Mamba C API Reference Manual

Automatically generated using doxygen

www.mamba-image.org

October 30, 2015

Contents

Contents 1

1 Data Structure Index 2
1.1 Data Structures . . . 2

2 File Index 2
2.1 File List . . . 2

3 Data Structure Documentation 2
3.1 MB3D_Image Struct Reference . . . 2
3.1.1 Detailed Description . . . 2
3.1.2 Field Documentation . . . 2
3.2 MB_Image Struct Reference . . . 2
3.2.1 Detailed Description . . . 3
3.2.2 Field Documentation . . . 3

4 File Documentation 3
4.1 mamba/mamba.h File Reference . . . 3
4.2 mamba/MB_Api.h File Reference . . . 3
4.2.1 Function Documentation . . . 4
4.3 mamba/MB_Api_3D.h File Reference . . . 12
4.3.1 Function Documentation . . . 13
4.4 mamba/MB_Api_hierarchical.h File Reference . . . 14
4.4.1 Function Documentation . . . 15
4.5 mamba/MB_Api_neighbors.h File Reference . . . 16
4.5.1 Function Documentation . . . 16
4.6 mamba/MB_Common.h File Reference . . . 20
4.6.1 Macro Definition Documentation . . . 21
4.6.2 Typedef Documentation . . . 21
4.6.3 Enumeration Type Documentation . . . 22
4.7 mamba/MB_Error.h File Reference . . . 22
4.7.1 Enumeration Type Documentation . . . 22
4.7.2 Function Documentation . . . 23
4.8 mamba/MB_Image.h File Reference . . . 23
4.8.1 Function Documentation . . . 23

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

MB3D_Image 2

MB_Image 2

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

mamba/mamba.h 3

mamba/MB_Api.h 3

mamba/MB_Api_3D.h 12

mamba/MB_Api_hierarchical.h 14

mamba/MB_Api_neighbors.h 16

mamba/MB_Common.h 20

mamba/MB_Error.h 22

mamba/MB_Image.h 23

3 Data Structure Documentation

3.1 MB3D_Image Struct Reference

`#include <MB_Common.h>`

Data Fields

- MB_Image ** seq
- Uint32 length

3.1.1 Detailed Description

3D image

3.1.2 Field Documentation

Uint32 length the length of the sequence

MB_Image seq** The images sequence composing the 3D data

The documentation for this struct was generated from the following file:

- mamba/MB_Common.h

3.2 MB_Image Struct Reference

`#include <MB_Common.h>`

Data Fields

- Uint32 width
- Uint32 height
- Uint32 depth
- PLINE ∗ plines
- PIX8 ∗ pixels

3.2.1 Detailed Description

2D image

3.2.2 Field Documentation

Uint32 depth The depth of the image

Uint32 height The height of the image

PIX8∗ pixels pixel array

PLINE∗ plines access to pixel lines

Uint32 width The width of the image
The documentation for this struct was generated from the following file:

- mamba/MB_Common.h

4 File Documentation

4.1 mamba/mamba.h File Reference

```
#include <mamba/MB_Common.h>
#include <mamba/MB_Error.h>
#include <mamba/MB_Image.h>
#include <mamba/MB_Api.h>
#include <mamba/MB_Api_3D.h>
#include <mamba/MB_Api_neighbors.h>
#include <mamba/MB_Api_hierarchical.h>
```

4.2 mamba/MB_Api.h File Reference

Functions

- MB_API_ENTRY MB_errcode MB_API_CALL MB_And (MB_Image ∗src1, MB_Image ∗src2, M↩ B_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Or (MB_Image ∗src1, MB_Image ∗src2, M↩ B_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Xor (MB_Image ∗src1, MB_Image ∗src2, M↩ B_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Inv (MB_Image ∗src, MB_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Inf (MB_Image ∗src1, MB_Image ∗src2, M↩ B_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Sup (MB_Image ∗src1, MB_Image ∗src2, M↩ B_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_SupMask (MB_Image ∗src1, MB_Image ∗src2, MB_Image ∗dest, Uint32 strict)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Add (MB_Image ∗src1, MB_Image ∗src2, M↩ B_Image ∗dest)

- MB_API_ENTRY MB_errcode MB_API_CALL MB_Sub (MB_Image ∗src1, MB_Image ∗src2, M↩ B_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Mul (MB_Image ∗src1, MB_Image ∗src2, M↩ B_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Div (MB_Image ∗src1, MB_Image ∗src2, M↩ B_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Diff (MB_Image ∗src1, MB_Image ∗src2, M↩ B_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_ConAdd (MB_Image ∗src, Sint64 value, MB↩ _Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_ConSub (MB_Image ∗src, Sint64 value, MB↩ _Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_ConMul (MB_Image ∗src, Uint32 value, MB↩ _Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_ConDiv (MB_Image ∗src, Uint32 value, MB↩ _Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_ConSet (MB_Image ∗dest, Uint32 value)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Volume (MB_Image ∗src, Uint64 ∗pVolume)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Check (MB_Image ∗src, Uint32 ∗isEmpty)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Lookup (MB_Image ∗src, MB_Image ∗dest, Uint32 ∗ptab)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Histo (MB_Image ∗src, Uint32 ∗phisto)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Compare (MB_Image ∗src, MB_Image ∗cmp, MB_Image ∗dest, Sint32 ∗px, Sint32 ∗py)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Thresh (MB_Image ∗src, MB_Image ∗dest, Uint32 low, Uint32 high)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Mask (MB_Image ∗src, MB_Image ∗dest, Uint32 maskf, Uint32 maskt)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Range (MB_Image ∗src, Uint32 ∗min, Uint32 ∗max)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_depthRange (MB_Image ∗src, Uint32 ∗min, Uint32 ∗max)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_CopyBitPlane (MB_Image ∗src, MB_Image ∗dest, Uint32 plane)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_CopyBytePlane (MB_Image ∗src, MB_Image ∗dest, Uint32 plane)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Label (MB_Image ∗src, MB_Image ∗dest, Uint32 lblow, Uint32 lbhigh, Uint32 ∗pNbobj, enum MB_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Distanceb (MB_Image ∗src, MB_Image ∗dest, enum MB_grid_t grid, enum MB_edgemode_t edge)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Frame (MB_Image ∗src, Uint32 thresval, Uint32 ∗ulx, Uint32 ∗uly, Uint32 ∗brx, Uint32 ∗bry)

