == 255) && (green == 239) && (blue == 213));
}
```

FreeImage_LookupSVGColor

```
DLL_API BOOL DLL_CALLCONV FreeImage_LookupSVGColor(const char *szColor, BYTE *nRed,
BYTE *nGreen, BYTE *nBlue);
```

Converts a SVG color name into a corresponding RGB value. Upon entry, szColor is the color name. On output, nRed, nGreen and nBlue are the color components in the range [0..255]. The function returns TRUE if successful, FALSE otherwise.

Toolkit function reference

Rotation and flipping

FreeImage_RotateClassic

8 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_RotateClassic(FIBITMAP *dib, double angle);
```

This function rotates an 8-bit greyscale, 24- or 32-bit image by means of 3 shears. The angle of rotation is specified by the angle parameter in degrees. Rotation occurs around the center of the image area. Rotated image retains size and aspect ratio of source image (destination image size is usually bigger), so that this function should be used when rotating an image by 90°, 180° or 270°.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'dib'

// perform a 90° rotation (CCW rotation)
FIBITMAP *rotated = FreeImage_RotateClassic(dib, 90);
```

A demonstration of this function is given in the Appendix (see *Using the rotation functions*).

References

Paeth A., A Fast Algorithm for General Raster Rotation. Graphics Gems, p. 179, Andrew Glassner editor, Academic Press, 1990.

Yariv E., High quality image rotation (rotate by shear).
[Online] http://www.codeproject.com/bitmap/rotatebyshear.asp

FreeImage_RotateEx

8 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_RotateEx(FIBITMAP *dib, double angle, double
x_shift, double y_shift, double x_origin, double y_origin, BOOL use_mask);
```

This function performs a rotation and / or translation of an 8-bit greyscale, 24- or 32-bit image, using a 3rd order (cubic) B-Spline. The rotated image will have the same width and height as the source image, so that this function is better suited for computer vision and robotics.

The angle of rotation is specified by the angle parameter in degrees. Horizontal and vertical image translations (in pixel units) are specified by the x_shift and y_shift parameters. Rotation occurs around the center specified by x_origin and y_origin, also given in pixel units. When use_mask is set to TRUE, the irrelevant part of the image is set to a black color, otherwise, a mirroring technique is used to fill irrelevant pixels.

```
// this code assumes there is a bitmap loaded and
// present in a variable called 'dib'

// rotate the image about the center of the image area

double x_orig = FreeImage_GetWidth(dib) / (double)2;
double y_orig = FreeImage_GetHeight(dib) / (double)2;

// perform a 15° CCW rotation using a mask (no translation)
FIBITMAP *rotated = FreeImage_RotateEx(dib, 15, 0, 0, x_orig, y_orig, TRUE);
```

A demonstration of this function is given in the Appendix (see *Using the rotation functions*).

References

Philippe Thévenaz, Spline interpolation, a C source code implementation. [Online] http://bigwww.epfl.ch/thevenaz/

Unser M., Splines: A Perfect Fit for Signal and Image Processing. IEEE Signal Processing Magazine, vol. 16, no. 6, pp. 22-38, November 1999.

Unser M., Aldroubi A., Eden M., B-Spline Signal Processing: Part I--Theory. IEEE Transactions on Signal Processing, vol. 41, no. 2, pp. 821-832, February 1993.

Unser M., Aldroubi A., Eden M., B-Spline Signal Processing: Part II--Efficient Design and Applications. IEEE Transactions on Signal Processing, vol. 41, no. 2, pp. 834-848, February 1993.

FreeImage_FlipHorizontal

1 4 8 16 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_FlipHorizontal(FIBITMAP *dib);
```

Flip the input dib horizontally along the vertical axis.

FreeImage_FlipVertical

1 4 8 16 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_FlipVertical(FIBITMAP *dib);
```

Flip the input dib vertically along the horizontal axis.

Upsampling / downsampling

FreeImage_Rescale

8 24 32

```
DLL_API FIBITMAP * DLL_CALLCONV FreeImage_Rescale(FIBITMAP *dib, int dst_width, int
dst_height, FREE_IMAGE_FILTER filter);
```

This function performs resampling (or scaling, zooming) of a 8-, 24- or 32-bit image to the desired destination width and height. A NULL value is returned when the bitdepth cannot be handled or when there's not enough memory (this may happen with very large images).

