Basic Bitmaps Documentation

By Seven

This library contains very basic tools that allow you to load, create, manipulate, and save bitmap images. This is limited to a specific type of bitmap though; they must have only the Bitmap file header, the DIB header, and the Pixel array, with the DIB header being the 'BITMAPINFOHEADER' taking up 40 bytes.

Contents

Structs:

BITMAP

Declaration:

```
typedef struct
{
    BITMAPHEADER   bitmapHeader;
    unsigned char  *imageData;
} BITMAP;
```

Description:

Contains all the necessary information to create a bitmap. Created by bmGetBitmap.

Variables:

bitmapHeader – A BITMAPHEADER struct.

imageData – A pointer to the image data for the bitmap.

BITMAPHEADER

Declaration:

```
typedef struct
{

    unsigned short      identifier;
    unsigned int        bitmapFileSize
    unsigned short      reserved1;
    unsigned short      reserved2;
    unsigned int        offset;
    unsigned int        infoHeaderSize;
    int                 width;
    int                 height;
    unsigned short      colourPlanes;
    unsigned short      bitsPerPixel;
    unsigned int        compressionMethod;
    unsigned int        imageSize;
    int                 horizontalResolution;
    int                 verticalResolution;
    unsigned int        colourPaletteNumber;
    unsigned int        importantColours;
} BITMAPHEADER;
```

Description:

Contains the necessary information regarding the bitmap header. Contained in BITMAP. Not necessary to create.

Variables:

identifier - 2 Bytes: The header field used to identify the BMP file, should always be "BM".

bitmapFileSize – 4 Bytes: The size of the file in bytes.

reserved1 – 2 Bytes: Reserved section, should be 0.

reserved2 – 2 Bytes: Reserved section, should be 0.

offset – 4 Bytes: The offset to the start of the bitmap image data, the pixel array, should be 54.

infoHeaderSize – 4 Bytes: The size of this header, should be 40.

width – 4 Bytes: The width of the pixel array, signed integer.

height – 4 Bytes: The height of the pixel array, signed integer.

colourPlanes – 2 Bytes: The number of colour planes, must be 1 according to Wikipedia.

bitsPerPixel – 2 Bytes: The number of bits per pixel, should be 32.

compressionMethod – 4 Bytes: The compression method used, 0 for no compression, should be 0.

imageSize – 4 Bytes: The size of the raw bitmap data in bytes.

horizontalResolution – 4 Bytes: The horizontal resolution of the image, pixel per metre, signed integer, should be 0.

verticalResolution – 4 Bytes: The vertical resolution of the image, pixel per metre, signed integer, should be 0.

colourPaletteNumber - 4 Bytes: The number of colours in the colour palatte, should be 0 as we wont use a colour palette.

importantColours - 4 Bytes: The number of important colours used, 0 when every colour is important, generally ignored, should be 0.

COLOUR

Declaration:

```
typedef struct
{
    unsigned char red, green, blue;
} COLOUR;
```

Description:

A simple struct used to contain 3 unsigned char values representing the red, green and blue channels for a pixel. Created by bmGetColour.

Description:

red – The colour channel for red, 0 – 255.

green – The colour channel for green, 0 – 255.

blue – The colour channel for blue, 0 – 255.

Constants:

Flags:

`BM_BLEND_RGB_ADD`

Declaration:

```
#define BM_BLEND_RGB_ADD 1
```

Description:

Passed to the flag parameter of the draw functions to make them, rather than set the colour, add the given red, green and blue values to the already existing ones. If any channel exceeds 255, then it is set to 255.

`BM_BLEND_RGB_SUB`

Declaration:

```
#define BM_BLEND_RGB_SUB 2
```

Description:

Passed to the flag parameter of the draw functions to make them, rather than set the colour, subtract the given red, green and blue values from the already existing ones. If any channel goes below 0, then it is set to 0.

Functions:

Setup, Saving and Reading of Bitmaps:

bmGetBitmap

Declaration:

```
BITMAP bmGetBitmap(int width, int height);
```

Description:

Creates a BITMAP struct for a bitmap of the given width and height and returns it. Easier to use than manually initialising the header and getting the image data yourself.

Parameters:

width – An integer representing the width of the bitmap in pixels.

height – An integer representing the height of the bitmap in pixels.

Return Value:

This function returns the created BITMAP struct.

bmHeaderInit

Declaration:

```
void bmHeaderInit(BITMAPHEADER *bitmapHeader, int width, int
height);
```

Description:

Initialises the given bitmap header to contain information regarding an image of the given width and height. Used in bmGetBitmap. Not necessary to use.