4.2.1 Function Documentation

MB_API_ENTRY MB_errcode MB_API_CALL MB_Add (MB_Image ∗ *src1*, MB_↩ Image ∗ *src2*, MB_Image ∗ *dest*) Adds the pixels of two images and puts the result in the third image. Depending on the format of the target image, the result may be saturated or not. You can perform the following additions :

- 1-bit + 1-bit = 1-bit (binary OR)
- 1-bit + 8-bit = 8-bit (saturated)
- 1-bit + 8-bit = 32-bit
- 1-bit + 32-bit = 32-bit
- 8-bit + 8-bit = 8-bit (saturated)

- 8-bit + 8-bit = 32-bit
- 8-bit + 32-bit = 32-bit
- 32-bit + 32-bit = 32-bit

See also

MB_Or for the binary OR definition.

Parameters

src1	image 1
src2	image 2
dest	image resulting of the addition of image 1 and 2

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_And (MB_Image ∗ *src1*, MB_Image ∗ *src2*, MB_Image ∗ *dest*) Performs a bitwise AND between the pixels of two images.

Parameters

src1	image 1
src2	image 2
dest	destination image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Check (MB_Image ∗ *src*, Uint32 ∗ *isEmpty*) Verifies that the image is not empty (all pixels to 0).

Parameters

src	the source image
isEmpty	an integer which is set to 1 if empty or 0 if not

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Compare (MB_Image ∗ *src*, MB↩_Image ∗ *cmp*, MB_Image ∗ *dest*, Sint32 ∗ *px*, Sint32 ∗ *py*) Performs a comparaison between a source image and a given base image.

Parameters

src	the source image
cmp	the image to which the source image is compared
dest	destination image
px	position in x of the first different pixel between the two images (-1 if images are similar)
py	position in y of the first different pixel between the two images (-1 if images are similar)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_ConAdd (MB_Image ∗ *src*, Sint64 *value*, MB_Image ∗ *dest*) Adds a constant value to the pixels of an image.

Parameters

src	the source image
value	the constant value to be added to the pixels
dest	the image resulting of the addition of image 1 and value

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_ConDiv (MB_Image ∗ *src*, Uint32 *value*, MB_Image ∗ *dest*) Divides (quotient) the pixels of an image by a constant value.

Parameters

src	the source image
value	the constant value used in the division
dest	the image resulting of the division of image 1 by the value

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_ConMul (MB_Image ∗ *src*, Uint32 *value*, MB_Image ∗ *dest*) Multiplies a constant value to the pixels of an image.

Parameters

src	the source image
value	the constant value to be multiplied to the pixels
dest	the image resulting of the multiplication of image 1 by the value

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_ConSet (MB_Image ∗ *dest*, Uint32 *value*) Fills an image with a specific value.

Parameters

dest	the image
value	the value to fill the image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_ConSub (MB_Image ∗ *src*, Sint64 *value*, MB_Image ∗ *dest*) Subtracts a constant value to the pixels of an image.

Parameters

src	the source image
value	the constant value to be subtracted to the pixels
dest	the image resulting of the subtraction of image 1 and value

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_CopyBitPlane (MB_Image ∗ *src*, MB_Image ∗ *dest*, Uint32 *plane*) Inserts or extracts the bit plane in/out of image src into dest.

Parameters

src	source image
dest	destination image
plane	the plane number

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_CopyBytePlane (MB_Image ∗ *src*, MB_Image ∗ *dest*, Uint32 *plane*) Inserts or extracts the byte plane in/out of image src into dest.
Parameters

src	source image
dest	destination image
plane	the plane number

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_depthRange (MB_Image ∗ *src*, Uint32 ∗ *min*, Uint32 ∗ *max*) gives the minimum and maximum possible values of the image pixels given the image depth.
Parameters

src	source image
min	the minimum possible value of the pixels
max	the maximum possible value of the pixels

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Diff (MB_Image ∗ *src1*, MB_Image ∗ *src2*, MB_Image ∗ *dest*) Computes the set difference between two images. The result image pixel value is the pixel value of image 1 if this value was greater than value of pixel 2 otherwise the minimum possible value is set for the pixel.
Parameters

src1	source image 1
src2	source image 2
dest	destination image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Distanceb (MB_Image ∗ *src*, MB↩ _Image ∗ *dest*, enum MB_grid_t *grid*, enum MB_edgemode_t *edge*) Computes for each pixel the distance to the edge of the set in which the pixel is found.

The algorithm works with a list.
Parameters

src	the binary source image
dest	the 32-bit image in which the distance for each pixel is stored

grid	the grid used (either hexagonal or square)
edge	the kind of edge to use (behavior for pixels near edge depends on it)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Div (MB_Image ∗ *src1*, MB_Image ∗ *src2*, MB_Image ∗ *dest*) Divides the pixels of two images and puts the result in the third image. You can perform the following multiplications :

- 8-bit / 8-bit = 8-bit (saturated)
- 8-bit / 8-bit = 32-bit
- 32-bit / 8-bit = 32-bit
- 32-bit / 32-bit = 32-bit

