MinImgAPI Library Reference

version 2.5.0

Generated by Doxygen 1.8.13

Contents

1 Overview

MinImgAPI is an open-source platform-independent library that contains image processing functions which treat the image as a matrix. That is, these functions know nothing about "pixel" essence. Examples of such functions are: allocation memory for image data, copying images, rotating an image by right angle and others.

For the internal representation of images is used cross-platform open-source container - MinImg (see MinUtils_↩ MinImg section for more information). The advantages of this container are the using a minimal number of fields needed to represent the bitmap image and the easy way to cast it to other standard and popular containers (for instance, Windows DIB, GDI+ BitmapData, Intel/OpenCV IplImage).

The library is written in C++ and can be compiled under Linux (GCC) and Windows (MSVC 8 and later). Though the library has been written in C++, it has C interface, so it can be embedded in different systems.

2 Quick Tutorial

This tutorial is intended to get you start using **MinImgAPI** library. The tutorial demonstrates popular use cases of library usages, therefore it is not a complete or detailed documentation. Note also, that some secondary operations will be purposely omitted for brevity.

2.1 Allocate and Deallocate Images

This is the most popular use case of usage the library. To do that you should define image header at first and then allocate memory for image data. The following example shows the way to allocate 24-bit RGB image of 640x480 size:

```
// Define header
MinImg image = {0};
image.width = 640;
image.height = 480;
image.channels = 3;
image.channelDepth = 1;
image.format = FMT_UINT;

// Allocates the memory for the image data
PROPAGATE_ERROR(AllocMinImage(&image, 16));
```

If you use `AllocMinImage()` for allocation of memory then you **must** use `FreeMinImage()` to deallocate that. The following example demonstrates the usage of `FreeMinImage()` function:

```
PROPAGATE_ERROR(FreeMinImage(&image));
```

2.2 Copy Images

Another popular use case is cloning the image. Let we have `sourceImage` and want to clone it. The following code shows the proper way to do that:

```
// Define clone image
MinImg cloneImage = {0};

// Make a copy of the header and allocate it
PROPAGATE_ERROR(CloneMinImagePrototype(&cloneImage, &sourceImage));

// Copy image data
PROPAGATE_ERROR(CopyMinImage(&cloneImage, &sourceImage));
```

3 MinImgAPI License Agreements

3.1 Library License Agreement

MinImgAPI is released under FreeBSD License. It is free for both academic and commercial use.

3.2 Documentation License Agreement

This documentation is released under FreeBSD Documentation License. It is free for both academic and commercial use.

4 Deprecated List

Class imgGuard

This class is deprecated now, it is better to use `MinImgGuard`.

5 Module Documentation

5.1 MinImgAPI Library API

This section describes an application programming interface (API) of **MinImgAPI** library. Though **MinImgAPI** has been written in C++, it has C interface to make it easy embedding the different systems.

Macros

- #define MINIMGAPI_API

  Specifies storage-class information (only for MSC).
- #define GET_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) & (0x80U >> ((x) & 7)))

  Returns value of x-th bit of image line pointed by p. If bit is on, returns not just 1, but returns it in the same position within byte, in which it was within the image byte.
- #define SET_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) |= (0x80U >> ((x) & 7)))

  Sets x-th bit of image line pointed by p to be 1.
- #define CLEAR_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) &= (0xFF7FU >> ((x) & 7)))

  Sets x-th bit of image line pointed by p to be 0.
- #define INVERT_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) ^= (0x80U >> ((x) & 7)))

  Switches the value of x-th bit of image line pointed by p.

Functions

- template<typename Type >
  MUSTINLINE MinFmt GetMinFmtByCType ()

  Returns format that corresponds to the given template parameter type.
- template<typename Type >
  MUSTINLINE MinTyp GetMinTypByCType ()

  Returns type (MinTyp value) that corresponds to the given template parameter type.
- int NewMinImagePrototype (MinImg *p_image, int width, int height, int channels, MinTyp element_type, int address_space=0, AllocationOption allocation=AO_PREALLOCATED)

  Makes new MinImg, allocated or not.
- int AllocMinImage (MinImg *p_image, int alignment=16)

  Allocates an image.
- int FreeMinImage (MinImg *p_image)

  Deallocates an image.
- int CloneMinImagePrototype (MinImg *p_dst_image, const MinImg *p_src_image, AllocationOption allocation=AO_PREALLOCATED)

  Makes a copy of the image header.
- int CloneTransposedMinImagePrototype (MinImg *p_dst_image, const MinImg *p_src_image, AllocationOption allocation=AO_PREALLOCATED)

  Makes a copy of the transposed image header.
- int CloneRetypifiedMinImagePrototype (MinImg *p_dst_image, const MinImg *p_src_image, MinTyp type, AllocationOption allocation=AO_PREALLOCATED)

  Makes a copy of the image header with another type (MinTyp value).
- int CloneDimensionedMinImagePrototype (MinImg *p_dst_image, const MinImg *p_src_image, int channels, AllocationOption allocation=AO_PREALLOCATED)

  Makes a copy of the image header with another number of channels.
- int CloneResizedMinImagePrototype (MinImg *p_dst_image, const MinImg *p_src_image, int width, int height, AllocationOption allocation=AO_PREALLOCATED)

Makes a copy of the image header with another size.

- int WrapScalarWithMinImage (MinImg *p_image, void *p_scalar, MinTyp element_type, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user scalar.
- int WrapPixelWithMinImage (MinImg *p_image, void *p_pixel, int channels, MinTyp element_type, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user pixel.
- int WrapScalarVectorWithMinImage (MinImg *p_image, void *p_vector, int size, DirectionOption direction, MinTyp element_type, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user vector of scalars.
- int WrapPixelVectorWithMinImage (MinImg *p_image, void *p_vector, int size, DirectionOption direction, int channels, MinTyp element_type, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user vector of pixels.
- int WrapSolidBufferWithMinImage (MinImg *p_image, void *p_buffer, int width, int height, int channels, MinTyp element_type, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user solid memory buffer.
- int WrapAlignedBufferWithMinImage (MinImg *p_image, void *p_buffer, int width, int height, int channels, MinTyp element_type, int stride, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user memory buffer with fixed stride.
- int GetMinImageRegion (MinImg *p_dst_image, const MinImg *p_src_image, int x0, int y0, int width, int height, RulesOption rules=RO_STRICT)

  Gets a region of an image.
- int FlipMinImageVertically (MinImg *p_dst_image, const MinImg *p_src_image, RulesOption rules=RO_STRICT)

  Flips an image in vertical without copying.
- int UnfoldMinImageChannels (MinImg *p_dst_image, const MinImg *p_src_image, RulesOption rules=RO_STRICT)

  Makes an image header where every pixel element is considered as a separate pixel.
- int SliceMinImageVertically (MinImg *p_dst_image, const MinImg *p_src_image, int begin, int period, int end=-1, RulesOption rules=RO_STRICT)

  Takes a subset of equidistant image lines without copying.
- int UnrollSolidMinImage (MinImg *p_dst_image, const MinImg *p_src_image, RulesOption rules=RO_STRICT)

  Unrolls solid image into one-line image without copying.
- int GetFmtByTyp (MinTyp typ)

  Returns format that corresponds to the given type (MinTyp value).
- int GetDepthByTyp (MinTyp typ)

  Returns channel depth that corresponds to the given type (MinTyp value).
- int GetTypByFmtAndDepth (MinFmt fmt, int depth)

  Returns type (MinTyp value) that corresponds to the given format and channel depth.
- int GetMinImageType (const MinImg *p_image)

  Returns type (MinTyp value) of an image channel element.
- int SetMinImageType (MinImg *p_image, MinTyp element_type)

  Assigns type (MinTyp value) to the image.
- int GetMinImageBitsPerPixel (const MinImg *p_image)

  Returns the amount of bits in one pixel of image.
- int GetMinImageBytesPerLine (const MinImg *p_image)

  Returns the amount of bytes in one line of an image.
- int AssureMinImagePrototypeIsValid (const MinImg *p_image)

  Checks whether image prototype is valid or not.
- int AssureMinImageIsValid (const MinImg *p_image)

  Checks whether image is valid or not.

Deinterleaves pixels of the source image in the resulting images.

- int ResampleMinImage (const MinImg *p_dst_image, const MinImg *p_src_image, double x_phase=0.5, double y_phase=0.5)

  Changes image sample rate.

5.1.1 Detailed Description

5.1.2 Function Documentation

5.1.2.1 AllocMinImage()

```
int AllocMinImage (
            MinImg * p_image,
            int alignment = 16 )
```

Parameters

p_image	The image to be allocated.
alignment	Alignment for image rows, by default 16 bytes.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function allocates the memory for the image data. The memory block size to allocate is specified by the "header fields" of the p_image. On success the function updates p_image->pScan0 and p_image->stride fields in accordance with allocated memory block. Function fails if p_image->pScan0 is not NULL.

5.1.2.2 AssureMinImageFits()

```
int AssureMinImageFits (
            const MinImg * p_image,
            MinTyp element_type,
            int channels = -1,
            int width = -1,
            int height = -1 )
```

Parameters

p_image	The image.
element_type	type (MinTyp value) of element to be checked for, or -1 for not specified. channels amount of channels in image pixel, or -1 for not specified. width width of the image, or -1 for not specified. height height of the image, or -1 for not specified.

Returns

NO_ERRORS if image fits given parameters, >0 if image doesn't fit given parameters, or negative error code (see #MinErr).

This function checks the image header for congruence with given parameters. Parameters given as -1 are considered not restricted and are not checked. Note, that the function returns NO_ERRORS on success, which should not be wrongly interpreted as false.

5.1.2.3 AssureMinImageIsEmpty()

```
int AssureMinImageIsEmpty (
            const MinImg * p_image )
```

Parameters

p_image	The image.

Returns

NO_ERRORS if image is valid and empty, >0 if image is valid but not empty, or BAD_ARGS otherwise.

The function checks image to be valid, then analyzes p_image->width, p_image->height and p_image->channels fields to check if image has at least one element. Note, that the function returns NO_ERRORS on success, which should not be wrongly interpreted as false.

5.1.2.4 AssureMinImageIsPixel()

```
int AssureMinImageIsPixel (
            const MinImg * p_image )
```

Parameters

p_image	The image.

Returns

NO_ERRORS if image has exactly one pixel, >0 if image has not exactly one pixel, or negative error code (see #MinErr).

This function analyzes p_image->width and p_image->height fields to check whether image has exactly one pixel or not.