Parameters:

bitmapHeader – This is a pointer to the BITMAPHEADER struct to be initialised.

width – An integer representing the width of the bitmap in pixels.

height – An integer representing the height of the bitmap in pixels.

Return Value:

This function doesn’t return anything.

bmCreateImageData

Declaration:

```
unsigned char *bmCreateImageData(BITMAPHEADER *bitmapHeader);
```

Description:

Allocates memory for the image data described by the bitmap header and returns the pointer to this memory. Used in bmGetBitmap. Not necessary to use.

Parameters:

bitmapHeader – This is a pointer to the BITMAPHEADER struct that the image data should be created off.

Return Value:

This function returns a pointer to the destination of the allocated image memory.

bmWriteToFile

Declaration:

```
int bmWriteToFile(BITMAP bitmap, const char *fileName);
```

Description:

Saves the bitmap to the given filename, the file name must end in '.bmp'.

Parameters:

bitmap – The BITMAP struct containing the information to be saved.

filename – A pointer to a string which is the file name to be saved as.

Return Value:

An integer representing success, a 1, or failure, a 0.

bmGetBitmapFromFile

Declaration:

```
int bmGetBitmapFromFile(BITMAP *bitmap, const char *fileName);
```

Description:

Reads the given bitmap file and places the data into the bitmap struct given.

Parameters:

bitmap – The BITMAP struct wherein the bitmap data should be placed.

filename – A pointer to a string which is the name of the bitmap file to be read.

Return Value:

An integer representing success, a 1, or failure, a 0.

bmFreeBitmapImageData

Declaration:

```
void bmFreeBitmapImageData(BITMAP *bitmap);
```

Description:

Frees the image data withing the given bitmap. The image data is stored in the heap, but the rest of the information, the header, is in the stack, meaning that it cannot be freed in a like manner.

Parameters:

bitmap – The BITMAP struct of which you want to free the image data.

Return Value:

This function doesn't return anything.

Retrieving Information from Bitmaps:

`bmGetWidth`

Declaration:

```
int bmGetWidth(BITMAP bitmap);
```

Description:

Gets the width of the given bitmap and returns it.

Parameters:

bitmap – The BITMAP struct to retrieve the width from.

Return Value:

An integer representing the width of the bitmap.

`bmGetHeight`

Declaration:

```
int bmGetHeight(BITMAP bitmap);
```

Description:

Gets the height of the given bitmap and returns it.

Parameters:

bitmap – The BITMAP struct to retrieve the height from.

Return Value:

An integer representing the height of the bitmap.

`bmGetBitmapFileSize`

Declaration:

```
unsigned int bmGetBitmapFileSize(BITMAP bitmap);
```

Description:

Gets and returns the size of the whole bitmap file in bytes.

Parameters:

bitmap – The BITMAP struct to retrieve the file size from.

Return Value:

An unsigned integer representing the size of the whole bitmap file in bytes.

`bmGetImageSize`

Declaration:

```
unsigned int bmGetImageSize(BITMAP bitmap);
```

Description:

Gets and returns the size of the image data in bytes, this doesn’t include the header data like bmGetBitmapFileSize does.

Parameters:

bitmap – The BITMAP struct to retrieve the image data size from.

Return Value:

An unsigned integer representing the size of the image data in bytes.

`bmGetImageCenterX`

Declaration:

```
double bmGetImageCenterX(BITMAP bitmap);
```

Description:

Gets and returns the x coordinate of the center of the given bitmap.

Parameters:

bitmap – The BITMAP struct to retrieve the center x coordinate.

Return Value:

A double representing the x coordinate of the center.

`bmGetImageCenterY`

Declaration:

```
double bmGetImageCenterY(BITMAP bitmap);
```

Description:

Gets and returns the y coordinate of the center of the given bitmap.

Parameters:

bitmap – The BITMAP struct to retrieve the center y coordinate.

Return Value:

A double representing the y coordinate of the center.

Drawing to the Bitmap:

`bmFillImageData`

Declaration:

```
void bmFillImageData(BITMAP bitmap, COLOUR colour);
```

Description:

Fills the image data of the given bitmap with the given colour.

Parameters:

bitmap – The BITMAP struct of which to fill the image data.

colour – The COLOUR struct containing the colour to fill the image data with

Return Value:

This function doesn’t return anything.

`bmDrawRectangle`

Declaration

```
void bmDrawRectangle(BITMAP bitmap, COLOUR colour, int left, int
right, int bottom, int top, char flags);
```

Description:

Draws a rectangle with the given sides to the given bitmap with the given colour.

Ignores any part of the rectangle that is out of the bounds of the bitmap.