Division by zero results in the maximum value possible for the pixels of the destination image depth.
Parameters

src1	image 1
src2	image 2
dest	image resulting of the division of image 1 and 2 (quotient)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Frame (MB_Image ∗ *src*, Uint32 *thresval*, Uint32 ∗ *ulx*, Uint32 ∗ *uly*, Uint32 ∗ *brx*, Uint32 ∗ *bry*) Returns the smallest frame that contains all the pixels of image that are greater or equal to the given threshold value, using the four last pointers to describe it.
Parameters

src	source image
thresval	the threshold value used to compute the frame
ulx	the x-coordinate of the upper left corner of the frame
uly	the y-coordinate of the upper left corner of the frame
brx	the x-coordinate of the bottom right corner of the frame
bry	the y-coordinate of the bottom right corner of the frame

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Histo (MB_Image ∗ *src*, Uint32 ∗ *phisto*) Computes the histogram of an image. The histogram is an array with a minimal size of 256.
Parameters

src	source image
phisto	pointer to the histogram array

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Inf (MB_Image ∗ *src1*, MB_Image ∗ *src2*, MB_Image ∗ *dest*) Determines the inferior value between the pixels of two images. The result is put in the corresponding pixel position in the destination image.

Parameters

src1	image 1
src2	image 2
dest	destination image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Inv (MB_Image ∗ *src,* MB_Image ∗ *dest*) Inverts the pixels values (logical NOT) of the source image.
Parameters

src	source image
dest	destination image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Label (MB_Image ∗ *src,* MB_Image ∗ *dest,* Uint32 *lblow,* Uint32 *lbhigh,* Uint32 ∗ *pNbobj,* enum MB_grid_t *grid*) Labeling the object found in src image.
Parameters

src	the source image where the object must be labelled
dest	the 32-bit image where object are labelled
lblow	the lowest value allowed for label on the low byte (must be inferior to lbhigh)
lbhigh	the first high value NOT allowed for label on the low byte (maximum allowed is 256)
pNbobj	the number of object found
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Lookup (MB_Image ∗ *src,* MB_↩ Image ∗ *dest,* Uint32 ∗ *ptab*) Applies the function in the lookup table to the pixels of source image. A pixel value is changed to a new value accordingly with its current value.
Parameters

src	source image
dest	destination image
ptab	the lookup table pointer

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Mask (MB_Image ∗ *src,* MB_Image ∗ *dest,* Uint32 *maskf,* Uint32 *maskt*) Converts a binary image in a grey scale image (8-bit) or in a 32-bit image using value maskf to replace 0 and maskt to replace 1.
Parameters

src	binary source image
dest	destination image

maskf	for 0 (false) pixel value
maskt	for 1 (true) pixel value

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Mul (MB_Image ∗ *src1*, MB_Image ∗ *src2*, MB_Image ∗ *dest*) Multiplies the pixels of two images and puts the result in the third image. Depending on the format of the target image, the result may be saturated or not. You can perform the following multiplications :

- 1-bit ∗ 1-bit = 1-bit (binary AND)
- 1-bit ∗ 8-bit = 8-bit
- 1-bit ∗ 8-bit = 32-bit
- 1-bit ∗ 32-bit = 32-bit
- 8-bit ∗ 8-bit = 8-bit (saturated)
- 8-bit ∗ 8-bit = 32-bit
- 8-bit ∗ 32-bit = 32-bit
- 32-bit ∗ 32-bit = 32-bit

See also

MB_And for the binary AND.

Parameters

src1	image 1
src2	image 2
dest	image resulting of the multiplication of image 1 and 2

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Or (MB_Image ∗ *src1*, MB_Image ∗ *src2*, MB_Image ∗ *dest*) Applies a bitwise OR on the pixels of two images. All the images must have the same depth for correct work.

Parameters

src1	image 1
src2	image 2
dest	destination image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Range (MB_Image ∗ *src*, Uint32 ∗ *min*, Uint32 ∗ *max*) gives the minimum and maximum values of the image pixels i.e its range.

Parameters

src	source image
min	the minimum value of the pixels
max	the maximum value of the pixels

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Sub (MB_Image ∗ ***src1*, MB_Image** ∗ ***src2*, MB_Image** ∗ ***dest*)** Subtracts the values of pixels of the second image to the values of the pixels in the first image You can perform the following substractions :

- 1-bit - 1-bit = 1-bit (binary AND)
- 8-bit - 1-bit = 8-bit (saturated)
- 8-bit - 8-bit = 8-bit (saturated)
- 8-bit - 8-bit = 32-bit
- 8-bit - 32-bit = 32-bit
- 32-bit - 8-bit = 32-bit
- 32-bit - 32-bit = 32-bit

See also

MB_Diff for the set difference.

Parameters

src1	image 1
src2	image 2
dest	image resulting of the subtraction

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Sup (MB_Image ∗ ***src1*, MB_Image** ∗ ***src2*, MB_Image** ∗ ***dest*)** Determines the superior value between the pixels of two images. The result is put in the corresponding pixel position in the destination image.
Parameters

src1	image 1
src2	image 2
dest	destination image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_SupMask (MB_Image ∗ ***src1*, M↩B_Image** ∗ ***src2*, MB_Image** ∗ ***dest*, Uint32 *strict*)** Computes a binary image where pixels are set to 1 when the pixels of image 1 have greater values than pixels of image 2 otherwise 0.
Parameters

src1	source image 1

src2	source image 2
dest	destination image
strict	flag indicating if the comparison is strict or large

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Thresh (MB_Image ∗ *src,* MB_↩ Image ∗ *dest,* Uint32 *low,* Uint32 *high*) Fills a binary image according to the following rules: if pixel value lower than low or higher than high the binary pixel is set to 0, in other cases the pixel is set to 1.
Parameters

src	source image
dest	destination image
low	low value for threshold
high	high value for treshold

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Volume (MB_Image ∗ *src,* Uint64 ∗ *pVolume*) Computes the volume of an image. The volume is the sum of the pixel values (i.e. integration of the image).
Parameters

src	source image
pVolume	pointer to the volume variable

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Xor (MB_Image ∗ *src1,* MB_Image ∗ *src2,* MB_Image ∗ *dest*) Applies a bitwise XOR on the pixels of two images. All the images must have the same depth for correct work.
Parameters

src1	image 1
src2	image 2
dest	destination image

Returns

An error code (NO_ERR if successful)