5.1.2.5 AssureMinImageIsScalar()

```
int AssureMinImageIsScalar (
            const MinImg * p_image )
```

Parameters

p_image	The image.

Returns

NO_ERRORS if image has exactly one element, >0 if image has not exactly one element, or negative error code (see #MinErr).

This function analyzes p_image->width, p_image->height and p_image->channels fields to check whether image has exactly one element (one-channeled pixel) or not.

5.1.2.6 AssureMinImageIsSolid()

```
int AssureMinImageIsSolid (
            const MinImg * p_image )
```

Parameters

p_image	The image.

Returns

NO_ERRORS if image buffer is contiguous, >0 if image buffer is not contiguous, or negative error code (see #MinErr).

This function checks if bits of image form solid memory chunk. Note, that the function returns NO_ERRORS on success, which should not be wrongly interpreted as false. Note that images with negative stride is always treated as NOT solid due to safety reasons: even if memory chunk is contiguous, it starts not from p_image->pScan0 address.

5.1.2.7 AssureMinImageIsValid()

```
int AssureMinImageIsValid (
            const MinImg * p_image )
```

Parameters

p_image	The image.

Returns

NO_ERRORS if image is valid or BAD_ARGS otherwise.

The function checks image to have valid prototype and, if image has at least one element, checks p_image->p↩Scan0 to be non-zero and image lines to not intersect in memory (analyzing p_image->stride and amount of bytes per image line). Note, that the function returns NO_ERRORS on success, which should not be wrongly interpreted as false.

5.1.2.8 AssureMinImagePrototypeIsValid()

```
int AssureMinImagePrototypeIsValid (
            const MinImg * p_image )
```

Parameters

p_image	The image.

Returns

NO_ERRORS if image prototype is valid or BAD_ARGS otherwise.

The function checks p_image->format and p_image->channelDepth to have one of provided values (see #MinFmt and #MinTyp), then checks p_image->width, p_image->height and p_image->channels fields to be non-negative. Note, that the function returns NO_ERRORS on success, which should not be wrongly interpreted as false.

5.1.2.9 CheckMinImagesTangle()

```
int CheckMinImagesTangle (
            uint32_t * p_result,
            const MinImg * p_dst_image,
            const MinImg * p_src_image )
```

Parameters

p_result	Variable to place result to.
p_dst_image	The image to be destination in later manipulations.
p_src_image	The image to be source in later manipulations.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function checks how two images are placed in memory with respect to each other, treating one image as source for later manipulations and another one as destination, describing the ways of manipulating that are possible without damaging images' contents (see TangleCheckResult).

5.1.2.10 CloneDimensionedMinImagePrototype()

```
int CloneDimensionedMinImagePrototype (
            MinImg * p_dst_image,
            const MinImg * p_src_image,
            int channels,
            AllocationOption allocation = AO_PREALLOCATED )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
channels	The required number of the destination image channels.
allocation	Specifies whether the destination image should be allocated.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function makes a full copy of the image header with required number of channels. If allocation is set to AO_PREALLOCATED (the default) then a new image will also be allocated.

5.1.2.11 CloneMinImagePrototype()

```
int CloneMinImagePrototype (
            MinImg * p_dst_image,
            const MinImg * p_src_image,
            AllocationOption allocation = AO_PREALLOCATED )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
allocation	Specifies whether the destination image should be allocated.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function makes a full copy of the image header. If allocation is set to AO_PREALLOCATED (the default) then a new image will also be allocated.

5.1.2.12 CloneResizedMinImagePrototype()

```
int CloneResizedMinImagePrototype (
            MinImg * p_dst_image,
            const MinImg * p_src_image,
            int width,
            int height,
            AllocationOption allocation = AO_PREALLOCATED )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
width	The required width of the destination image.
height	The required height of the destination image.
allocation	Specifies whether the destination image should be allocated.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function makes a full copy of the image header with required size. If allocation is set to AO_PREALLO↩CATED (the default) then a new image will also be allocated.

5.1.2.13 CloneRetypifiedMinImagePrototype()

```
int CloneRetypifiedMinImagePrototype (
            MinImg * p_dst_image,
            const MinImg * p_src_image,
            MinTyp type,
            AllocationOption allocation = AO_PREALLOCATED )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
type	The required type (MinTyp value) of the destination image.
allocation	Specifies whether the destination image should be allocated.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function makes a full copy of the image header with required type (MinTyp value). If allocation is set to AO_PREALLOCATED (the default) then a new image will also be allocated.

5.1.2.14 CloneTransposedMinImagePrototype()

```
int CloneTransposedMinImagePrototype (
            MinImg * p_dst_image,
            const MinImg * p_src_image,
            AllocationOption allocation = AO_PREALLOCATED )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
allocation	Specifies whether the destination image should be allocated.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function makes a full copy of the transposed image header (that is, p_dst_image->width = p_src↩ _image->height and p_dst_image->height = p_src_image->width). If allocation is set to AO_PREALLOCATED (the default) then a new image will also be allocated.

5.1.2.15 CompareMinImage2DSizes()

```
int CompareMinImage2DSizes (
            const MinImg * p_image_a,
            const MinImg * p_image_b )
```

Parameters

p_image↩ _a	First image.
p_image↩ _b	Second image.

Returns

Zero if the sizes are equal, a positive value if they are not or negative error code (see `#MinErr`).

The function compares width and height of the source and destination images. It does not matter whether the images are allocated or not.

5.1.2.16 CompareMinImage3DSizes()

```
int CompareMinImage3DSizes (
            const MinImg * p_image_a,
            const MinImg * p_image_b )
```

Parameters

p_image_a	First image.
p_image_b	Second image.

Returns

Zero if the sizes are equal, a positive value if they are not or negative error code (see `#MinErr`).

The function compares width, height and number of channels of the source and destination images. It does not matter whether the images are allocated or not.

5.1.2.17 CompareMinImagePixels()

```
int CompareMinImagePixels (
            const MinImg * p_image_a,
            const MinImg * p_image_b )
```

Parameters

p_image_a	First image.
p_image_b	Second image.

Returns

Zero if the pixel types (MinTyp values) are the same, a positive value if they are not or negative error code (see `MinErr`).

The function compares types (MinTyp values) of elements (channels) and number of channels in pixels of two images. It does not matter whether the images are allocated or not.

5.1.2.18 CompareMinImagePrototypes()

```
int CompareMinImagePrototypes (
            const MinImg * p_image_a,
            const MinImg * p_image_b )
```

Parameters

p_image_a	First image.
p_image_b	Second image.

Returns

Zero if the headers are equal, a positive value if they are not or negative error code (see `#MinErr`).

The function just compares the header information (that is width, height, number of channels, channel depth and format) of the source and destination images. It does not matter whether the images are allocated or not.

5.1.2.19 CompareMinImages()

```
int CompareMinImages (
            const MinImg * p_image_a,
            const MinImg * p_image_b )
```

Parameters

p_image_a	First image.
p_image_b	Second image.

Returns

Zero if the images are equal, a positive value if they are not or negative error code (see `#MinErr`).

The function compares two images to be equal in header and contents. Images must be allocated.

5.1.2.20 CompareMinImageTypes()

```
int CompareMinImageTypes (
            const MinImg * p_image_a,
            const MinImg * p_image_b )
```

Parameters

p_image_a	First image.
p_image_b	Second image.

Returns

Zero if the types (MinTyp values) are the same, a positive value if they are not or negative error code (see `MinErr`).

The function compares types (MinTyp values) of elements (channels) of two images. It does not matter whether the images are allocated or not.

5.1.2.21 CopyMinImage()

```
int CopyMinImage (
            const MinImg * p_dst_image,
            const MinImg * p_src_image )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

Remarks

The destination image must be already allocated.
Both source and destination images must have the same size, the same format, and the same number of channels.

The function copies all elements from the source image to the destination one:

$$p_dst_image(i,j) = p_src_image(i,j)$$

5.1.2.22 CopyMinImageChannels()

```
int CopyMinImageChannels (
            const MinImg * p_dst_image,
            const MinImg * p_src_image,
            const int * p_dst_channels,
            const int * p_src_channels,
            int num_channels )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
p_dst_channels	0-based destination channel indices.
p_src_channels	0-based source channel indices.
num_channels	The number of channels to copy.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

Remarks

The destination image must be already allocated.
Both source and destination images must have the same size and the same format.

The function copies the specified channels of the source image to the destination one.

5.1.2.23 CopyMinImageFragment()

```
int CopyMinImageFragment (
            const MinImg * p_dst_image,
            const MinImg * p_src_image,
            int dst_x0,
            int dst_y0,
            int src_x0,
            int src_y0,
            int width,
            int height )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
dst_x0	The x-coordinate of the top-left corner of the region of destination image.
dst_y0	The y-coordinate of the top-left corner of the region of destination image.
src_x0	The x-coordinate of the top-left corner of the region of source image.
src_y0	The y-coordinate of the top-left corner of the region of source image.
width	The width of the region.
height	The height of the region.

Returns

`NO_ERRORS` on success or an error code otherwise (see `#MinErr`).

Remarks

The destination image must be already allocated.
Both source and destination images must have the same format and the same number of channels.

The function copies all elements from the region of the source image to the region of the destination one.

5.1.2.24 DeinterleaveMinImage()

```
int DeinterleaveMinImage (
            const MinImg *const * p_p_dst_images,
            const MinImg * p_src_image,
            int num_dst_images )
```

Parameters

p_p_dst_images	The list of destination images.
p_src_image	The source image.
num_dst_images	The number of destination images.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function deinterleaves pixels of the source image into the resulting images. The list of destination images defines proper partition of the channels of the source pixels. The destination pixels of the i-th destination image contains corresponding channels of the source image.

5.1.2.25 FillMinImage()

```
int FillMinImage (
            const MinImg * p_image,
            const void * p_canvas,
            int value_size = 0 )
```

Parameters

p_image	The input image.
p_canvas	The pointer to the fill value.
value_size	The size of the fill value. If it is equal to zero then pixel size is used.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

Remarks

The input image must be already allocated.

The function fills each line of the input image with repeating given value.

The efficient way to fill 1-channel bit image with constant value is to fill the byte pointed by p_canvas with that value and pass 1 as value_size.