Parameters:

bitmap – The BITMAP struct to draw the rectangle to.

colour – The COLOUR struct containing the colour to draw the rectangle with.

left – An integer representing the x coordinate of the left of the rectangle to draw.

right – An integer representing the x coordinate of the right of the rectangle to draw, not inclusive.

bottom – An integer representing the y coordinate of the bottom of the rectangle to draw.

top – An integer representing the y coordinate of the top of the rectangle to draw, not inclusive.

flags – Any flags to draw the rectangle with, can be anything within the flags section under constants.

Return Value:

This function doesn’t return anything.

`bmDrawCircle`

Declaration:

```
void bmDrawCircle(BITMAP bitmap, COLOUR colour, int x, int y, int
radius, char flags);
```

Description:

Draws a circle with the given center coordinates and radius to the given bitmap with the given colour.

Ignores any part of the circle that is out of the bounds of the bitmap.

Parameters:

bitmap – The BITMAP struct to draw the circle to.

colour – The COLOUR struct containing the colour to draw the circle with.

x – An integer representing the x coordinate of the center of the circle.

y – An integer representing the y coordinate of the center of the circle.

radius – An integer representing the radius of the circle.

flags – Any flags to draw the circle with, can be anything within the flags section under constants.

Return Value:

This function doesn’t return anything.

`bmDrawLine`

Declaration:

```
void bmDrawLine(BITMAP bitmap, COLOUR colour, int startX, int
startY, int endX, int endY, char flags);
```

Description:

Draws a line with the given start and end coordinates to the given bitmap with the given colour.

Ignores any part of the line that is out of the bounds of the bitmap.

Parameters:

bitmap – The BITMAP struct to draw the line to.

colour – The COLOUR struct containing the colour to draw the line with.

startX – An integer representing the x coordinate of the start of the line.

startY – An integer representing the y coordinate of the start of the line.

endX – An integer representing the x coordinate of the end of the line.

endY – An integer representing the y coordinate of the end of the line.

flags – Any flags to draw the line with, can be anything within the flags section under constants.

Return Value:

This function doesn’t return anything.

```
bmSetColorAt
```

Declaration:

```
void bmSetColorAt(BITMAP bitmap, COLOUR colour, int x, int y, char
flags);
```

Description:

Sets the colour of the pixel at the given coordinates of the given bitmap to the given colour.

If the coordinates are out of the bounds of the bitmap, nothing happens.

Parameters:

bitmap – The BITMAP struct of which to set the pixel colour.

colour – The COLOUR struct containing the colour to set the pixel to.

x – An integer representing the x coordinate of the pixel.

y – An integer representing the y coordinate of the pixel.

flags – Any flags to draw the pixel with, can be anything within the flags section under constants.

Return Value:

This function doesn’t return anything.

`bmRotateImage`

Declaration:

```
void bmRotateImage(BITMAP bitmap, double xCenter, double yCenter,
double angle);
```

Description:

Rotates the given bitmap around the given coordinates by the given angle. Any part of the bitmap that is out of bounds at the end of the rotation is lost.

This rotation is by no means perfect and will most likely leave behind visual artefacts. This is especially present when rotating a bitmap with lines drawn.

Parameters:

bitmap – The BITMAP struct of which to rotate.

xCenter – The x coordinate of the point to rotate the image around.

yCenter – The y coordinate of the point to rotate the image around.

angle – The angle to rotate the image.

Return Value:

This function doesn’t return anything.

Miscellaneous:

```
bmGetColour
```

Declaration:

```
COLOUR bmGetColour(unsigned char red, unsigned char green, unsigned char blue);
```

Description:

Returns a colour struct containing the red, green, and blue values given. Its purpose is to make parts of the code much easier to read.

Parameters:

red – The red value of the colour, 0 – 255.

green – The green value of the colour, 0 – 255.

blue – The blue value of the colour, 0 – 255.

Return Value:

The COLOUR struct created from the red, green and blue values.

Example:

```c
// Including standard headers
#include <stdio.h>

// My library
#include "basicBitmaps.h"

int main(void)
{
    printf("main\n\n");

    // Setting up and creating the bitmap
    printf("Creating the bitmap\n");
    BITMAP myBitmap = bmGetBitmap(100, 100);

    // Writing some pixel data
    printf("Writing stuff to the bitmap\n");

    // Fill image with a colour
    printf("Filling image\n");
    bmFillImageData(myBitmap, bmGetColour(75, 125, 255));

    // Draw a rectangle
    printf("Drawing rectangles\n");
    bmDrawRectangle(myBitmap, bmGetColour(0, 255, 255), 10, 20, 10, 20, 0);

    // Writing the bitmap to a file
    printf("Writing to file\n");
    bmWriteToFile(myBitmap, "myBitmap.bmp");

    return 0;
}
```