Resampling refers to changing the pixel dimensions (and therefore display size) of an image. When you downsample (or decrease the number of pixels), information is deleted from the image. When you upsample (or increase the number of pixels), new pixels are added based on color values of existing pixels. You specify an interpolation filter to determine how pixels are added or deleted.

The following filters can be used as resampling filters:

Filter flag	Description
FILTER_BOX	Box, pulse, Fourier window, 1st order (constant) B-Spline
FILTER_BILINEAR	Bilinear filter
FILTER_BSPLINE	4th order (cubic) B-Spline
FILTER_BICUBIC	Mitchell and Netravali's two-param cubic filter
FILTER_CATMULLROM	Catmull-Rom spline, Overhauser spline
FILTER_LANCZOS3	Lanczos-windowed sinc filter

Table 10: IMAGE_FILTER constants.

Some hints on how to use these filters are given in the Appendix (see *Choosing the right resampling filter*).

References

Paul Heckbert, C code to zoom raster images up or down, with nice filtering. UC Berkeley, August 1989.
[Online] http://www-2.cs.cmu.edu/afs/cs.cmu.edu/Web/People/ph/heckbert.html

Hou H.S., Andrews H.C., Cubic Splines for Image Interpolation and Digital Filtering. IEEE Trans. Acoustics, Speech, and Signal Proc., vol. ASSP-26, no. 6, pp. 508-517, Dec. 1978.

Glassner A.S., Principles of digital image synthesis. Morgan Kaufmann Publishers, Inc, San Francisco, Vol. 2, 1995.

Mitchell Don P., Netravali Arun N., Reconstruction filters in computer graphics. In John Dill, editor, Computer Graphics (SIGGRAPH '88 Proceedings), Vol. 22, No. 4, pp. 221-228, August 1988.

Keys R.G., Cubic Convolution Interpolation for Digital Image Processing. IEEE Trans. Acoustics, Speech, and Signal Processing, vol. 29, no. 6, pp. 1153-1160, Dec. 1981.

Color manipulation

FreeImage uses the RGB(A) color model to represent color images in memory. A 8-bit greyscale image has a single channel, often called the black channel. A 24-bit image is made up of three 8-bit channels: one for each of the red, green and blue colors. For 32-bit images, a fourth 8-bit channel, called alpha channel, is used to create and store masks, which let you manipulate, isolate, and protect specific parts of an image. Unlike the others channels, the alpha channel doesn't convey color information, in a physical sense.

Color manipulation functions used in FreeImage allow you to modify the histogram of a specific channel. This transformation is known as a point operation, and may be used to adjust brightness, contrast or gamma of an image, to perform image enhancement (e.g. histogram equalization, non-linear contrast adjustment) or even to invert or threshold an image.

Currently, the following channels are defined in FreeImage:

Channel flag	Description
FICC_RGB	Function applies to red, green and blue channels
FICC_RED	Function applies to red channel only
FICC_GREEN	Function applies to green channel only
FICC_BLUE	Function applies to blue channel only
FICC_ALPHA	Function applies to alpha channel only
FICC_BLACK	Function applies to black channel
FICC_REAL	Complex images: function applies to the real part
FICC_IMAG	Complex images: function applies to the imaginary part
FICC_MAG	Complex images: function applies to the magnitude
FICC_PHASE	Complex images: function applies to the phase

Table 11: FREE_IMAGE_COLOR_CHANNEL constants.

FreeImage_AdjustCurve

8 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_AdjustCurve(FIBITMAP *dib, BYTE *LUT,
FREE_IMAGE_COLOR_CHANNEL channel);
```

Perfoms an histogram transformation on a 8-, 24- or 32-bit image according to the values of a lookup table (LUT). The size of 'LUT' is assumed to be 256. The color channel to be transformed is specified by the channel parameter. The transformation is done as follows:

- 8-bit images: if the image has a color palette, the LUT is applied to this palette, otherwise, it is applied to the grey values. The channel parameter is not used.
- 24-bit & 32-bit images: if channel is equal to FICC_RGB, the same LUT is applied to each color plane (R,G, and B). Otherwise, the LUT is applied to the specified channel only (R, G, B or A).

The function returns TRUE on success, FALSE otherwise (e.g. when the bitdepth of the source dib cannot be handled).

FreeImage_AdjustGamma

8 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_AdjustGamma(FIBITMAP *dib, double gamma);
```

Performs gamma correction on a 8-, 24- or 32-bit image. The gamma parameter represents the gamma value to use (gamma > 0). A value of 1.0 leaves the image alone, less than one darkens it, and greater than one lightens it.

