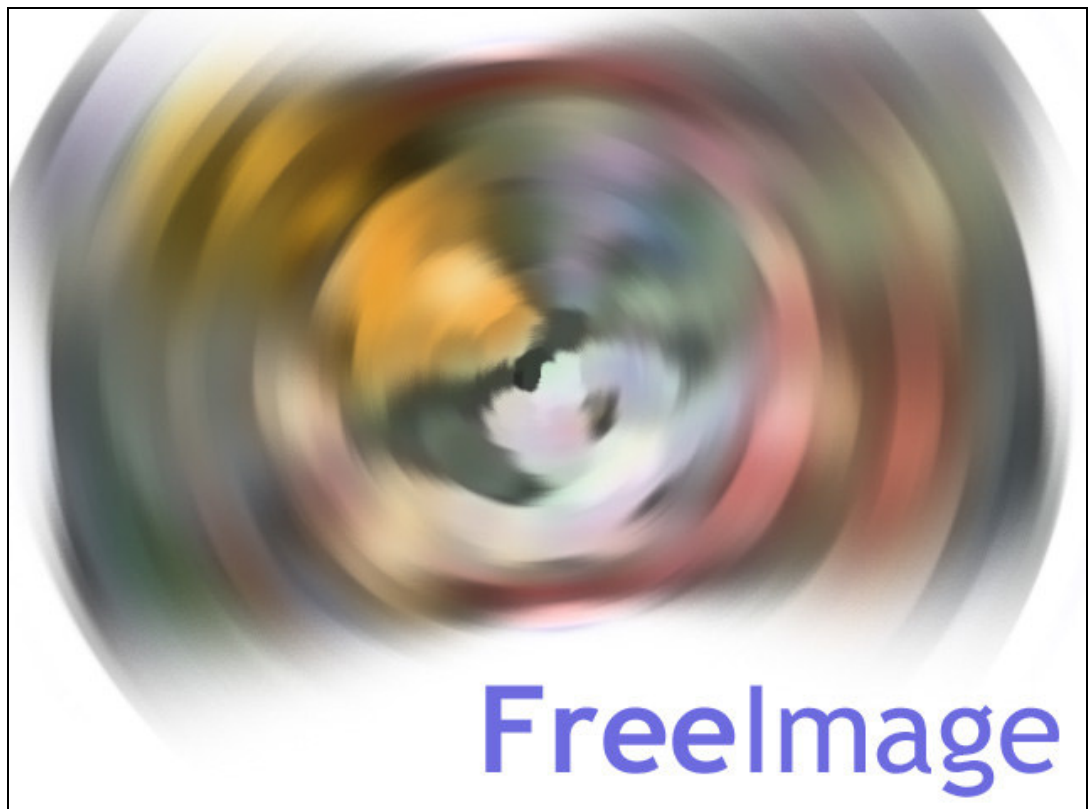

FreeImage

a free, open source graphics library

Library version 2.6.1



Contents

Introduction	1
Introduction	1
Purpose of FreeImage	1
Function reference	2
General functions	2
FreeImage_Initialise	2
FreeImage_DeInitialise	2
FreeImage_GetVersion	2
FreeImage_GetCopyrightMessage	2
FreeImage_SetOutputMessage	2
Bitmap management functions	4
FreeImage_Allocate	5
FreeImage_Load	5
FreeImage_LoadFromHandle	6
FreeImage_Save	7
FreeImage_SaveToHandle	8
FreeImage_Clone	9
FreeImage_Unload	9
Bitmap information functions	10
FreeImage_GetColorsUsed	10
FreeImage_GetBits	10
FreeImage_GetScanLine	10
FreeImage_GetBPP	10
FreeImage_GetWidth	10
FreeImage_GetHeight	11
FreeImage_GetLine	11
FreeImage_GetPitch	11
FreeImage_GetDIBSize	11
FreeImage_GetPalette	11
FreeImage_GetDotsPerMeterX	11
FreeImage_GetDotsPerMeterY	11
FreeImage_GetInfoHeader	12
FreeImage_GetInfo	12
FreeImage_GetColorType	12
FreeImage_GetRedMask	12
FreeImage_GetGreenMask	12
FreeImage_GetBlueMask	12
FreeImage_GetTransparencyCount	13
FreeImage_GetTransparencyTable	13
FreeImage_SetTransparent	13
FreeImage_IsTransparent	13
Filetype functions	14
FreeImage_GetFileType	14
FreeImage_GetFileTypeFromHandle	14
Conversion functions	15
FreeImage_ConvertTo8Bits	15