5.1.2.26 FlipMinImage()

```
int FlipMinImage (
            const MinImg * p_dst_image,
            const MinImg * p_src_image,
            DirectionOption direction )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image
direction	Specifies how to flip the image (see DirectionOption).

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

Remarks

The destination image must be already allocated.
Both source and destination images must have the same size, the same format, and the same number of channels.

The function flips the image around vertical or horizontal axis. That is $p_src_image(i,j) = p_src_image(p_src_image->height - i - 1, j)$ for vertical flipping and $p_src_image(i,j) = p_src_image(i, p_src_image->width - j - 1)$ for horizontal flipping.

5.1.2.27 FlipMinImageVertically()

```
int FlipMinImageVertically (
            MinImg * p_dst_image,
            const MinImg * p_src_image,
            RulesOption rules = RO_STRICT )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

This function flips the source image in vertical direction. Note, that the function **does not** make a copy of the specified region. Therefore, **it is forbidden** to call FreeMinImage() for the p_dst_image.

5.1.2.28 FreeMinImage()

```
int FreeMinImage (
            MinImg * p_image )
```

Parameters

p_image	The image to be deallocated.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function deallocates the image data and clean p_image->pScan0 and p_image->stride fields.

5.1.2.29 GetDepthByTyp()

```
int GetDepthByTyp (
            MinTyp typ )
```

Parameters

typ	The type (MinTyp value) to get channel depth from (see #MinTyp).

Returns

Appropriate channel depth on success or an error code otherwise (see #MinErr).

The function returns the channel depth that corresponds to the given type (MinTyp value) (see #MinTyp).

5.1.2.30 GetFmtByTyp()

```
int GetFmtByTyp (
            MinTyp typ )
```

Parameters

typ	The type (MinTyp value) to get format from (see #MinTyp).

Returns

Appropriate format on success or an error code otherwise (see #MinErr).

The function returns the format that corresponds to the given type (MinTyp value) (see #MinTyp and #MinFmt).

5.1.2.31 GetMinFmtByCType()

```
template<typename Type >
MUSTINLINE MinFmt GetMinFmtByCType ( )
```

- <Type> parameter type.

  Returns

  Appropriate format (see #MinFmt).

  The function returns the format that corresponds to the given template parameter type (see #MinFmt).

5.1.2.32 GetMinImageBitsPerPixel()

```
int GetMinImageBitsPerPixel (
            const MinImg * p_image )
```

Parameters

p_image	The image.

Returns

amount of bits per image pixel on success or an error code otherwise (see `#MinErr`).

The function analyzes `p_image->channels` and `p_image->channelDepth` fields and returns the amount of bits per image pixel.

5.1.2.33 GetMinImageBytesPerLine()

```
int GetMinImageBytesPerLine (
            const MinImg * p_image )
```

Parameters

p_image	The image.

Returns

amount of bytes per image line on success or an error code otherwise (see `#MinErr`).

The function analyzes `p_image->channels`, `p_image->channelDepth` and `p_image->width` fields and returns the amount of bytes per image line (for bit images incomplete byte counts as the whole one).

5.1.2.34 GetMinImageLine()

```
uint8_t* GetMinImageLine (
            const MinImg * p_image,
            int y,
            BorderOption border = BO_VOID,
            void * p_canvas = NULL )
```

Parameters

p_image	The input image.
y	0-based line index.
border	The border condition (see BorderOption).
p_canvas	The line to be used if the `border` is `BO_CONSTANT`.

Returns

A pointer to the specified line on success or NULL otherwise.

The function returns a pointer to the specified image line. If the `y` is out of the range then the function will return the pointer in accordance with the specified border condition (see BorderOption).

5.1.2.35 GetMinImageRegion()

```
int GetMinImageRegion (
            MinImg * p_dst_image,
            const MinImg * p_src_image,
            int x0,
```

```
            int y0,
            int width,
            int height,
            RulesOption rules = RO_STRICT )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
x0	The x-coordinate of the top-left corner of the region.
y0	The y-coordinate of the top-left corner of the region.
width	The width of the region.
height	The height of the region.
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function get a subimage from the source image. Note, that the function **does not** make a copy of the specified region. Therefore, **it is strongly forbidden** to call FreeMinImage() for the p_dst_image.

5.1.2.36 GetMinImageType()

```
int GetMinImageType (
            const MinImg * p_image )
```

Parameters

p_image	The input image.

Returns

Appropriate image type (MinTyp value) or an error code otherwise (see #MinErr).

The function analyzes p_image->format and p_image->channelDepth fields and returns the type (MinTyp value) of the input image elements (see #MinTyp).

5.1.2.37 GetMinTypByCType()

```
template<typename Type >
MUSTINLINE MinTyp GetMinTypByCType ( )
```

- <Type> parameter type.

  Returns

  Appropriate type (MinTyp value) (see #MinTyp).

  The function returns the type (MinTyp value) that corresponds to the given template parameter type (see #MinTyp).

Definition at line 84 of file minimgapi-helpers.hpp.

5.1.2.38 GetTypByFmtAndDepth()

```
int GetTypByFmtAndDepth (
            MinFmt fmt,
            int depth )
```

Parameters

fmt	The format of number presentation (see #MinFmt).
depth	The byte-depth of number presentation.

Returns

Appropriate type (MinTyp value) on success or an error code otherwise (see #MinErr).

The function returns the type (MinTyp value) that corresponds to the given format and depth (see #MinTyp and #MinFmt).

5.1.2.39 InterleaveMinImages()

```
int InterleaveMinImages (
            const MinImg * p_dst_image,
            const MinImg *const * p_p_src_images,
            int num_src_images )
```

Parameters

p_dst_image	The destination image.
p_p_src_images	The pointers to the source images.
num_src_images	The number of source images.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function interleaves pixels of the source images in the resulting image. The number of channels of a pixel in the resulting image equals therefore to the sum of the number of channels over the list of source images.

5.1.2.40 NewMinImagePrototype()

```
int NewMinImagePrototype (
            MinImg * p_image,
            int width,
            int height,
            int channels,
            MinTyp element_type,
            int address_space = 0,
            AllocationOption allocation = AO_PREALLOCATED )
```

Parameters

p_image	The image.

Parameters

width	Width of the image.
height	Height of the image.
channels	Number of image channels.
element_type	Type (MinTyp value) of the image content.
address_space	Number of the virtual device hosting the image.
allocation	Specifies whether the image should be allocated.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function fills the image header. If allocation is set to AO_PREALLOCATED (the default) then a new image will also be allocated. Function fails if p_image->pScan0 is not NULL.

5.1.2.41 ResampleMinImage()

```
int ResampleMinImage (
            const MinImg * p_dst_image,
            const MinImg * p_src_image,
            double x_phase = 0.5,
            double y_phase = 0.5 )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
x_phase	Horizontal phase of resampling.
y_phase	Vertical phase of resampling.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

Remarks

The destination image must be already allocated.
Both source and destination images must have the same format and the same number of channels.

The function resamples an image in the sense of changing image sample rate. The source image pixels are copied to destination one as whole entities, with no interpolation.

5.1.2.42 RotateMinImageBy90()

```
int RotateMinImageBy90 (
            const MinImg * p_dst_image,
            const MinImg * p_src_image,
            int num_rotations )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
num_rotations	The multiplication factor.

Returns

`NO_ERRORS` on success or an error code otherwise (see `#MinErr`).

Remarks

The destination image must be already allocated.
Both source and destination images must have the same format and the same number of channels.

The function rotates the image clockwise by `num_rotations` * 90 degrees.

5.1.2.43 SetMinImageType()

```
int SetMinImageType (
            MinImg * p_image,
            MinTyp element_type )
```

Parameters

p_image	The input image.
element_type	New image element type (MinTyp value).

Returns

`NO_ERRORS` on success or an error code otherwise (see `#MinErr`).

The function updates `p_image->format` and `p_image->channelDepth` field values according to assignable image element type (MinTyp value).

5.1.2.44 SliceMinImageVertically()

```
int SliceMinImageVertically (
            MinImg * p_dst_image,
            const MinImg * p_src_image,
            int begin,
            int period,
            int end = -1,
            RulesOption rules = RO_STRICT )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
begin	Y coordinate to start with.
period	Distance between lines.
end	Max Y coordinate or -1.
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

This function fills the image header pointed by p_dst_image to describe a subset of equidistant lines of the source image starting with begin, with distance period between them and finishing with y coordinate not greater then end if it is non-negative, or height of the source image otherwise. Note, that the function **does not** make a copy of the specified region. Therefore, **it is forbidden** to call FreeMinImage() for the p_dst_image.

5.1.2.45 TransposeMinImage()

```
int TransposeMinImage (
            const MinImg * p_dst_image,
            const MinImg * p_src_image )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

Remarks

The destination image must be already allocated.
Both source and destination images must have the same format and the same number of channels.

The function transpose the source image:

$$pDst(i,j) = pSrc(j,i)$$

5.1.2.46 UnfoldMinImageChannels()

```
int UnfoldMinImageChannels (
            MinImg * p_dst_image,
            const MinImg * p_src_image,
            RulesOption rules = RO_STRICT )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

This function fills the image header pointed by p_dst_image to describe the same image as p_src_image, but treating every pixel element as a separate pixel. Note, that the function **does not** make a copy of the specified region. Therefore, **it is forbidden** to call FreeMinImage() for the p_dst_image.