4.3 mamba/MB_Api_3D.h File Reference

Functions

- MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Watershed (MB3D_Image ∗src, MB3D_↩ Image ∗marker, Uint32 max_level, enum MB3D_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Basins (MB3D_Image ∗src, MB3D_Image ∗marker, Uint32 max_level, enum MB3D_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB3D_HierarBld (MB3D_Image ∗mask, MB3D_↩ Image ∗srcdest, enum MB3D_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB3D_HierarDualBld (MB3D_Image ∗mask, M↩ B3D_Image ∗srcdest, enum MB3D_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Label (MB3D_Image ∗src, MB3D_Image ∗dest, Uint32 lblow, Uint32 lbhigh, Uint32 ∗pNbobj, enum MB3D_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Distanceb (MB3D_Image ∗src, MB3D_↩ Image ∗dest, enum MB3D_grid_t grid, enum MB_edgemode_t edge)

4.3.1 Function Documentation

MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Basins (MB3D_Image ∗ *src*, M↩ B3D_Image ∗ *marker*, Uint32 *max_level*, enum MB3D_grid_t *grid*) Performs a watershed segmentation of the 3D image using the 3D marker image as a starting point for the flooding. The function returns the catchment basins of the watershed but no actual watershed line. It is recommended to use this function rather than MB_Watershed if you are only interested in catchment basins (faster).

The result is put into the 32-bit marker image.

The segmentation is coded as follows into the 32-bit values.

0	**1**	**2**	**3**
<—	label	—>	unused

See also

MB_CopyBytePlane to access each byte individually.

Parameters

src	the 3D image (greyscale or 32-bit) to segment
marker	the 3D marker image in which the result of segmentation will be put
max_level	the maximum level reached by the water
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Distanceb (MB3D_Image ∗ *src*, MB3D_Image ∗ *dest*, enum MB3D_grid_t *grid*, enum MB_edgemode_t *edge*) Computes for each pixel the distance to the edge of the set in which the pixel is found.

Parameters

src	the binary source 3D image
dest	the 32-bit 3D image in which the distance for each pixel is stored
grid	the grid used (either cubic or face_center_cubic)
edge	the kind of edge to use (behavior for pixels near edge depends on it)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB3D_HierarBld (MB3D_Image ∗ *mask*, MB3D_Image ∗ *srcdest*, enum MB3D_grid_t *grid*) (re)Builds a 3D image according to a 3D mask image and using a hierarchical list to compute the rebuild.

Parameters

mask	the mask image
srcdest	the rebuild image
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB3D_HierarDualBld (MB3D_Image ∗ *mask*, MB3D_Image ∗ *srcdest*, enum MB3D_grid_t *grid*) (re)Builds (dual operation) a 3D image according to a 3D mask image and using a hierarchical list to compute the rebuild.

Parameters

mask	the mask image
srcdest	the rebuild image
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Label (MB3D_Image ∗ *src*, M↩ B3D_Image ∗ *dest*, Uint32 *lblow*, Uint32 *lbhigh*, Uint32 ∗ *pNbobj*, enum MB3D_grid_t *grid*) Labeling the object found in src 3D image.

Parameters

src	the source 3D image where the object must be labelled
dest	the 32-bit 3D image where object are labelled
lblow	the lowest value allowed for label on the low byte (must be inferior to lbhigh)
lbhigh	the first high value NOT allowed for label on the low byte (maximum allowed is 256)
pNbobj	the number of object found
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Watershed (MB3D_Image ∗ *src*, MB3D_Image ∗ *marker*, Uint32 *max_level*, enum MB3D_grid_t *grid*) Performs a watershed segmentation of the 3D image using the marker image as a starting point for the flooding. The function builds the actual watershed line (idempotent) plus catchment basins (not idempotent). The result is put into the 32-bit marker image.

The segmentation is coded as follows into the 32-bit values.

0	**1**	**2**	**3**
<—	label	—>	isLine

isLine is a value indicating if the pixel belongs to the watershed (255 if this is the case, undefined otherwise).

See also

MB_CopyBytePlane to access each byte individually.

Parameters

src	the 3D image (greyscale or 32-bit) to segment
marker	the marker 3D image in which the result of segmentation will be put
max_level	the maximum level reach by the water.
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

4.4 mamba/MB_Api_hierarchical.h File Reference

Functions

- MB_API_ENTRY MB_errcode MB_API_CALL MB_HierarBld (MB_Image ∗mask, MB_Image ∗srcdest, enum MB_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_HierarDualBld (MB_Image ∗mask, MB_Image ∗srcdest, enum MB_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Watershed (MB_Image ∗src, MB_Image ∗marker, Uint32 max_level, enum MB_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Basins (MB_Image ∗src, MB_Image ∗marker, Uint32 max_level, enum MB_grid_t grid)

4.4.1 Function Documentation

MB_API_ENTRY MB_errcode MB_API_CALL MB_Basins (MB_Image ∗ *src*, MB_↩ Image ∗ *marker*, Uint32 *max_level*, enum MB_grid_t *grid*) Performs a watershed segmentation of the image using the marker image as a starting point for the flooding. The function returns the catchment basins of the watershed but no actual watershed line. It is recommended to use this functions rather than MB_Watershed if you are only interested in catchment basins (faster).

The result is put into a the 32-bit marker image.

The segmentation is coded as follows into the 32-bit values.

0	**1**	**2**	**3**
<—	label	—>	unused

See also

MB_CopyBytePlane to access each byte individually.