FreeImage_ConvertTo16Bits555.....	15
FreeImage_ConvertTo16Bits565.....	15
FreeImage_ConvertTo24Bits.....	15
FreeImage_ConvertTo32Bits.....	15
FreeImage_ColorQuantize.....	15
FreeImage_Threshold.....	16
FreeImage_Dither.....	16
FreeImage_ConvertFromRawBits.....	16
FreeImage_ConvertToRawBits.....	17
ICC profile functions.....	18
FreeImage_GetICCProfile.....	18
FreeImage_CreateICCProfile.....	18
FreeImage_DestroyICCProfile.....	19
Plugin functions.....	20
FreeImage_GetFIFCount.....	20
FreeImage_SetPluginEnabled.....	20
FreeImage_IsPluginEnabled.....	20
FreeImage_GetFIFFromFormat.....	20
FreeImage_GetFIFFromMime.....	20
FreeImage_GetFormatFromFIF.....	21
FreeImage_GetFIFExtensionList.....	21
FreeImage_GetFIFDescription.....	21
FreeImage_GetFIFRegExpr.....	21
FreeImage_GetFIFFromFilename.....	21
FreeImage_FIFSupportsReading.....	21
FreeImage_FIFSupportsWriting.....	22
FreeImage_FIFSupportsExportBPP.....	22
FreeImage_FIFSupportsICCProfiles.....	22
FreeImage_RegisterLocalPlugin.....	22
FreeImage_RegisterExternalPlugin.....	23
Multipage functions.....	24
FreeImage_OpenMultiBitmap.....	24
FreeImage_CloseMultiBitmap.....	24
FreeImage_GetPageCount.....	24
FreeImage_AppendPage.....	24
FreeImage_InsertPage.....	24
FreeImage_DeletePage.....	25
FreeImage_LockPage.....	25
FreeImage_UnlockPage.....	25
FreeImage_MovePage.....	25
FreeImage_GetLockedPageNumbers.....	25

Glossary of Terms	27
--------------------------	-----------

Index	29
--------------	-----------

Introduction

Introduction

Thank you for downloading FreeImage, a free and open source graphics library for Windows and Linux. FreeImage is widely used and praised for its speed and simplicity. It has been under development for more than 3 years.

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. I would like to say thank you to Jan Nauta, Herve Drolon, Markus Loibl and Martin Weber for being there from the start. Other people I greatly respect: Adam Gates, Alexander Dymerefs, Jani Kajala and Matthias Wandel. Last but not least I want to greet the people who hang out together with me on IRC on the Undernet #c++ channel. I learn a lot from these guys, and I don't want that to go unnoted. If you want to meet me: I go under the nickname NoEscom.

Purpose of FreeImage

When I started working on FreeImage I didn't have a clear idea yet what the DLL should exactly be used for. I knew it should be loading and saving bitmaps, of course, because it was originally developed to provide bitmap loading support to an authoring tool named the Magenta Multimedia Tool. It was the other features I was not so sure about. Fortunately for me over time the picture has become much clearer. 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:

- 1) Loading and saving of as many bitmap types as possible.
- 2) Easy access to bitmap components, such as palettes and data bits
- 3) Converting bitmap's bit depths from one to another
- 4) Accessing pages in a bitmap when there are multiple, such as in TIFF

FreeImage does not support:

- 1) Manipulation of bitmaps
- 2) Overlaying bitmaps
- 3) Bitmap drawing
- 4) Vector graphics

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.

```
void
MessageFunction(FREE_IMAGE_FORMAT fif, const char *msg) {
    printf("%d : %s", (int)fif, msg);
}

FreeImage_SetOutputMessage(MessageFunction);
```

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 type 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
FIF_BMP	Windows or OS/2 Bitmap File (*.BMP)
FIF_CUT	Dr. Halo (*.CUT)
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)

Table 1: 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, 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 on the same place as you would pass a `FREE_IMAGE_FORMAT`.

FreeImage_Allocate

```
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 colour 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 colour red. When `green_mask` is `0x000000FF`, it means that the first 8 bits in a pixel are used for the colour green.

Note that `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_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_FIRST	Loads the first bitmap in the icon
	ICO_SECOND	Loads the second bitmap in the icon
	ICO_THIRD	Loads the third bitmap in the icon
JPEG	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_BASE	A PhotoCD picture comes in many sizes. 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 2: 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.

Note: 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 grayscale 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.

```

// 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_SAVE_RLE	Compress the bitmap using RLE when saving
JPEG	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_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 PACKBITS 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.
	TIFF_ADOBE_DEFLATE	Save using ADOBE DEFLATE compression.
	TIFF_NONE	Save without any compression.