5.1.2.47 UnrollSolidMinImage()

```
int UnrollSolidMinImage (
            MinImg * p_dst_image,
            const MinImg * p_src_image,
            RulesOption rules = RO_STRICT )
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

Remarks

The source image must be solid, i.e. it's buffer must be contiguous.

This function fills the image header pointed by p_dst_image to describe the same image as p_src_image as one horizontal line. Note, that the function **does not** make a copy of the specified region. Therefore, **it is forbidden** to call FreeMinImage() for the p_dst_image.

5.1.2.48 WrapAlignedBufferWithMinImage()

```
int WrapAlignedBufferWithMinImage (
            MinImg * p_image,
            void * p_buffer,
            int width,
            int height,
            int channels,
            MinTyp element_type,
            int stride,
            RulesOption rules = RO_STRICT )
```

Parameters

p_image	Pointer to the MinImg structure.
p_buffer	Pointer to the user buffer.
width	Width of the image.
height	Height of the image.
channels	Number of image channels.
element_type	Type (MinTyp value) of the image content.
stride	Stride of the image.
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function fills MinImg structure to represent user memory buffer as image. Content of the user memory is not affected. Ownership of the memory buffer is not affected, so user must deallocate memory same way it was allocated. Function fails if p_image->pScan0 is not NULL.

5.1.2.49 WrapPixelVectorWithMinImage()

```
int WrapPixelVectorWithMinImage (
            MinImg * p_image,
            void * p_vector,
            int size,
            DirectionOption direction,
            int channels,
            MinTyp element_type,
            RulesOption rules = RO_STRICT )
```

Parameters

p_image	Pointer to the MinImg structure.
p_vector	Pointer to the user vector of pixels.
size	Size of the user vector.
direction	Direction of the vector to be in image.
channels	Number of pixel channels.
element_type	Type (MinTyp value) of channel element (pixel element).
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function fills MinImg structure to represent user vector of pixels as one-line image (horizontal or vertical, depending on direction). Content of the user memory is not affected. Ownership of the memory buffer is not affected, so user must deallocate memory same way it was allocated. Function fails if p_image->pScan0 is not NULL.

5.1.2.50 WrapPixelWithMinImage()

```
int WrapPixelWithMinImage (
            MinImg * p_image,
            void * p_pixel,
            int channels,
            MinTyp element_type,
            RulesOption rules = RO_STRICT )
```

Parameters

p_image	Pointer to the MinImg structure.
p_pixel	Pointer to the user pixel.
channels	Number of pixel channels.
element_type	Type (MinTyp value) of channel element (pixel element).
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function fills MinImg structure to represent user pixel as one-pixel image. Content of the user memory is not affected. Ownership of the memory buffer is not affected, so user must deallocate memory same way it was allocated. Function fails if p_image->pScan0 is not NULL.

5.1.2.51 WrapScalarVectorWithMinImage()

```
int WrapScalarVectorWithMinImage (
            MinImg * p_image,
            void * p_vector,
            int size,
            DirectionOption direction,
            MinTyp element_type,
            RulesOption rules = RO_STRICT )
```

Parameters

p_image	Pointer to the MinImg structure.
p_vector	Pointer to the user vector of scalars.
size	Size of the user vector.
direction	Direction of the vector to be in image.
element_type	Type (MinTyp value) of channel element (vector element).
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function fills MinImg structure to represent user vector of scalars as one-line image (horizontal or vertical, depending on direction). Content of the user memory is not affected. Ownership of the memory buffer is not affected, so user must deallocate memory same way it was allocated. Function fails if p_image->pScan0 is not NULL.

5.1.2.52 WrapScalarWithMinImage()

```
int WrapScalarWithMinImage (
            MinImg * p_image,
            void * p_scalar,
            MinTyp element_type,
            RulesOption rules = RO_STRICT )
```

Parameters

p_image	Pointer to the MinImg structure.
p_scalar	Pointer to the user scalar.
element_type	Type (MinTyp value) of channel element (scalar).
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function fills MinImg structure to represent user scalar as one-channel, one-pixel image. Content of the user memory is not affected. Ownership of the memory buffer is not affected, so user must deallocate memory same way it was allocated. Function fails if p_image->pScan0 is not NULL.

5.1.2.53 WrapSolidBufferWithMinImage()

```
int WrapSolidBufferWithMinImage (
            MinImg * p_image,
            void * p_buffer,
            int width,
            int height,
            int channels,
            MinTyp element_type,
            RulesOption rules = RO_STRICT )
```

Parameters

p_image	Pointer to the MinImg structure.
p_buffer	Pointer to the user buffer.
width	Width of the image.
height	Height of the image.
channels	Number of image channels.
element_type	Type (MinTyp value) of the image content.
rules	The degree of validation.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function fills MinImg structure to represent user memory buffer as image. Content of the user memory is not affected. Ownership of the memory buffer is not affected, so user must deallocate memory same way it was allocated. Function fails if p_image->pScan0 is not NULL.

5.1.2.54 ZeroFillMinImage()

```
int ZeroFillMinImage (
            const MinImg * p_image )
```

Parameters

p_image	The input image.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

Remarks

The input image must be already allocated.

The function fills the whole image with zero value.

5.2 MinImgAPI Library Utility

This section describes different utility functions and classes.

Data Structures

- class imgGuard

  Specifies a class which is used to avoid "free image" problems. More...
- class MinImgGuard

  Specifies a class which is used to avoid "free image" problems. Instead of the `imgGuard` class this one stores a reference to the image. More...

Macros

- #define DECLARE_GUARDED_MINIMG(name) MinImg name = {0}; MinImgGuard name##_MinImg↩ Guard(name)

  Declares a new called <name> and the `MinImgGuard` called <name>_MinImgGuard.

5.2.1 Detailed Description

5.2.2 Data Structure Documentation

5.2.2.1 class imgGuard

Deprecated This class is deprecated now, it is better to use `MinImgGuard`.

Definition at line 48 of file imgguard.hpp.

Public Member Functions

- imgGuard (const MinImg &image)

  Constructor. Setups the image.
- virtual ~imgGuard ()

Private Member Functions

- imgGuard (const imgGuard &)

  Forbidden copy constructor.
- imgGuard & operator= (const imgGuard &)

  Forbidden assignment operator.

Private Attributes

- MinImg image

  The image to be freed while a function exit.

Constructor & Destructor Documentation

5.2.2.1.1 ~imgGuard()

```
virtual imgGuard::~imgGuard ( )  [inline], [virtual]
```

< Destructor. Frees the image.

Definition at line 53 of file imgguard.hpp.

5.2.2.2 class MinImgGuard

Definition at line 68 of file imgguard.hpp.

Public Member Functions

- MinImgGuard (MinImg &image)
  Constructor. Setups the image.
- virtual ~MinImgGuard ()

Private Attributes

- MinImg & image
  The reference to the image to be freed.

Constructor & Destructor Documentation

5.2.2.2.1 ~MinImgGuard()

```
virtual MinImgGuard::~MinImgGuard ( )  [inline], [virtual]
```

< Destructor. Frees the image.

Definition at line 73 of file imgguard.hpp.

6 File Documentation

6.1 imgguard.hpp File Reference

Definition of utility classes.

Data Structures

- class imgGuard

  Specifies a class which is used to avoid "free image" problems. More...
- class MinImgGuard

  Specifies a class which is used to avoid "free image" problems. Instead of the `imgGuard` class this one stores a reference to the image. More...

Macros

- #define **MINIMGAPI_IMGGUARD_HPP_INCLUDED**
- #define DECLARE_GUARDED_MINIMG(name) MinImg name = {0}; MinImgGuard name##_MinImg↩ Guard(name)

  Declares a new called <name> and the `MinImgGuard` called <name>_MinImgGuard.

6.2 imgguard.hpp

```
00001 /*
00002 Copyright (c) 2011-2013, Smart Engines Limited. All rights reserved.
00003
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without modification,
00007 are permitted provided that the following conditions are met:
00008
00009    1. Redistributions of source code must retain the above copyright notice,
00010       this list of conditions and the following disclaimer.
00011
00012    2. Redistributions in binary form must reproduce the above copyright notice,
00013       this list of conditions and the following disclaimer in the documentation
00014       and/or other materials provided with the distribution.
00015
00016 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS "AS IS" AND ANY EXPRESS OR
00017 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00018 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00019 SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
00020 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00021 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00022 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00023 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00024 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00025 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00026
00027 The views and conclusions contained in the software and documentation are those
00028 of the authors and should not be interpreted as representing official policies,
00029 either expressed or implied, of copyright holders.
00030 */
00031
00037 #pragma once
00038 #ifndef MINIMGAPI_IMGGUARD_HPP_INCLUDED
00039 #define MINIMGAPI_IMGGUARD_HPP_INCLUDED
00040
00041 #include <minimgapi/minimgapi.h>
00042
00048 class imgGuard {
00049 public:
00051   imgGuard(const MinImg &image) : image(image) {
00052   }
00053   virtual ~imgGuard() {
00054     FreeMinImage(&image);
```

```
00055   }
00056 private:
00057   imgGuard(const imgGuard &);
00058   imgGuard &operator =(const imgGuard &);
00059   MinImg image;
00060 };
00061
00068 class MinImgGuard {
00069 public:
00071   MinImgGuard(MinImg &image) : image(image) {
00072   }
00073   virtual ~MinImgGuard() {
00074     FreeMinImage(&image);
00075   }
00076 private:
00077   MinImg &image;
00078 };
00079
00085 #define DECLARE_GUARDED_MINIMG(name) \
00086   MinImg name = {0}; MinImgGuard name##_MinImgGuard(name)
00087
00088 #endif // #ifndef MINIMGAPI_IMGGUARD_HPP_INCLUDED
```

6.3 minimgapi-helpers.hpp File Reference

MinImgAPI c++ helpers interface.

Functions

- template<typename Type >
  MUSTINLINE MinFmt GetMinFmtByCType ()

  Returns format that corresponds to the given template parameter type.
- template<typename Type >
  MUSTINLINE MinTyp GetMinTypByCType ()

  Returns type (MinTyp value) that corresponds to the given template parameter type.
- template<typename T >
  MUSTINLINE T * **GetMinImageLineAs** (const MinImg *image, int y, BorderOption border=BO_VOID, T *p↩
  _canvas=NULL)
- static MUSTINLINE int AssignMinImage (MinImg *p_dst_image, const MinImg *p_src_image)

  Properly copies image to another one regardless of destination image allocation status.

6.3.1 Function Documentation

6.3.1.1 AssignMinImage()

```
static MUSTINLINE int AssignMinImage (
            MinImg * p_dst_image,
            const MinImg * p_src_image )  [static]
```

Parameters

p_dst_image	The destination image.
p_src_image	The source image.

Returns

NO_ERRORS on success or an error code otherwise (see #MinErr).

The function copies p_src_image to p_dst_image whatever p_dst_image status is. If p_dst_image is already allocated it's freed and then allocated again.

Definition at line 114 of file minimgapi-helpers.hpp.

6.4 minimgapi-helpers.hpp