Parameters

src	the greyscale or 32-bit image to be segmented
marker	the marker image in which the result of segmentation will be put
max_level	the maximum level reached by the water
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_HierarBld (MB_Image ∗ *mask*, M↩ B_Image ∗ *srcdest*, enum MB_grid_t *grid*) (re)Builds an image according to a mask image and using a hierarchical list to compute the rebuild.

Parameters

mask	the mask image
srcdest	the rebuild image
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_HierarDualBld (MB_Image ∗ *mask*, MB_Image ∗ *srcdest*, enum MB_grid_t *grid*) (re)Builds (dual operation) an image according to a mask image and using a hierarchical list to compute the rebuild.

Parameters

mask	the mask image
srcdest	the rebuild image
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Watershed (MB_Image ∗ *src*, MB↩ _Image ∗ *marker*, Uint32 *max_level*, enum MB_grid_t *grid*) Performs a watershed segmentation of the image using the marker image as a starting point for the flooding. The function builds the actual watershed line (idempotent) plus catchment basins (not idempotent).

The result is put into the 32-bit marker image.

The segmentation is coded as follows into the 32-bit values.

0	**1**	**2**	**3**
<—	label	—>	isLine

isLine is a value indicating if the pixel belongs to the watershed (255 if this is the case, undefined otherwise).

See also

MB_CopyBytePlane to access each byte individually.

Parameters

src	the greyscale or 32-bit image to segment
marker	the marker image in which the result of segmentation will be put
max_level	the maximum level reach by the water
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

4.5 mamba/MB_Api_neighbors.h File Reference

Functions

- MB_API_ENTRY MB_errcode MB_API_CALL MB_InfNb (MB_Image ∗src, MB_Image ∗srcdest, Uint32 neighbors, enum MB_grid_t grid, enum MB_edgemode_t edge)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_InfFarNb (MB_Image ∗src, MB_Image ∗srcdest, Uint32 nbrnum, Uint32 count, enum MB_grid_t grid, enum MB_edgemode_t edge)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_SupNb (MB_Image ∗src, MB_Image ∗srcdest, Uint32 neighbors, enum MB_grid_t grid, enum MB_edgemode_t edge)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_SupFarNb (MB_Image ∗src, MB_Image ∗srcdest, Uint32 nbrnum, Uint32 count, enum MB_grid_t grid, enum MB_edgemode_t edge)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_DiffNb (MB_Image ∗src, MB_Image ∗srcdest, Uint32 neighbors, enum MB_grid_t grid, enum MB_edgemode_t edge)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_BldNb (MB_Image ∗mask, MB_Image ∗srcdest, Uint32 dirnum, Uint64 ∗pVolume, enum MB_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_DualBldNb (MB_Image ∗mask, MB_Image ∗srcdest, Uint32 dirnum, Uint64 ∗pVolume, enum MB_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Shift (MB_Image ∗src, MB_Image ∗dest, Uint32 dirnum, Uint32 count, Uint32 long_filler_pix, enum MB_grid_t grid)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_ShiftVector (MB_Image ∗src, MB_Image ∗dest, Sint32 dx, Sint32 dy, Uint32 long_filler_pix)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_InfVector (MB_Image ∗src, MB_Image ∗srcdest, Sint32 dx, Sint32 dy, enum MB_edgemode_t edge)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_SupVector (MB_Image ∗src, MB_Image ∗srcdest, Sint32 dx, Sint32 dy, enum MB_edgemode_t edge)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_BinHitOrMiss (MB_Image ∗src, MB_Image ∗dest, Uint32 es0, Uint32 es1, enum MB_grid_t grid, enum MB_edgemode_t edge)

4.5.1 Function Documentation

MB_API_ENTRY MB_errcode MB_API_CALL MB_BinHitOrMiss (MB_Image ∗ *src*, MB_Image ∗ *dest*, Uint32 *es0*, Uint32 *es1*, enum MB_grid_t *grid*, enum MB_edgemode_t *edge*) Performs a binary Hit-or-Miss operation on src image using the structuring elements es0 and es1. Structuring elements are integer values coding which direction must be taken into account. es0 indicates which neighbor of the current pixel will be checked for 0 value. es1 those which will be evaluated for 1 value.

For example, in hexagonal grid, it means that if you want to look for a pattern where the neighbors in direction 6 and 1 are true while the current pixel is false just as neighbors 2 and 5, you will encode this in the elements es0 and es1 like this :

```
es0 = 1+4+32

es1 = 64+2
```

See also

MB_Neighbors_code_t for encoded structuring elements.

Parameters

src	output image
dest	input image (must be different of src)
es0	structuring element for 0 value.
es1	structuring element for 1 value.
grid	grid configuration
edge	the kind of edge to use (behavior for pixel near edge depends on it)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_BldNb (MB_Image ∗ *mask*, MB↩ _Image ∗ *srcdest*, Uint32 *dirnum*, Uint64 ∗ *pVolume*, enum MB_grid_t *grid*) (re)Builds an image according to a direction and a mask image. The direction depends on the grid used.
Parameters

mask	the mask image
srcdest	the rebuild image
dirnum	the direction number
pVolume	the computed volume of the output image
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_DiffNb (MB_Image ∗ *src*, MB_↩ Image ∗ *srcdest*, Uint32 *neighbors*, enum MB_grid_t *grid*, enum MB_edgemode_t *edge*) Computes the set difference between two image pixels (a central pixel and its neighbors in the other image) The neighbor depends on the grid used. Neighbors are described using a pattern. If no neighbor is defined, the function will leave silently doing nothing.

See also

MB_Neighbors_code_t for encoding neighbors patterns.