Table 3: 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 file.

```
// 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

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

Makes an exact reproduction of an existing bitmap. Example:

```
// 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

```
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_GetColorsUsed

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

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

Note: 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_GetBits

```
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 and GetRedMask, FreeImage_GetGreenMask and FreeImage_GetBlueMask.

Note: in FreeImage, FIBITMAP are based in a coordinate system that is upside down relative to usual graphics conventions.

FreeImage_GetScanLine

```
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.

Note: in FreeImage, the scanlines are stored upside down, with the first scan in memory being the bottommost scan in the image.

FreeImage_GetBPP

```
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 and 32.

FreeImage_GetWidth

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

Returns the width of the bitmap in pixels.

FreeImage_GetHeight

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

Returns the height of the bitmap in pixels.

FreeImage_GetLine

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

Returns the width of the bitmap in bytes.

Note: 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

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

FreeImage_GetDIBSize

```
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

```
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, this function returns NULL.

FreeImage_GetDotsPerMeterX

```
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

```
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

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

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

FreeImage_GetInfo

```
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

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

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

Value	Description
FIC_MINISBLACK	Monochrome bitmap: first palette entry is black (1 bit) Palettized bitmap: grayscale palette (8 bit)
FIC_MINISWHITE	Monochrome bitmap: first palette entry is white (1 bit)
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)

FreeImage_GetRedMask

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

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

FreeImage_GetGreenMask

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

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

FreeImage_GetBlueMask

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

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

FreeImage_GetTransparencyCount

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

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

FreeImage_GetTransparencyTable

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

Returns a pointer to the bitmap's transparency table. Only palettised bitmaps have a transparency table. High-color bitmaps store the transparency values directly in the bitmap bits.

FreeImage_SetTransparent

```
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.

FreeImage_IsTransparent

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

Returns TRUE when the transparency table is enabled, FALSE otherwise.

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, for example.

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.

Conversion functions

The following functions make it possible to convert a bitmap from one bit depth to another. In FreeImage bitmaps are always stored blue first, then green then red, then alpha (BGR[A] convention).

FreeImage_ConvertTo8Bits

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

Converts a bitmap to 8 bits. If the bitmap was a high-colour bitmap (16, 24 or 32 bit) or if it was a monochrome or grayscale bitmap (1 or 4 bit), the end result will be a grayscale bitmap, otherwise it will a palettized bitmap.

Note: when creating the grayscale palette, the grayscale intensity of a result pixel is based on red, green, and blue levels of the corresponding source pixel using the following formula:

$$\text{gray} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

The values 0.299, 0.587 and 0.114 represent the relative red, green, and blue intensities.

FreeImage_ConvertTo16Bits555

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

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

FreeImage_ConvertTo16Bits565

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

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

FreeImage_ConvertTo24Bits

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

Converts a bitmap to 24 bits.

FreeImage_ConvertTo32Bits

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

Converts a bitmap to 32 bits.

FreeImage_ColorQuantize

```
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 colour 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

FreeImage_Threshold

```
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.

FreeImage_Dither

```
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. 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 algorithm:

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)

FreeImage_ConvertFromRawBits

```
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 last three parameters are optional, and tell FreeImage the bit-layout of the colour components in the bitmap.

FreeImage_ConvertToRawBits

```
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.

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.

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

Note: If the FIICCPROFILE is flagged with FIICC_COLOR_IS_CMYK the bitmap is a representation of a CMYK separation. Together with colour management this information is important, because the profile data and the bitmap must reside in the same colour 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 colour 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

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

Retrieve the 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

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

Create a new FIICCPROFILE block from ICC profile data previously read from a file or build by a colour 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_CreateICCPProfile (
        bitmap, data, size);

    free (data);
}
```

FreeImage_DestroyICCPProfile

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

This functions destroys an FIICCPROFILE previously created by FreeImage_CreateICCPProfile. 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_DestroyICCPProfile(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 I decided to create plugins, because I wanted to support some bitmaps formats that have license restrictions on them, such as GIF. In fear that I would put all my bitmap loaders/savers in tiny DLLs that would splatter the hard drive, my most important 'customer' strongly encouraged me to keep as much bitmap formats in one DLL as possible. I 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 I now often reuse 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.

FreeImage_IsPluginEnabled

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

Returns TRUE when the plugin is enabled, FALSE 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).

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.

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.

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.

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, FALSE otherwise.

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

From version 2.5.0 on FreeImage features a new set of functions that can be used to manipulate pages in a multi-page bitmap format. Currently the TIFF format is 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.

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 is 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);
```

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

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 equal or 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.

Glossary of Terms

Index