```
00001 /*
00002 Copyright (c) 2011-2013, Smart Engines Limited. All rights reserved.
00003
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without modification,
00007 are permitted provided that the following conditions are met:
00008
00009    1. Redistributions of source code must retain the above copyright notice,
00010       this list of conditions and the following disclaimer.
00011
00012    2. Redistributions in binary form must reproduce the above copyright notice,
00013       this list of conditions and the following disclaimer in the documentation
00014       and/or other materials provided with the distribution.
00015
00016 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS "AS IS" AND ANY EXPRESS OR
00017 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00018 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00019 SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
00020 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00021 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00022 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00023 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00024 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00025 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00026
00027 The views and conclusions contained in the software and documentation are those
00028 of the authors and should not be interpreted as representing official policies,
00029 either expressed or implied, of copyright holders.
00030 */
00031
00037 #pragma once
00038 #ifndef MINIMGAPI_MINIMGAPI_HELPERS_HPP_INCLUDED
00039 #define MINIMGAPI_MINIMGAPI_HELPERS_HPP_INCLUDED
00040
00041 #include <minbase/minresult.h>
00042 #include <minbase/crossplat.h>
00043 #include <minimgapi/minimgapi.h>
00044 #include <minimgapi/minimgapi-inl.h>
00045
00057 template<typename Type> MUSTINLINE MinFmt GetMinFmtByCType();
00058
00060 template<> MinFmt MUSTINLINE GetMinFmtByCType<uint8_t>()  { return FMT_UINT; }
00061 template<> MinFmt MUSTINLINE GetMinFmtByCType<uint16_t>() { return FMT_UINT; }
00062 template<> MinFmt MUSTINLINE GetMinFmtByCType<uint32_t>() { return FMT_UINT; }
00063 template<> MinFmt MUSTINLINE GetMinFmtByCType<uint64_t>() { return FMT_UINT; }
00064 template<> MinFmt MUSTINLINE GetMinFmtByCType<int8_t>()   { return FMT_INT;  }
00065 template<> MinFmt MUSTINLINE GetMinFmtByCType<int16_t>()  { return FMT_INT;  }
00066 template<> MinFmt MUSTINLINE GetMinFmtByCType<int32_t>()  { return FMT_INT;  }
00067 template<> MinFmt MUSTINLINE GetMinFmtByCType<int64_t>()  { return FMT_INT;  }
00068 template<> MinFmt MUSTINLINE GetMinFmtByCType<real16_t>() { return FMT_REAL; }
00069 template<> MinFmt MUSTINLINE GetMinFmtByCType<real32_t>() { return FMT_REAL; }
00070 template<> MinFmt MUSTINLINE GetMinFmtByCType<real64_t>() { return FMT_REAL; }
00072
00084 template<typename Type> MUSTINLINE MinTyp GetMinTypByCType() {
00085   return static_cast<MinTyp>(_GetTypByFmtAndDepth(GetMinFmtByCType<Type>(),
00086                                                  sizeof(Type)));
00087 }
00088
00090 template<> MUSTINLINE MinTyp GetMinTypByCType<bool>() {
00091   return TYP_UINT1;
00092 }
00094
00095 template<typename T> MUSTINLINE T *GetMinImageLineAs(
00096     const MinImg* image,
00097     int           y,
00098     BorderOption  border   = BO_VOID,
00099     T*            p_canvas = NULL) {
```

```
00100   return reinterpret_cast<T*>(_GetMinImageLine(image, y, border, p_canvas));
00101 }
00102
00114 static MUSTINLINE int AssignMinImage(
00115     MinImg*       p_dst_image,
00116     const MinImg* p_src_image) {
00117   PROPAGATE_ERROR(FreeMinImage(p_dst_image));
00118   PROPAGATE_ERROR(CloneMinImagePrototype(p_dst_image, p_src_image));
00119   return CopyMinImage(p_dst_image, p_src_image);
00120 }
00121
00122 #endif // #ifndef MINIMGAPI_MINIMGAPI_HELPERS_HPP_INCLUDED
```

6.5 minimgapi-inl.h File Reference

MinImgAPI inlining interface.

Functions

- MUSTINLINE int **_GetFmtByTyp** (MinTyp typ)
- MUSTINLINE int **_GetDepthByTyp** (MinTyp typ)
- MUSTINLINE int **_GetTypByFmtAndDepth** (MinFmt fmt, int depth)
- MUSTINLINE int **_GetMinImageType** (const MinImg *p_image)
- MUSTINLINE int **_SetMinImageType** (MinImg *p_image, MinTyp element_type)
- MUSTINLINE int **_AssureMinImagePrototypeIsValid** (const MinImg *p_image)
- MUSTINLINE int **_GetMinImageBitsPerPixel** (const MinImg *p_image)
- MUSTINLINE int **_GetMinImageBytesPerLine** (const MinImg *p_image)
- MUSTINLINE int **_AssureMinImageIsValid** (const MinImg *p_image)
- MUSTINLINE int **_AssureMinImageIsEmpty** (const MinImg *p_image)
- MUSTINLINE int **_AssureMinImageIsSolid** (const MinImg *p_image)
- MUSTINLINE int **_AssureMinImageFits** (const MinImg *p_image, MinTyp element_type, int channels=-1, int width=-1, int height=-1)
- MUSTINLINE int **_AssureMinImageIsScalar** (const MinImg *p_image)
- MUSTINLINE int **_AssureMinImageIsPixel** (const MinImg *p_image)
- MUSTINLINE uint8_t * **_GetMinImageLineUnsafe** (const MinImg *p_image, int y, BorderOption border=BO_VOID, void *p_canvas=NULL)
- MUSTINLINE uint8_t * **_GetMinImageLine** (const MinImg *p_image, int y, BorderOption border=BO_VOID, void *p_canvas=NULL)
- MUSTINLINE int **_CloneMinImagePrototype** (MinImg *p_dst_image, const MinImg *p_src_image, AllocationOption allocation=AO_PREALLOCATED)
- MUSTINLINE int **_CloneResizedMinImagePrototype** (MinImg *p_dst_image, const MinImg *p_src_image, int width, int height, AllocationOption allocation=AO_PREALLOCATED)
- MUSTINLINE int **_CloneTransposedMinImagePrototype** (MinImg *p_dst_image, const MinImg *p_src_image, AllocationOption allocation=AO_PREALLOCATED)
- MUSTINLINE int **_CloneRetypifiedMinImagePrototype** (MinImg *p_dst_image, const MinImg *p_src_image, MinTyp type, AllocationOption allocation=AO_PREALLOCATED)
- MUSTINLINE int **_CloneDimensionedMinImagePrototype** (MinImg *p_dst_image, const MinImg *p_src_image, int channels, AllocationOption allocation=AO_PREALLOCATED)
- MUSTINLINE int **_CompareMinImagePrototypes** (const MinImg *p_image_a, const MinImg *p_image_b)
- MUSTINLINE int **_CompareMinImage2DSizes** (const MinImg *p_image_a, const MinImg *p_image_b)
- MUSTINLINE int **_CompareMinImage3DSizes** (const MinImg *p_image_a, const MinImg *p_image_b)
- MUSTINLINE int **_CompareMinImageTypes** (const MinImg *p_image_a, const MinImg *p_image_b)
- MUSTINLINE int **_CompareMinImagePixels** (const MinImg *p_image_a, const MinImg *p_image_b)
- MUSTINLINE int **_CompareMinImages** (const MinImg *p_image_a, const MinImg *p_image_b)
- MUSTINLINE int **_WrapAlignedBufferWithMinImage** (MinImg *p_image, void *p_buffer, int width, int height, int channels, MinTyp element_type, int stride, RulesOption rules=RO_STRICT)

- MUSTINLINE int **_WrapSolidBufferWithMinImage** (MinImg *p_image, void *p_buffer, int width, int height, int channels, MinTyp element_type, RulesOption rules=RO_STRICT)
- MUSTINLINE int **_WrapScalarWithMinImage** (MinImg *p_image, void *p_scalar, MinTyp element_type, RulesOption rules=RO_STRICT)
- MUSTINLINE int **_WrapPixelWithMinImage** (MinImg *p_image, void *p_pixel, int channels, MinTyp element_type, RulesOption rules=RO_STRICT)
- MUSTINLINE int **_WrapScalarVectorWithMinImage** (MinImg *p_image, void *p_vector, int size, DirectionOption direction, MinTyp element_type, RulesOption rules=RO_STRICT)
- MUSTINLINE int **_WrapPixelVectorWithMinImage** (MinImg *p_image, void *p_vector, int size, DirectionOption direction, int channels, MinTyp element_type, RulesOption rules=RO_STRICT)
- MUSTINLINE int **_GetMinImageRegion** (MinImg *p_dst_image, const MinImg *p_src_image, int x0, int y0, int width, int height, RulesOption rules=RO_STRICT)
- MUSTINLINE int **_FlipMinImageVertically** (MinImg *p_dst_image, const MinImg *p_src_image, RulesOption rules=RO_STRICT)
- MUSTINLINE int **_UnfoldMinImageChannels** (MinImg *p_dst_image, const MinImg *p_src_image, RulesOption rules=RO_STRICT)
- MUSTINLINE int **_SliceMinImageVertically** (MinImg *p_dst_image, const MinImg *p_src_image, int begin, int period, int end=-1, RulesOption rules=RO_STRICT)
- MUSTINLINE int **_UnrollSolidMinImage** (MinImg *p_dst_image, const MinImg *p_src_image, RulesOption rules=RO_STRICT)

6.5.1 Detailed Description

This header contains inline versions of some MinimgAPI functions. See description of the functions in file minimgapi.h

Definition in file minimgapi-inl.h.

6.6 minimgapi-inl.h

```
00001 /*
00002 Copyright (c) 2011-2013, Smart Engines Limited. All rights reserved.
00003
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without modification,
00007 are permitted provided that the following conditions are met:
00008
00009    1. Redistributions of source code must retain the above copyright notice,
00010       this list of conditions and the following disclaimer.
00011
00012    2. Redistributions in binary form must reproduce the above copyright notice,
00013       this list of conditions and the following disclaimer in the documentation
00014       and/or other materials provided with the distribution.
00015
00016 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS "AS IS" AND ANY EXPRESS OR
00017 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00018 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00019 SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
00020 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00021 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00022 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00023 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00024 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00025 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00026
00027 The views and conclusions contained in the software and documentation are those
00028 of the authors and should not be interpreted as representing official policies,
00029 either expressed or implied, of copyright holders.
00030 */
00031
00040 #pragma once
00041 #ifndef MINIMGAPI_MINIMGAPI_INL_H_INCLUDED
00042 #define MINIMGAPI_MINIMGAPI_INL_H_INCLUDED
00043
00044 #include <cstring>
```