Parameters

src	source image in which the neighbor are taken
srcdest	source of the central pixel and destination image
neighbors	the neighbors to take into account
grid	the grid used (either square or hexagonal)
edge	the kind of edge to use (behavior for pixel near edge depends on it)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_DualBldNb (MB_Image ∗ *mask*, M↩ B_Image ∗ *srcdest*, Uint32 *dirnum*, Uint64 ∗ *pVolume*, enum MB_grid_t *grid*) (re)Builds (dual operation) an image according to a direction and a mask image. The direction depends on the grid used.
Parameters

mask	the mask image
srcdest	the rebuild image
dirnum	the direction number
pVolume	the computed volume of the output image
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_InfFarNb (MB_Image * *src*, MB_Image * *srcdest*, Uint32 *nbrnum*, Uint32 *count*, enum MB_grid_t *grid*, enum MB_edgemode_t *edge*) Looks for the minimum between two image pixels (a central pixel and its far neighbor in the other image) The neighbor depends on the grid used.

Parameters

src	source image in which the neighbor are taken
srcdest	source of the central pixel and destination image
nbrnum	the neighbor index
count	the amplitude of the shift (in pixels)
grid	the grid used (either square or hexagonal)
edge	the kind of edge to use (behavior for pixel near edge depends on it)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_InfNb (MB_Image * *src*, MB_Image * *srcdest*, Uint32 *neighbors*, enum MB_grid_t *grid*, enum MB_edgemode_t *edge*) Looks for the minimum between two image pixels (a central pixel and its neighbors in the other image) The neighbor depends on the grid used. Neighbors are described using a pattern. If no neighbor is defined, the function will leave silently doing nothing.

See also

MB_Neighbors_code_t for encoding neighbors patterns.

Parameters

src	source image in which the neighbor are taken
srcdest	source of the central pixel and destination image
neighbors	the neighbors to take into account
grid	the grid used (either square or hexagonal)
edge	the kind of edge to use (behavior for pixel near edge depends on it)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_InfVector (MB_Image * *src*, MB_Image * *srcdest*, Sint32 *dx*, Sint32 *dy*, enum MB_edgemode_t *edge*) Looks for the minimum between two image pixels (a central pixel and its neighbor in the other image previously shifted by the given vector)

Parameters

src	source image in which the neighbor are taken
srcdest	source of the central pixel and destination image

dx	the vector amplitude in x
dy	the vector amplitude in y
edge	the kind of edge to use (behavior for pixels near edge depends on it)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Shift (MB_Image ∗ *src*, MB_Image ∗ *dest*, Uint32 *dirnum*, Uint32 *count*, Uint32 *long_filler_pix*, enum MB_grid_t *grid*) Shifts the contents of an image in a given direction with a given amplitude The direction depends on the grid used.
Parameters

src	source image
dest	destination image
dirnum	the direction index
count	the amplitude of the shift
long_filler_pix	the value used to fill the created space
grid	the grid used (either square or hexagonal)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_ShiftVector (MB_Image ∗ *src*, M↩ B_Image ∗ *dest*, Sint32 *dx*, Sint32 *dy*, Uint32 *long_filler_pix*) Shifts the contents of an image by a given vector.
Parameters

src	source image
dest	destination image
dx	the vector amplitude in x
dy	the vector amplitude in y
long_filler_pix	the value used to fill the created space

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_SupFarNb (MB_Image ∗ *src*, M↩ B_Image ∗ *srcdest*, Uint32 *nbrnum*, Uint32 *count*, enum MB_grid_t *grid*, enum MB_↩ edgemode_t *edge*) Looks for the maximum between two image pixels (a central pixel and its far neighbor in the other image) The neighbor depends on the grid used.
Parameters

src	source image in which the neighbor are taken
srcdest	source of the central pixel and destination image
nbrnum	the neighbor index
count	the amplitude of the shift (in pixels)
grid	the grid used (either square or hexagonal)
edge	the kind of edge to use (behavior for pixels near edge depends on it)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_SupNb (MB_Image ∗ *src*, MB_↩ Image ∗ *srcdest*, Uint32 *neighbors*, enum MB_grid_t *grid*, enum MB_edgemode_t *edge*) Looks for the maximum between two image pixels (a central pixel and its neighbors in the other image) The neighbor depends on the grid used. Neighbors are described using a pattern. If no neighbor is defined, the function will leave silently doing nothing.

See also

MB_Neighbors_code_t for encoding neighbors patterns.

Parameters

src	source image in which the neighbor are taken
srcdest	source of the central pixel and destination image
neighbors	the neighbors to take into account
grid	the grid used (either square or hexagonal)
edge	the kind of edge to use (behavior for pixel near edge depends on it)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_SupVector (MB_Image ∗ *src*, MB↩_Image ∗ *srcdest*, Sint32 *dx*, Sint32 *dy*, enum MB_edgemode_t *edge*) Looks for the maximum between two images pixels (a central pixel and its neighbor in the other image previously shifted by the given vector)

Parameters

src	source image in which the neighbor are taken
srcdest	source of the central pixel and destination image
dx	the vector amplitude in x
dy	the vector amplitude in y
edge	the kind of edge to use (behavior for pixels near edge depends on it)

Returns

An error code (NO_ERR if successful)

4.6 mamba/MB_Common.h File Reference

`#include <stdint.h>`

Data Structures

- struct MB_Image
- struct MB3D_Image

Macros

- #define **MB_API_ENTRY**
- #define **MB_API_CALL**
- #define MB_ROUND_W 64
- #define MB_ROUND_H 2
- #define MB_MAX_IMAGE_SIZE ((Uint64)4294967296)

Typedefs

- typedef uint8_t Uint8
- typedef uint16_t Uint16
- typedef uint32_t Uint32
- typedef uint64_t Uint64
- typedef int8_t Sint8
- typedef int16_t Sint16
- typedef int32_t Sint32
- typedef int64_t Sint64
- typedef uint8_t PIX8