The function returns TRUE on success. It returns FALSE when gamma is less than or equal to zero or when the bitdepth of the source dib cannot be handled.

FreeImage_AdjustBrightness

8 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_AdjustBrightness(FIBITMAP *dib, double
percentage);
```

Adjusts the brightness of a 8-, 24- or 32-bit image by a certain amount. This amount is given by the percentage parameter, where percentage is a value between [-100..100]. A value 0 means no change, less than 0 will make the image darker and greater than 0 will make the image brighter.

The function returns TRUE on success, FALSE otherwise (e.g. when the bitdepth of the source dib cannot be handled).

FreeImage_AdjustContrast

8 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_AdjustContrast(FIBITMAP *dib, double percentage);
```

Adjusts the contrast of a 8-, 24- or 32-bit image by a certain amount. This amount is given by the percentage parameter, where percentage is a value between [-100..100]. A value 0 means no change, less than 0 will decrease the contrast and greater than 0 will increase the contrast of the image.

The function returns TRUE on success, FALSE otherwise (e.g. when the bitdepth of the source dib cannot be handled).

FreeImage_Invert

1 4 8 16 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_Invert(FIBITMAP *dib);
```

Inverts each pixel data.

FreeImage_GetHistogram

8 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_GetHistogram(FIBITMAP *dib, DWORD *histo,
FREE_IMAGE_COLOR_CHANNEL channel FI_DEFAULT(FICC_BLACK));
```

Computes the image histogram. For 24-bit and 32-bit images, histogram can be computed from red, green, blue and black channels. For 8-bit images, histogram is computed from the black channel. Other bit depth is not supported (nothing is done and the function returns FALSE). The histo variable must be allocated by the application driving FreeImage. Its size is assumed to be equal to 256.

Channel processing

FreeImage_GetChannel

24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_GetChannel(FIBITMAP *dib,
FREE_IMAGE_COLOR_CHANNEL channel);
```

Retrieves the red, green, blue or alpha channel of a 24- or 32-bit BGR[A] image. dib is the input image to be processed and channel is the color channel to extract. The function returns the extracted channel if successful and returns NULL otherwise.

FreeImage_SetChannel

24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_SetChannel(FIBITMAP *dib, FIBITMAP *dib8,
FREE_IMAGE_COLOR_CHANNEL channel);
```

Insert a 8-bit dib into a 24- or 32-bit image. dib8 and dib must have the same width and height. dib is the destination image to modify (24- or 32-bit), dib8 is the image to insert and channel is the color channel to replace. The function returns TRUE if successful, FALSE otherwise.

FreeImage_GetComplexChannel

2x64

```
DLL_API FIBITMAP * DLL_CALLCONV FreeImage_GetComplexChannel(FIBITMAP *src,
FREE_IMAGE_COLOR_CHANNEL channel);
```

Retrieves the real part, imaginary part, magnitude or phase of a complex image (image whose type is FIT_COMPLEX). The function returns the extracted channel as a FIT_DOUBLE image if successful and returns NULL otherwise.

FreeImage_SetComplexChannel

2x64

```
DLL_API BOOL DLL_CALLCONV FreeImage_SetComplexChannel(FIBITMAP *dst, FIBITMAP *src,
FREE_IMAGE_COLOR_CHANNEL channel);
```

Set the real or imaginary part of a complex image (image whose type is FIT_COMPLEX). Both src and dst must have the same width and height. Upon entry, dst is the image to modify (image of type FIT_COMPLEX) and src is the channel to replace (image of type FIT_DOUBLE). The function returns TRUE if successful, FALSE otherwise.