```
00045 #include <cstdlib>
00046 #include <cstddef>
00047 #include <algorithm>
00048
00049 #include <minbase/minresult.h>
00050 #include <minbase/crossplat.h>
00051 #include <minimgapi/minimgapi.h>
00052
00053 MUSTINLINE int _GetFmtByTyp(
00054     MinTyp typ) {
00055   switch (typ) {
00056   case TYP_UINT1:
00057   case TYP_UINT8:
00058   case TYP_UINT16:
00059   case TYP_UINT32:
00060   case TYP_UINT64:
00061     return FMT_UINT;
00062   case TYP_INT8:
00063   case TYP_INT16:
00064   case TYP_INT32:
00065   case TYP_INT64:
00066     return FMT_INT;
00067   case TYP_REAL16:
00068   case TYP_REAL32:
00069   case TYP_REAL64:
00070     return FMT_REAL;
00071   default:
00072     return BAD_ARGS;
00073   }
00074 }
00075
00076 MUSTINLINE int _GetDepthByTyp(
00077     MinTyp typ) {
00078   switch (typ) {
00079   case TYP_UINT1:
00080     return 0;
00081   case TYP_UINT8:
00082   case TYP_INT8:
00083     return 1;
00084   case TYP_UINT16:
00085   case TYP_INT16:
00086   case TYP_REAL16:
00087     return 2;
00088   case TYP_UINT32:
00089   case TYP_INT32:
00090   case TYP_REAL32:
00091     return 4;
00092   case TYP_UINT64:
00093   case TYP_INT64:
00094   case TYP_REAL64:
00095     return 8;
00096   default:
00097     return BAD_ARGS;
00098   }
00099 }
00100
00101 MUSTINLINE int _GetTypByFmtAndDepth(
00102     MinFmt fmt,
00103     int    depth) {
00104   switch (fmt) {
00105   case FMT_UINT:
00106     switch (depth) {
00107     case 0:
00108       return TYP_UINT1;
00109     case 1:
00110       return TYP_UINT8;
00111     case 2:
00112       return TYP_UINT16;
00113     case 4:
00114       return TYP_UINT32;
00115     case 8:
00116       return TYP_UINT64;
00117     default:
00118       return BAD_ARGS;
00119     }
00120   case FMT_INT:
00121     switch (depth) {
00122     case 1:
00123       return TYP_INT8;
00124     case 2:
00125       return TYP_INT16;
00126     case 4:
00127       return TYP_INT32;
00128     case 8:
00129       return TYP_INT64;
00130     default:
00131       return BAD_ARGS;
```

```
00132     }
00133   case FMT_REAL:
00134     switch (depth) {
00135     case 2:
00136       return TYP_REAL16;
00137     case 4:
00138       return TYP_REAL32;
00139     case 8:
00140       return TYP_REAL64;
00141     default:
00142       return BAD_ARGS;
00143     }
00144   default:
00145     return BAD_ARGS;
00146   }
00147 }
00148
00149 MUSTINLINE int _GetMinImageType(
00150     const MinImg *p_image) {
00151   if (!p_image)
00152     return BAD_ARGS;
00153   return _GetTypByFmtAndDepth(p_image->format, p_image->channelDepth);
00154 }
00155
00156 MUSTINLINE int _SetMinImageType(
00157     MinImg *p_image,
00158     MinTyp  element_type) {
00159   if (!p_image || p_image->pScan0)
00160     return BAD_ARGS;
00161
00162   int fmt = _GetFmtByTyp(element_type);
00163   PROPAGATE_ERROR(fmt);
00164   p_image->format = static_cast<MinFmt>(fmt);
00165   int depth = _GetDepthByTyp(element_type);
00166   PROPAGATE_ERROR(depth);
00167   p_image->channelDepth = depth;
00168
00169   return NO_ERRORS;
00170 }
00171
00172 MUSTINLINE int _AssureMinImagePrototypeIsValid(
00173     const MinImg *p_image) {
00174   PROPAGATE_ERROR(_GetMinImageType(p_image));
00175   if (p_image->width < 0 || p_image->height < 0 || p_image->channels < 0)
00176     return BAD_ARGS;
00177   return NO_ERRORS;
00178 }
00179
00180 MUSTINLINE int _GetMinImageBitsPerPixel(
00181     const MinImg *p_image) {
00182   PROPAGATE_ERROR(_AssureMinImagePrototypeIsValid(p_image));
00183
00184   int bit_depth = p_image->channelDepth ? p_image->channelDepth << 3 : 1;
00185   return p_image->channels * bit_depth;
00186 }
00187
00188 MUSTINLINE int _GetMinImageBytesPerLine(
00189     const MinImg *p_image) {
00190   PROPAGATE_ERROR(_AssureMinImagePrototypeIsValid(p_image));
00191
00192   int elements_per_line = p_image->width * p_image->channels;
00193   return p_image->channelDepth > 0 ? elements_per_line * p_image->channelDepth :
00194                                      (elements_per_line + 7) >> 3;
00195 }
00196
00197 MUSTINLINE int _AssureMinImageIsValid(
00198     const MinImg *p_image) {
00199   PROPAGATE_ERROR(_AssureMinImagePrototypeIsValid(p_image));
00200   if (!p_image->width || !p_image->height || !p_image->channels)
00201     return NO_ERRORS;
00202   if (!p_image->pScan0)
00203     return BAD_ARGS;
00204   if (p_image->height > 1 &&
00205       std::abs(p_image->stride) < _GetMinImageBytesPerLine(p_image))
00206     return BAD_ARGS;
00207   return NO_ERRORS;
00208 }
00209
00210 MUSTINLINE int _AssureMinImageIsEmpty(
00211     const MinImg *p_image) {
00212   PROPAGATE_ERROR(_AssureMinImagePrototypeIsValid(p_image));
00213   return p_image->width && p_image->height && p_image->channels;
00214 }
00215
00216 MUSTINLINE int _AssureMinImageIsSolid(
00217     const MinImg *p_image) {
00218   PROPAGATE_ERROR(_AssureMinImageIsValid(p_image));
```

```
00219
00220   int bits_per_pixel = _GetMinImageBitsPerPixel(p_image);
00221   return p_image->height > 1 &&
00222          p_image->stride << 3 != p_image->width * bits_per_pixel;
00223 }
00224
00225 MUSTINLINE int _AssureMinImageFits(
00226     const MinImg *p_image,
00227     MinTyp        element_type,
00228     int           channels = -1,
00229     int           width = -1,
00230     int           height = -1) {
00231   PROPAGATE_ERROR(_AssureMinImageIsValid(p_image));
00232
00233   int retVal = 0;
00234   PROPAGATE_ERROR(retVal = _GetMinImageType(p_image));
00235   if (element_type >= 0 && element_type != retVal)
00236     return 1;
00237   if (channels >= 0 && channels != p_image->channels)
00238     return 1;
00239   if (width >= 0 && width != p_image->width)
00240     return 1;
00241   if (height >= 0 && height != p_image->height)
00242     return 1;
00243
00244   return NO_ERRORS;
00245 }
00246
00247 MUSTINLINE int _AssureMinImageIsScalar(
00248     const MinImg *p_image) {
00249   return _AssureMinImageFits(p_image, static_cast<MinTyp>(-1), 1, 1, 1);
00250 }
00251
00252 MUSTINLINE int _AssureMinImageIsPixel(
00253     const MinImg *p_image) {
00254   return _AssureMinImageFits(p_image, static_cast<MinTyp>(-1), -1, 1, 1);
00255 }
00256
00257 MUSTINLINE uint8_t *_GetMinImageLineUnsafe(
00258     const MinImg *p_image,
00259     int           y,
00260     BorderOption  border   = BO_VOID,
00261     void         *p_canvas = NULL) {
00262   const int ht = p_image->height;
00263   if (y < 0 || y >= ht)
00264     switch (border) {
00265       case BO_REPEAT: {
00266         y = std::min(std::max(0, y), ht - 1);
00267         break;
00268       }
00269       case BO_CYCLIC: {
00270         y = (y % ht + ht) % ht;
00271         break;
00272       }
00273       case BO_SYMMETRIC: {
00274         int ht2 = ht * 2;
00275         y = (y % ht2 + ht2) % ht2;
00276         y = std::min(y, ht2 - 1 - y);
00277         break;
00278       }
00279       case BO_CONSTANT: {
00280         return reinterpret_cast<uint8_t *>(p_canvas);
00281       }
00282       case BO_VOID: {
00283         return NULL;
00284       }
00285       default: {
00286       }
00287     }
00288
00289   return p_image->pScan0 + static_cast<ptrdiff_t>(p_image->stride) * y;
00290 }
00291
00292 MUSTINLINE uint8_t *_GetMinImageLine(
00293     const MinImg *p_image,
00294     int           y,
00295     BorderOption  border   = BO_VOID,
00296     void         *p_canvas = NULL) {
00297   if (_AssureMinImageIsValid(p_image) != NO_ERRORS)
00298     return NULL;
00299
00300   if (_AssureMinImageIsEmpty(p_image) == NO_ERRORS)
00301     switch (border) {
00302     case BO_CONSTANT:
00303       return reinterpret_cast<uint8_t *>(p_canvas);
00304     case BO_IGNORE:
00305       return p_image->pScan0 ? p_image->pScan0 + static_cast<ptrdiff_t>(p_image->stride) * y : NULL;
```