- typedef PIX8 ∗ PLINE
- typedef uint32_t PIX32
- typedef PIX32 ∗ PLINE32

Enumerations

- enum MB_grid_t { MB_HEXAGONAL_GRID = 1, MB_SQUARE_GRID = 0 }
- enum MB_edgemode_t { MB_EMPTY_EDGE = 0, MB_FILLED_EDGE = 1 }
- enum MB3D_grid_t { MB3D_INVALID_GRID = -1, MB3D_CUBIC_GRID = 1024, MB3D_FCC←_GRID = 1025 }
- enum MB_Neighbors_code_t {
  MB_NEIGHBOR_0 = 0x0001, **MB_NEIGHBOR_1** = 0x0002, **MB_NEIGHBOR_2** = 0x0004,
  MB_NEIGHBOR_3 = 0x0008,
  MB_NEIGHBOR_4 = 0x0010, **MB_NEIGHBOR_5** = 0x0020, **MB_NEIGHBOR_6** = 0x0040,
  MB_NEIGHBOR_7 = 0x0080,
  MB_NEIGHBOR_8 = 0x0100, **MB_NEIGHBOR_ALL_HEXAGONAL** = 0x07f, **MB_N←EIGHBOR_ALL_SQUARE** = 0x01ff }

4.6.1 Macro Definition Documentation

#define MB_MAX_IMAGE_SIZE ((Uint64)4294967296) Image limit size in total number of pixels. When considering the limits on the image size, remember that the function computing the volume which returns Uint64 should not overflow on the 32-bit images. (that is, max volume for the 32-bit image is 2^64-1) which yields approx 4.3 billions pixels so roughly 65536∗65536 images size. However, if we compute a watershed transform, the number of allowed labels is 2^24 (3 lower bytes of the label image). Therefore, if, in a large image, the number of labels exceeds this value, some basins of the watershed transform will share the same label. You must be aware of this possibility.

#define MB_ROUND_H 2 Making sure the image size is multiple of 2 for the height.

#define MB_ROUND_W 64 Making sure the image size is multiple of 64 for the width.

4.6.2 Typedef Documentation

typedef uint32_t PIX32 Signed 32-bit pixels value type

typedef uint8_t PIX8 grey-scale pixels value type

typedef PIX8∗ PLINE Pixels line pointers type

typedef PIX32∗ PLINE32 32-bit pixels line pointers type

typedef int16_t Sint16 Signed 16-bit value type

typedef int32_t Sint32 Signed 32-bit value type

typedef int64_t Sint64 Signed 64-bit value type

typedef int8_t Sint8 Signed 8-bit value type

typedef uint16_t Uint16 Unsigned 16-bit value type

typedef uint32_t Uint32 Unsigned 32-bit value type

typedef uint64_t Uint64 Unsigned 64-bit value type

typedef uint8_t Uint8 Unsigned 8-bit value type

4.6.3 Enumeration Type Documentation

enum MB3D_grid_t Possible 3D grid values: Values are specificly chosen not to match 2D grid values.

Enumerator

MB3D_INVALID_GRID Invalid grid

MB3D_CUBIC_GRID Cubic grid

MB3D_FCC_GRID Face centered cubic grid (fcc, also known as cubic close-packed or ccp)

enum MB_edgemode_t Possible edge modes:

Enumerator

MB_EMPTY_EDGE Empty edge (zero)

MB_FILLED_EDGE Filled edge (maximum value for a given depth)

enum MB_grid_t Possible grid values:

Enumerator

MB_HEXAGONAL_GRID Hexagonal grid

MB_SQUARE_GRID Square grid

enum MB_Neighbors_code_t Neighbors encoding:

4.7 mamba/MB_Error.h File Reference

Enumerations

- enum MB_errcode {
  MB_NO_ERR, MB_ERR_BAD_SIZE, MB_ERR_BAD_DEPTH, MB_ERR_BAD_PARAMET↩
  ER,
  MB_ERR_BAD_VALUE, MB_ERR_BAD_DIRECTION, MB_ERR_CANT_ALLOCATE_MEM↩
  ORY, MB_ERR_BAD_IMAGE_DIMENSIONS,
  MB_ERR_LOAD_DATA, MB_ERR_UNKNOWN }

Functions

- MB_API_ENTRY char ∗MB_API_CALL MB_StrErr (MB_errcode error_nb)

4.7.1 Enumeration Type Documentation

enum MB_errcode Type definition for error code.

Enumerator

MB_NO_ERR Value returned by function when no error was encountered.

MB_ERR_BAD_SIZE Value returned by function when an error of size inside the image was encountered. For example, if the width of a first image was not corresponding with the width of a second image when both are used in the same computations.

MB_ERR_BAD_DEPTH Value returned by function when an error of depth inside the image was encountered. For example, the function expected 8-bit pixels and got 1-bit pixels.

MB_ERR_BAD_PARAMETER Value returned by function when a given parameter was incorrect.

MB_ERR_BAD_VALUE Value returned by function when a given value (as argument) was incorrect.

MB_ERR_BAD_DIRECTION Value returned when a function requiring a direction (shift, neighbor) is given an incorrect argument for direction (allowed value depends on the grid).

MB_ERR_CANT_ALLOCATE_MEMORY Value returned when the allocation for image memory failed.

MB_ERR_BAD_IMAGE_DIMENSIONS Value returned when the dimension of the image given in argument of a function is incorrect.

MB_ERR_LOAD_DATA Value returned when the data given to a load function is incorrect (size or type).

MB_ERR_UNKNOWN Value never returned (reserved for the error function).

4.7.2 Function Documentation

MB_API_ENTRY char∗ MB_API_CALL MB_StrErr (MB_errcode *error_nb*) Returns an explanation of the error code.