Copy / Paste / Composite routines

FreeImage_Copy

1 4 8 16 24 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Copy(FIBITMAP *dib, int left, int top, int
right, int bottom);
```

Copy a sub part of the current dib image. The rectangle defined by the (left, top, right, bottom) parameters is first normalized such that the value of the left coordinate is less than the right and the top is less than the bottom. Then, the returned bitmap is defined by a width equal to (right - left) and a height equal to (bottom - top).

The function parameters follow:

left : specifies the left position of the cropped rectangle.

top : specifies the top position of the cropped rectangle.

right : specifies the right position of the cropped rectangle.

bottom : specifies the bottom position of the cropped rectangle.

The function returns the subimage if successful and returns NULL otherwise.

FreeImage_Paste

8 16 24 32

```
DLL_API BOOL DLL_CALLCONV FreeImage_Paste(FIBITMAP *dst, FIBITMAP *src, int left, int
top, int alpha);
```

Alpha blend or combine a sub part image with the current dib image. The bit depth of dst bitmap must be greater than or equal to the bit depth of src. Upper promotion of src is done internally, without modifying src. Supported bit depth equals to 8, 16, 24 or 32. Parameters follow:

dst :destination image

src : source subimage

left : specifies the left position of the sub image.

top : specifies the top position of the sub image.

alpha : alpha blend factor. The source and destination images are alpha blended if alpha=0..255. If alpha > 255, then the source image is combined to the destination image.

The function returns TRUE if successful, FALSE otherwise.

FreeImage_Composite

8 32

```
DLL_API FIBITMAP *DLL_CALLCONV FreeImage_Composite(FIBITMAP *fg, BOOL useFileBkg
FI_DEFAULT(FALSE), RGBQUAD *appBkColor FI_DEFAULT(NULL), FIBITMAP *bg
FI_DEFAULT(NULL));
```

This function composite a transparent foreground image against a single background color or against a background image. Upon entry, fg defines the foreground image and the transparency mask (implicitly included in the foreground image as a transparency table for 8-bit dib or as a alpha channel for 32-bit dib).

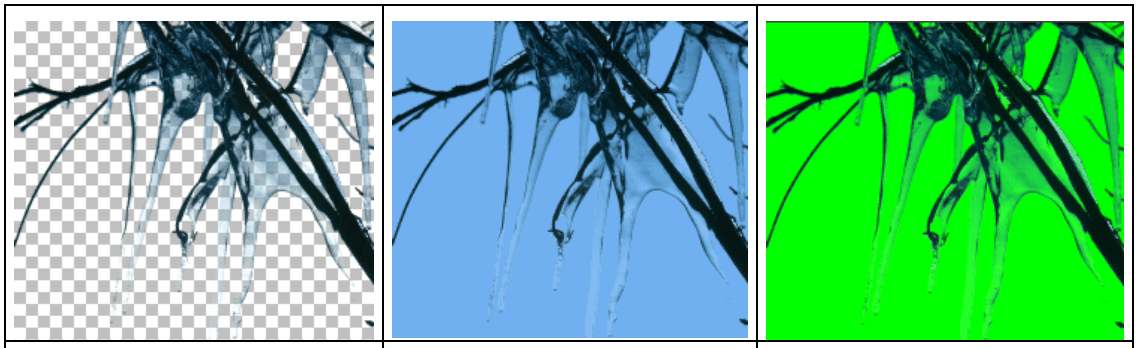
The equation for computing a composited sample value is:

*output = alpha * foreground + (1-alpha) * background*

where alpha and the input and output sample values are expressed as fractions in the range 0 to 1. For colour images, the computation is done separately for R, G, and B samples.

The following pseudo-code illustrates the internal use of the other parameters:

```
if(useFileBkg && FreeImage_HasBackgroundColor(fg)) {
  // Use the file background as the single background color
} else {
  // no file background color …
  // use application background color ?
  if(appBkColor) {
    // use the application background as the single background color
  }
  // no application background color …
  // use a background image ?
  else if(bg) {
    // use bg as the background image
    // bg MUST BE a 24-bit image with the same width and height as fg
  }
  else {
    // default case
    // use a checkerboard as the background image
  }
}
```



(a) (b) (c)

Figure 1: Illustration of the FreeImage_Composite function.

An illustration of the FreeImage_Composite function is given in Figure 1. This sample image is a 8-bit transparent PNG with a light-blue file color background. Each image was generated using the following calls:

```
FIBITMAP *fg = FreeImage_Load(FIF_PNG, "test.png", PNG_DEFAULT);
// image (a) : use a checkerboard background
FIBITMAP *display_dib_a = FreeImage_Composite(fg);
// image (b) : use the image file background if there is one
FIBITMAP *display_dib_b = FreeImage_Composite(fg, TRUE);
// image (c) : use a user specified background
RGBQUAD appColor = { 0, 255, 0, 0 };
FIBITMAP *display_dib_c = FreeImage_Composite(fg, FALSE, &appColor);
```

Reference

Portable Network Graphics (PNG) Specification (Second Edition).
[Online] http://www.w3.org/TR/PNG/

Appendix

Choosing the right resampling filter

The effect of a resampling filter is highly dependant on the physical characteristics of the image being resized. Nevertheless, the following hints may prove helpful when deciding which filter to use.

Box filter

Box scaling is the simplest and fastest of the scaling algorithms, from a computational standpoint. Various names are used to denote this simple kernel. They include the box filter, sample-and-hold function, pulse function, Fourier window, 1st order (constant) B-Spline and nearest neighbour. The technique achieves magnification by pixel replication, and minification by sparse point sampling. For large-scale changes, box interpolation produces images with a blocky appearance. In addition, shift errors of up to one-half pixel are possible. These problems make this technique inappropriate when sub-pixel accuracy is required.

Bilinear filter

Bilinear scaling is the second-fastest scaling function. It employs linear interpolation to determine the output image. Bilinear scaling provides reasonably good results at moderate cost for most applications where scale factors are relatively small (4X or less). Often, though, higher fidelity is required and thus more sophisticated filters have been formulated.

B-Spline filter