```
00306     default:
00307       return NULL;
00308     }
00309   return _GetMinImageLineUnsafe(p_image, y, border, p_canvas);
00310 }
00311
00312 MUSTINLINE int _CloneMinImagePrototype(
00313     MinImg          *p_dst_image,
00314     const MinImg    *p_src_image,
00315     AllocationOption allocation = AO_PREALLOCATED) {
00316   if (!p_dst_image || !p_src_image || p_dst_image->pScan0)
00317     return BAD_ARGS;
00318
00319   if (p_dst_image != p_src_image) {
00320     *p_dst_image = *p_src_image;
00321     p_dst_image->stride = 0;
00322     p_dst_image->pScan0 = 0;
00323   }
00324   if (allocation == AO_PREALLOCATED)
00325     PROPAGATE_ERROR(AllocMinImage(p_dst_image));
00326
00327   return NO_ERRORS;
00328 }
00329
00330 MUSTINLINE int _CloneResizedMinImagePrototype(
00331     MinImg          *p_dst_image,
00332     const MinImg    *p_src_image,
00333     int              width,
00334     int              height,
00335     AllocationOption allocation = AO_PREALLOCATED) {
00336   if (width < 0 || height < 0)
00337     return BAD_ARGS;
00338
00339   PROPAGATE_ERROR(_CloneMinImagePrototype(p_dst_image, p_src_image, AO_EMPTY));
00340   p_dst_image->width = width;
00341   p_dst_image->height = height;
00342   if (allocation == AO_PREALLOCATED)
00343     PROPAGATE_ERROR(AllocMinImage(p_dst_image));
00344
00345   return NO_ERRORS;
00346 }
00347
00348 MUSTINLINE int _CloneTransposedMinImagePrototype(
00349     MinImg          *p_dst_image,
00350     const MinImg    *p_src_image,
00351     AllocationOption allocation = AO_PREALLOCATED) {
00352   PROPAGATE_ERROR(_CloneResizedMinImagePrototype(p_dst_image, p_src_image,
00353                           p_src_image->height, p_src_image->width, allocation));
00354
00355   return NO_ERRORS;
00356 }
00357
00358 MUSTINLINE int _CloneRetypifiedMinImagePrototype(
00359     MinImg          *p_dst_image,
00360     const MinImg    *p_src_image,
00361     MinTyp           type,
00362     AllocationOption allocation = AO_PREALLOCATED) {
00363   PROPAGATE_ERROR(_CloneMinImagePrototype(p_dst_image, p_src_image, AO_EMPTY));
00364   PROPAGATE_ERROR(_SetMinImageType(p_dst_image, type));
00365   if (allocation == AO_PREALLOCATED)
00366     PROPAGATE_ERROR(AllocMinImage(p_dst_image));
00367
00368   return NO_ERRORS;
00369 }
00370
00371 MUSTINLINE int _CloneDimensionedMinImagePrototype(
00372     MinImg          *p_dst_image,
00373     const MinImg    *p_src_image,
00374     int              channels,
00375     AllocationOption allocation = AO_PREALLOCATED) {
00376   if (channels < 0)
00377     return BAD_ARGS;
00378
00379   PROPAGATE_ERROR(_CloneMinImagePrototype(p_dst_image, p_src_image, AO_EMPTY));
00380   p_dst_image->channels = channels;
00381   if (allocation == AO_PREALLOCATED)
00382     PROPAGATE_ERROR(AllocMinImage(p_dst_image));
00383
00384   return NO_ERRORS;
00385 }
00386
00387 MUSTINLINE int _CompareMinImagePrototypes(
00388     const MinImg *p_image_a,
00389     const MinImg *p_image_b) {
00390   MinImg prototype_a = {0};
00391   PROPAGATE_ERROR(_CloneMinImagePrototype(&prototype_a, p_image_a, AO_EMPTY));
00392   MinImg prototype_b = {0};
```

```
00393   PROPAGATE_ERROR(_CloneMinImagePrototype(&prototype_b, p_image_b, AO_EMPTY));
00394   return ::memcmp(&prototype_a, &prototype_b, sizeof(prototype_a)) != 0;
00395 }
00396
00397 MUSTINLINE int _CompareMinImage2DSizes(
00398     const MinImg *p_image_a,
00399     const MinImg *p_image_b) {
00400   if (!p_image_a || !p_image_b)
00401     return BAD_ARGS;
00402
00403   return p_image_a->width != p_image_b->width ||
00404          p_image_a->height != p_image_b->height;
00405 }
00406
00407 MUSTINLINE int _CompareMinImage3DSizes(
00408     const MinImg *p_image_a,
00409     const MinImg *p_image_b) {
00410   if (!p_image_a || !p_image_b)
00411     return BAD_ARGS;
00412
00413   return p_image_a->width != p_image_b->width ||
00414          p_image_a->height != p_image_b->height ||
00415          p_image_a->channels != p_image_b->channels;
00416 }
00417
00418 MUSTINLINE int _CompareMinImageTypes(
00419     const MinImg *p_image_a,
00420     const MinImg *p_image_b) {
00421   if (!p_image_a || !p_image_b)
00422     return BAD_ARGS;
00423
00424   return p_image_a->channelDepth != p_image_b->channelDepth ||
00425          p_image_a->format       != p_image_b->format;
00426 }
00427
00428 MUSTINLINE int _CompareMinImagePixels(
00429     const MinImg *p_image_a,
00430     const MinImg *p_image_b) {
00431   if (!p_image_a || !p_image_b)
00432     return BAD_ARGS;
00433
00434   return _CompareMinImageTypes(p_image_a, p_image_b) ||
00435          p_image_a->channels != p_image_b->channels;
00436 }
00437
00438 MUSTINLINE int _CompareMinImages(
00439     const MinImg *p_image_a,
00440     const MinImg *p_image_b) {
00441   PROPAGATE_ERROR(_AssureMinImageIsValid(p_image_a));
00442   PROPAGATE_ERROR(_AssureMinImageIsValid(p_image_b));
00443   int proto_res = _CompareMinImagePrototypes(p_image_a, p_image_b);
00444   if (proto_res)
00445     return proto_res;
00446   int height = p_image_a->height;
00447   int byte_line_width = _GetMinImageBytesPerLine(p_image_a);
00448   const uint8_t* p_line_a = _GetMinImageLine(p_image_a, 0);
00449   const uint8_t* p_line_b = _GetMinImageLine(p_image_b, 0);
00450   for (int y = 0; y < height; ++y) {
00451     if (::memcmp(p_line_a, p_line_b, byte_line_width))
00452       return 1;
00453     p_line_a += p_image_a->stride;
00454     p_line_b += p_image_b->stride;
00455   }
00456   return NO_ERRORS;
00457 }
00458
00459 MUSTINLINE int _WrapAlignedBufferWithMinImage(
00460     MinImg      *p_image,
00461     void        *p_buffer,
00462     int          width,
00463     int          height,
00464     int          channels,
00465     MinTyp       element_type,
00466     int          stride,
00467     RulesOption  rules = RO_STRICT) {
00468   if (!p_image || !p_buffer)
00469     return BAD_ARGS;
00470   if (width < 0 || height < 0 || channels < 0)
00471     return BAD_ARGS;
00472   if (!(rules & RO_REUSE_CONTAINER) && p_image->pScan0)
00473     return BAD_ARGS;
00474
00475   // TODO: Add check of std::abs(stride);
00476   // TODO: Add check of stride being >= width * channels (by abs value);
00477
00478   ::memset(p_image, 0, sizeof(*p_image));
00479   PROPAGATE_ERROR(_SetMinImageType(p_image, element_type));
```

```
00480   p_image->height = height;
00481   p_image->width = width;
00482   p_image->channels = channels;
00483   p_image->stride = stride ? stride : _GetMinImageBytesPerLine(p_image);
00484   p_image->pScan0 = reinterpret_cast<uint8_t *>(p_buffer);
00485
00486   return NO_ERRORS;
00487 }
00488
00489 MUSTINLINE int _WrapSolidBufferWithMinImage(
00490     MinImg      *p_image,
00491     void        *p_buffer,
00492     int          width,
00493     int          height,
00494     int          channels,
00495     MinTyp       element_type,
00496     RulesOption  rules = RO_STRICT) {
00497   return _WrapAlignedBufferWithMinImage(p_image, p_buffer, width, height,
00498                                         channels, element_type, 0, rules);
00499 }
00500
00501 MUSTINLINE int _WrapScalarWithMinImage(
00502     MinImg      *p_image,
00503     void        *p_scalar,
00504     MinTyp       element_type,
00505     RulesOption  rules = RO_STRICT) {
00506   return _WrapSolidBufferWithMinImage(p_image, p_scalar, 1, 1, 1,
00507                                       element_type, rules);
00508 }
00509
00510 MUSTINLINE int _WrapPixelWithMinImage(
00511     MinImg      *p_image,
00512     void        *p_pixel,
00513     int          channels,
00514     MinTyp       element_type,
00515     RulesOption  rules = RO_STRICT) {
00516   return _WrapSolidBufferWithMinImage(p_image, p_pixel, 1, 1,
00517                                       channels, element_type, rules);
00518 }
00519
00520 MUSTINLINE int _WrapScalarVectorWithMinImage(
00521     MinImg          *p_image,
00522     void            *p_vector,
00523     int              size,
00524     DirectionOption  direction,
00525     MinTyp           element_type,
00526     RulesOption  rules = RO_STRICT) {
00527   if (direction == DO_BOTH)
00528     return BAD_ARGS;
00529   return _WrapSolidBufferWithMinImage(p_image, p_vector,
00530                                       direction == DO_VERTICAL ? 1 : size,
00531                                       direction == DO_VERTICAL ? size : 1,
00532                                       1, element_type, rules);
00533 }
00534
00535 MUSTINLINE int _WrapPixelVectorWithMinImage(
00536     MinImg          *p_image,
00537     void            *p_vector,
00538     int              size,
00539     DirectionOption  direction,
00540     int              channels,
00541     MinTyp           element_type,
00542     RulesOption  rules = RO_STRICT) {
00543   if (direction == DO_BOTH)
00544     return BAD_ARGS;
00545   return _WrapSolidBufferWithMinImage(p_image, p_vector,
00546                                       direction == DO_VERTICAL ? 1 : size,
00547                                       direction == DO_VERTICAL ? size : 1,
00548                                       channels, element_type, rules);
00549 }
00550
00551 MUSTINLINE int _GetMinImageRegion(
00552     MinImg       *p_dst_image,
00553     const MinImg *p_src_image,
00554     int           x0,
00555     int           y0,
00556     int           width,
00557     int           height,
00558     RulesOption   rules = RO_STRICT) {
00559   PROPAGATE_ERROR(_AssureMinImageIsValid(p_src_image));
00560   if (!p_dst_image || width < 0 || height < 0)
00561     return BAD_ARGS;
00562   if (!(rules & RO_REUSE_CONTAINER) && p_dst_image->pScan0)
00563     return BAD_ARGS;
00564   if (!(rules & RO_IGNORE_BORDERS)) {
00565     if (x0 < 0 || static_cast<uint32_t>(x0) + static_cast<uint32_t>(width) >
00566         static_cast<uint32_t>(p_src_image->width))
```