Parameters

error_nb	the error code number

4.8 mamba/MB_Image.h File Reference

Functions

- MB_API_ENTRY MB_errcode MB_API_CALL MB_Create (MB_Image ∗image, Uint32 width, Uint32 height, Uint32 depth)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Destroy (MB_Image ∗image)
- MB_API_ENTRY Uint32 MB_API_CALL MB_getImageCounter (void)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Load (MB_Image ∗image, PIX8 ∗indata, Uint32 len)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Extract (MB_Image ∗image, PIX8 ∗∗outdata, Uint32 ∗len)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Convert (MB_Image ∗src, MB_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_Copy (MB_Image ∗src, MB_Image ∗dest)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_CopyLine (MB_Image ∗src, MB_Image ∗dest, Uint32 insrc_pos, Uint32 indest_pos)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_CropCopy (MB_Image ∗src, Uint32 x_src, Uint32 y_src, MB_Image ∗dest, Uint32 x_dest, Uint32 y_dest, Uint32 w, Uint32 h)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_PutPixel (MB_Image ∗dest, Uint32 pixVal, Uint32 x, Uint32 y)
- MB_API_ENTRY MB_errcode MB_API_CALL MB_GetPixel (MB_Image ∗src, Uint32 ∗pixVal, Uint32 x, Uint32 y)
- MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Create (MB3D_Image ∗image, Uint32 length)
- MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Stack (MB3D_Image ∗image, MB_Image ∗stacked, Uint32 position)
- MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Destroy (MB3D_Image ∗image)
- MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Convert (MB3D_Image ∗src, MB3D_Image ∗dest)

4.8.1 Function Documentation

MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Convert (MB3D_Image ∗ *src*, MB3D_Image ∗ *dest*) Converts a 3D image of a given depth into another depth. Supported conversions are: 1->8, 8->1 and 32->8 (downscaling).

Parameters

src	3D source image
dest	3D destination image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Create (MB3D_Image ∗ *image*, Uint32 *length*) Creates a 3D image container. 3D images are just a list of 2D images stacked together.

Parameters

image	the created image
length	the length of the 3D image (number of images stacked)

Returns

an error code (NO_ERR if everything went OK).

MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Destroy (MB3D_Image ∗ *image*) Destroys the 3D image (free memory).

Parameters

image	the image to destroy

Returns

an error code.

MB_API_ENTRY MB_errcode MB_API_CALL MB3D_Stack (MB3D_Image ∗ *image*, MB_Image ∗ *stacked*, Uint32 *position*) Stack the 2D image at the given position.

Parameters

image	the 3D image
stacked	the 2D image stacked in the 3D image
position	the position in the 3D image of the 2D image

Returns

an error code (NO_ERR if everything went OK).

MB_API_ENTRY MB_errcode MB_API_CALL MB_Convert (MB_Image ∗ *src*, MB↩ _Image ∗ *dest*) Converts an image of a given depth into another depth. All possible conversions are supported.

Parameters

src	source image
dest	destination image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Copy (MB_Image ∗ *src*, MB_Image ∗ *dest*) Copies an image data contents into another image. This copy works with same size images.

Parameters

src	the source image
dest	the destination image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_CopyLine (MB_Image ∗ *src*, MB↩ _Image ∗ *dest*, Uint32 *insrc_pos*, Uint32 *indest_pos*) Copies an image line contents into another image line.

Parameters

src	the source image
dest	the destination image
insrc_pos	the position of the line copied from src
indest_pos	the position in dest in which the line is copied

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Create (MB_Image ∗ *image*, Uint32 *width*, Uint32 *height*, Uint32 *depth*) Creates an image (memory allocation) with the correct size and depth given as argument. The size is deduced from the requested size given in argument. The size must be a multiple of MB_ROUND_W for width and MB_ROUND_H for height. The size cannot be greater than MB_MAX_IMAGE_SIZE.

Parameters

image	the created image
width	the width of the created image
height	the height of the created image
depth	the depth of the created image

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_CropCopy (MB_Image ∗ *src*, Uint32 *x_src*, Uint32 *y_src*, MB_Image ∗ *dest*, Uint32 *x_dest*, Uint32 *y_dest*, Uint32 *w*, Uint32 *h*) Copies an image data contents into another image. This copy can work with image of different sizes. As the size can be different the position where the copy occurs must be specified for both images as well as the size of the copy. The function will compute the actual crop inside the source and destination images. Works only with non binary images.

Parameters

src	the source image
x_src	the x position in the source image where the copy should begin
y_src	the y position in the source image where the copy should begin
dest	the destination image
x_dest	the x position in the destination image where the copy should happen
y_dest	the y position in the destination image where the copy should happen
w	the width of the copy
h	the height of the copy

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Destroy (MB_Image ∗ *image*) Destroys an image (memory freeing).

Parameters

image	the image to be destroyed

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Extract (MB_Image ∗ *image*, PIX8 ∗∗ *outdata*, Uint32 ∗ *len*) Reads an image data contents and put it in an array.

Parameters

image	the image to read
outdata	pointer to the array created (malloc) and filled with the pixel data of the image
len	the length in bytes of data extracted (0 if an error occured)

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY Uint32 MB_API_CALL MB_getImageCounter (void)

Returns

the number of image that have been allocated so far

MB_API_ENTRY MB_errcode MB_API_CALL MB_GetPixel (MB_Image ∗ *src*, Uint32 ∗ *pixVal*, Uint32 *x*, Uint32 *y*) Gets the pixel value inside the image at the given position.

Parameters

src	the image
pixVal	the returned pixel value
x	position in x of the pixel targeted
y	position in y of the pixel targeted

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_Load (MB_Image ∗ *image*, PIX8 ∗ *indata*, Uint32 *len*) Loads an image data with data given in argument.

Parameters

image	the image to fill
indata	the data to fill the image with (complete pixels values)
len	the length of data given

Returns

An error code (NO_ERR if successful)

MB_API_ENTRY MB_errcode MB_API_CALL MB_PutPixel (MB_Image ∗ *dest*, Uint32 *pixVal*, Uint32 *x*, Uint32 *y*) Puts the pixel value inside the image at the given position.

Parameters

dest	the image
pixVal	the pixel value
x	position in x of the pixel targeted
y	position in y of the pixel targeted

Returns

An error code (NO_ERR if successful)