The B-spline filter produces the smoothest output, but tends to smooth over fine details. This function requires the same processing time as Mitchell and Netravali's Bicubic filter. B-spline filter is recommended for applications where the smoothest output is required.

Bicubic filter

Mitchell and Netravali's bicubic filter is an advanced parameterized scaling filter. It uses a cubic function to produce very smooth output while maintaining dynamic range and sharpness. Bicubic scaling takes approximately twice the processing time as Bilinear. This filter can be used for any scaling application, especially when scaling factors are 2X or greater.

Catmull-Rom filter

When using Mitchell-Netravali filters, you have to set two parameters b and c such that b + 2c = 1, in order to use the numerically most accurate filter. The Bicubic filter uses the default values (b = 1/3, c = 1/3), which were the values recommended by Mitchell and Netravali as

yielding the most visually pleasing results in subjective tests of human beings. When b = 0, this gives the maximum value for c = 0.5, which is the Catmull-Rom spline and a good suggestion for sharpness. The Catmull-Rom filter is generally accepted as the best cubic interpolant filter.

Lanczos filter

Lanczos uses a filter based on the sinc function. This is the most theoretically correct filter and produces the best output for photographic images that do not have sharp transitions in them. However, Lanczos will produce ripple artefacts especially for block text, due to aliasing. Lanczos also requires three times the processing time of Bilinear. Lanczos is not recommended except in very rare applications using band-limited photographic images with no sharp edges.

Comparison of resampling methods

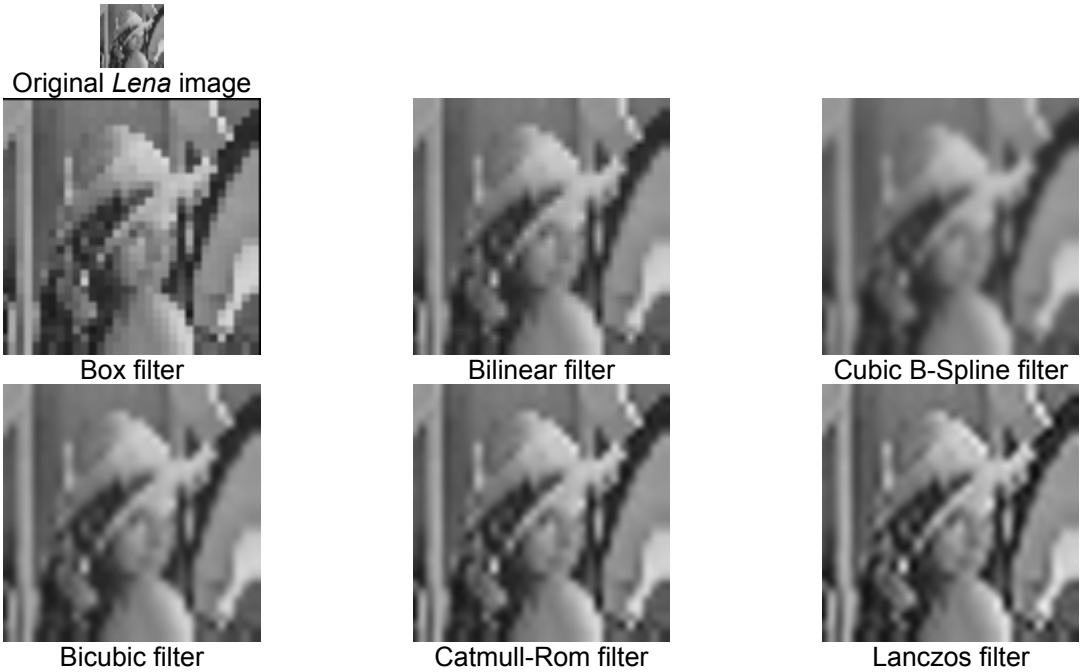
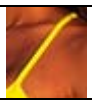


Figure 2: Comparison of resampling filters on a 32x32 Lena image resized to 400%.

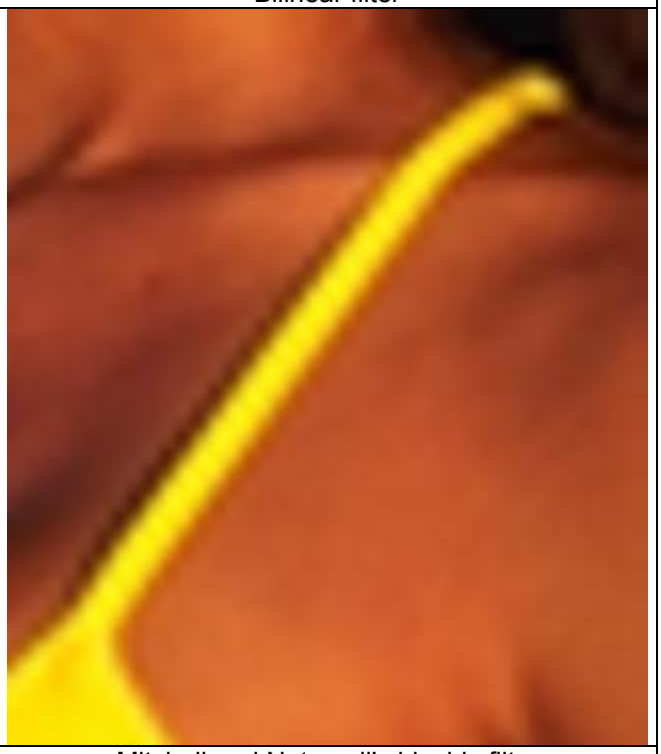


Original *Bikini* image

Box filter

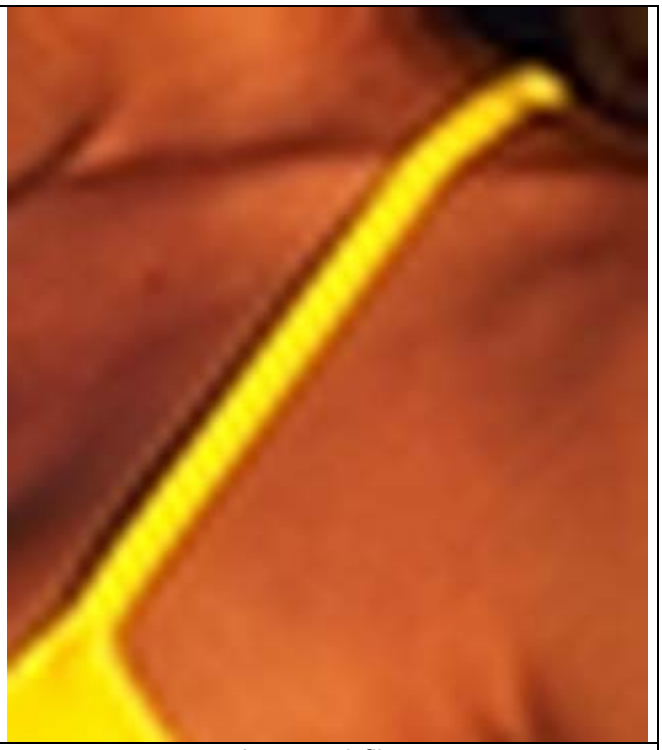
Bilinear filter

Cubic B-Spline filter



Mitchell and Netravali's bicubic filter

Catmull-Rom filter



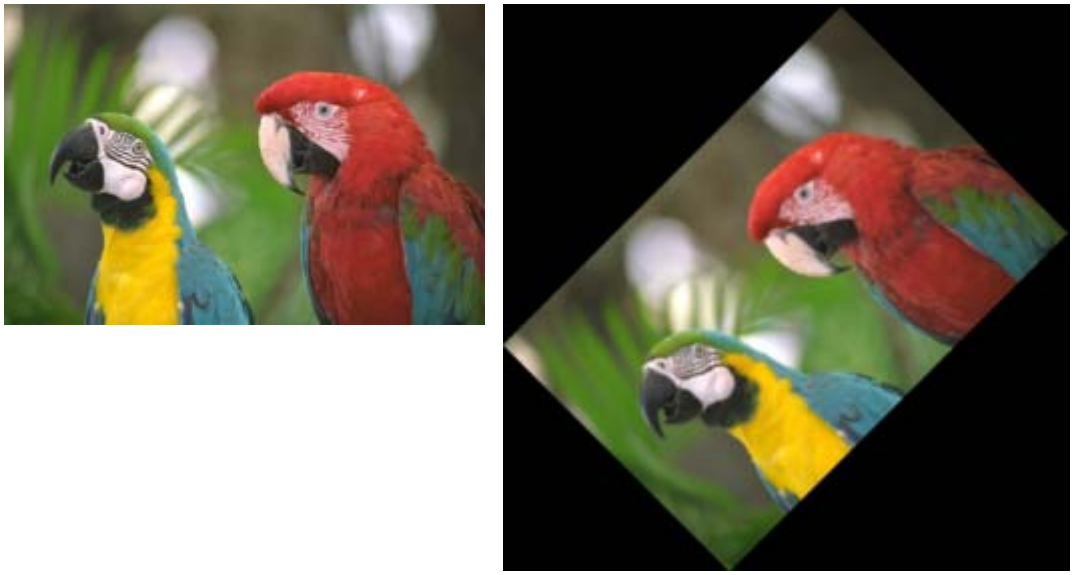
Lanczos3 filter

Figure 3: Comparison of resampling filters on a 40x46 Bikini image resized to 800%.

Using the rotation functions

FreeImage_RotateClassic

The following figure demonstrates the result of using FreeImage_RotateClassic when rotating an image by an angle of 45°. Note that the rotated image is larger than the original image.



Original *Parrot* image Rotated image

Figure 4: Parrot image rotated by 45° using FreeImage_RotateClassic.

The same image now rotated by an angle of 90° is showed in Figure 4. This time, the rotated image has the same size as the original one.

Figure 5: Parrot image rotated by 90° using FreeImage_RotateClassic.

FreeImage_RotateEx

Figure 6 shows some of the results you can obtain with the FreeImage_RotateEx function.

Original 256x256 *Lena* image

(a)

(b)

(c)

Figure 6: Some examples illustrating the use of FreeImage_RotateEx.

(a) : Image resulting from an arbitrary transformation (no masking). The image has been rotated by some angle around an arbitrary origin, while an additional translation has been thrown in for good measure. Observe the influence of mirroring the data (the function allows for the masking out of the extrapolated data, if desired).

```
FIBITMAP *dst = FreeImage_RotateEx(src, angle, x_shift, y_shift,
x_origin, y_origin, FALSE);
```

(b) : Image resulting from a simple integer translation using the following code :

```
FIBITMAP *dst = FreeImage_RotateEx(src, 0, -20, 30, 0, 0, TRUE);
```

This time, we set the *use_mask* parameter to TRUE, to mask out the irrelevant part of the image.

(c) : Image resulting from a rotation around the upper-left corner :

```
FIBITMAP *dst = FreeImage_RotateEx(src, 15, 0, 0, 0, 0, TRUE);
```