```
00567       return BAD_ARGS;
00568     if (y0 < 0 || static_cast<uint32_t>(y0) + static_cast<uint32_t>(height) >
00569         static_cast<uint32_t>(p_src_image->height))
00570       return BAD_ARGS;
00571   }
00572
00573   *p_dst_image = *p_src_image;
00574   p_dst_image->width = width;
00575   p_dst_image->height = height;
00576
00577   int bit_shift = x0 * _GetMinImageBitsPerPixel(p_dst_image);
00578   if (bit_shift & 0x07U)
00579     return BAD_ARGS;
00580   p_dst_image->pScan0 = _GetMinImageLine(p_src_image, y0, BO_IGNORE) +
00581                         (bit_shift >> 3);
00582   if (!p_dst_image->pScan0)
00583     return INTERNAL_ERROR;
00584   p_dst_image->stride = p_src_image->stride;
00585
00586   return NO_ERRORS;
00587 }
00588
00589 MUSTINLINE int _FlipMinImageVertically(
00590     MinImg       *p_dst_image,
00591     const MinImg *p_src_image,
00592     RulesOption   rules = RO_STRICT) {
00593   if (!p_dst_image)
00594     return BAD_ARGS;
00595   if (!(rules & RO_REUSE_CONTAINER) && p_dst_image->pScan0)
00596     return BAD_ARGS;
00597   PROPAGATE_ERROR(_AssureMinImageIsValid(p_src_image));
00598
00599   *p_dst_image = *p_src_image;
00600   uint8_t *p = _GetMinImageLine(p_dst_image, p_dst_image->height - 1);
00601   if (!p)
00602     return INTERNAL_ERROR;
00603   p_dst_image->pScan0 = p;
00604   p_dst_image->stride *= -1;
00605
00606   return NO_ERRORS;
00607 }
00608
00609 MUSTINLINE int _UnfoldMinImageChannels(
00610     MinImg       *p_dst_image,
00611     const MinImg *p_src_image,
00612     RulesOption   rules = RO_STRICT) {
00613   if (!p_dst_image)
00614     return BAD_ARGS;
00615   if (!(rules & RO_REUSE_CONTAINER) && p_dst_image->pScan0)
00616     return BAD_ARGS;
00617   PROPAGATE_ERROR(_AssureMinImageIsValid(p_src_image));
00618
00619   *p_dst_image = *p_src_image;
00620   p_dst_image->width *= p_dst_image->channels;
00621   p_dst_image->channels = 1;
00622
00623   return NO_ERRORS;
00624 }
00625
00626 MUSTINLINE int _SliceMinImageVertically(
00627     MinImg       *p_dst_image,
00628     const MinImg *p_src_image,
00629     int           begin,
00630     int           period,
00631     int           end = -1,
00632     RulesOption   rules = RO_STRICT) {
00633   if (!p_dst_image || p_dst_image->pScan0)
00634     return BAD_ARGS;
00635   if (!(rules & RO_REUSE_CONTAINER) && p_dst_image->pScan0)
00636     return BAD_ARGS;
00637   PROPAGATE_ERROR(_AssureMinImageIsValid(p_src_image));
00638   if (end < 0)
00639     end = p_src_image->height;
00640   if (begin < 0 || end < begin || p_src_image->height < end || period <= 0)
00641     return BAD_ARGS;
00642
00643
00644   *p_dst_image = *p_src_image;
00645   p_dst_image->pScan0 = _GetMinImageLine(p_dst_image, begin);
00646   if (!p_dst_image->pScan0)
00647     return INTERNAL_ERROR;
00648   p_dst_image->stride *= period;
00649   p_dst_image->height = (end - begin + period - 1) / period;
00650
00651   return NO_ERRORS;
00652 }
00653
```

```
00654 MUSTINLINE int _UnrollSolidMinImage(
00655     MinImg       *p_dst_image,
00656     const MinImg *p_src_image,
00657     RulesOption   rules = RO_STRICT) {
00658   if (!p_dst_image)
00659     return BAD_ARGS;
00660   if (!(rules & RO_REUSE_CONTAINER) && p_dst_image->pScan0)
00661     return BAD_ARGS;
00662   PROPAGATE_ERROR(_AssureMinImageIsValid(p_src_image));
00663   if (_AssureMinImageIsSolid(p_src_image) != NO_ERRORS)
00664     return BAD_ARGS;
00665
00666   *p_dst_image = *p_src_image;
00667   p_dst_image->width *= p_dst_image->height;
00668   p_dst_image->height = 1;
00669
00670   return NO_ERRORS;
00671 }
00672
00673 #endif // #ifndef MINIMGAPI_MINIMGAPI_INL_H_INCLUDED
```

6.7 minimgapi.h File Reference

MinImgAPI library application programming interface.

Macros

- #define **MINIMGAPI_MINIMGAPI_H_INCLUDED**
- #define MINIMGAPI_API

  Specifies storage-class information (only for MSC).
- #define GET_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) & (0x80U >> ((x) & 7)))

  Returns value of x-th bit of image line pointed by p. If bit is on, returns not just 1, but returns it in the same position within byte, in which it was within the image byte.
- #define SET_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) |= (0x80U >> ((x) & 7)))

  Sets x-th bit of image line pointed by p to be 1.
- #define CLEAR_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) &= (0xFF7FU >> ((x) & 7)))

  Sets x-th bit of image line pointed by p to be 0.
- #define INVERT_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) ^= (0x80U >> ((x) & 7)))

  Switches the value of x-th bit of image line pointed by p.

Enumerations

- enum AllocationOption

  Specifies allocation options.
- enum RulesOption

  Specifies the degree of rules validation.
- enum BorderOption

  Specifies border acceptable border conditions.
- enum DirectionOption

  Specifies acceptable directions.
- enum TangleCheckResult

  Specifies the way two images are placed in memory with respect to each other.

Functions

- int NewMinImagePrototype (MinImg *p_image, int width, int height, int channels, MinTyp element_type, int address_space=0, AllocationOption allocation=AO_PREALLOCATED)

  Makes new MinImg, allocated or not.
- int AllocMinImage (MinImg *p_image, int alignment=16)

  Allocates an image.
- int FreeMinImage (MinImg *p_image)

  Deallocates an image.
- int CloneMinImagePrototype (MinImg *p_dst_image, const MinImg *p_src_image, AllocationOption allocation=AO_PREALLOCATED)

  Makes a copy of the image header.
- int CloneTransposedMinImagePrototype (MinImg *p_dst_image, const MinImg *p_src_image, AllocationOption allocation=AO_PREALLOCATED)

  Makes a copy of the transposed image header.
- int CloneRetypifiedMinImagePrototype (MinImg *p_dst_image, const MinImg *p_src_image, MinTyp type, AllocationOption allocation=AO_PREALLOCATED)

  Makes a copy of the image header with another type (MinTyp value).
- int CloneDimensionedMinImagePrototype (MinImg *p_dst_image, const MinImg *p_src_image, int channels, AllocationOption allocation=AO_PREALLOCATED)

  Makes a copy of the image header with another number of channels.
- int CloneResizedMinImagePrototype (MinImg *p_dst_image, const MinImg *p_src_image, int width, int height, AllocationOption allocation=AO_PREALLOCATED)

  Makes a copy of the image header with another size.
- int WrapScalarWithMinImage (MinImg *p_image, void *p_scalar, MinTyp element_type, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user scalar.
- int WrapPixelWithMinImage (MinImg *p_image, void *p_pixel, int channels, MinTyp element_type, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user pixel.
- int WrapScalarVectorWithMinImage (MinImg *p_image, void *p_vector, int size, DirectionOption direction, MinTyp element_type, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user vector of scalars.
- int WrapPixelVectorWithMinImage (MinImg *p_image, void *p_vector, int size, DirectionOption direction, int channels, MinTyp element_type, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user vector of pixels.
- int WrapSolidBufferWithMinImage (MinImg *p_image, void *p_buffer, int width, int height, int channels, MinTyp element_type, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user solid memory buffer.
- int WrapAlignedBufferWithMinImage (MinImg *p_image, void *p_buffer, int width, int height, int channels, MinTyp element_type, int stride, RulesOption rules=RO_STRICT)

  Fills MinImg structure as pointer to the user memory buffer with fixed stride.
- int GetMinImageRegion (MinImg *p_dst_image, const MinImg *p_src_image, int x0, int y0, int width, int height, RulesOption rules=RO_STRICT)

  Gets a region of an image.
- int FlipMinImageVertically (MinImg *p_dst_image, const MinImg *p_src_image, RulesOption rules=RO_STRICT)

  Flips an image in vertical without copying.
- int UnfoldMinImageChannels (MinImg *p_dst_image, const MinImg *p_src_image, RulesOption rules=RO_STRICT)

  Makes an image header where every pixel element is considered as a separate pixel.
- int SliceMinImageVertically (MinImg *p_dst_image, const MinImg *p_src_image, int begin, int period, int end=-1, RulesOption rules=RO_STRICT)

Takes a subset of equidistant image lines without copying.

- int UnrollSolidMinImage (MinImg *p_dst_image, const MinImg *p_src_image, RulesOption rules=RO_STRICT)

  Unrolls solid image into one-line image without copying.
- int GetFmtByTyp (MinTyp typ)

  Returns format that corresponds to the given type (MinTyp value).
- int GetDepthByTyp (MinTyp typ)

  Returns channel depth that corresponds to the given type (MinTyp value).
- int GetTypByFmtAndDepth (MinFmt fmt, int depth)

  Returns type (MinTyp value) that corresponds to the given format and channel depth.
- int GetMinImageType (const MinImg *p_image)

  Returns type (MinTyp value) of an image channel element.
- int SetMinImageType (MinImg *p_image, MinTyp element_type)

  Assigns type (MinTyp value) to the image.
- int GetMinImageBitsPerPixel (const MinImg *p_image)

  Returns the amount of bits in one pixel of image.
- int GetMinImageBytesPerLine (const MinImg *p_image)

  Returns the amount of bytes in one line of an image.
- int AssureMinImagePrototypeIsValid (const MinImg *p_image)

  Checks whether image prototype is valid or not.
- int AssureMinImageIsValid (const MinImg *p_image)

  Checks whether image is valid or not.
- int AssureMinImageIsEmpty (const MinImg *p_image)

  Checks whether image is empty or not.
- int AssureMinImageIsSolid (const MinImg *p_image)

  Checks whether image memory buffer is contiguous or not.
- int AssureMinImageIsScalar (const MinImg *p_image)

  Checks whether image has exactly one element (channel) or not.
- int AssureMinImageIsPixel (const MinImg *p_image)

  Checks whether image has exactly one pixel or not.
- int AssureMinImageFits (const MinImg *p_image, MinTyp element_type, int channels=-1, int width=-1, int height=-1)

  Checks whether image fits given parameters or not.
- uint8_t * GetMinImageLine (const MinImg *p_image, int y, BorderOption border=BO_VOID, void *p_canvas=NULL)

  Returns a pointer to the specified image line.
- int CompareMinImagePrototypes (const MinImg *p_image_a, const MinImg *p_image_b)

  Compares headers of two images.
- int CompareMinImage2DSizes (const MinImg *p_image_a, const MinImg *p_image_b)

  Compares sizes of two images in pixels.
- int CompareMinImage3DSizes (const MinImg *p_image_a, const MinImg *p_image_b)

  Compares sizes of two images in pixel elements (channels).
- int CompareMinImagePixels (const MinImg *p_image_a, const MinImg *p_image_b)

  Compares pixel types (MinTyp values) of two images.
- int CompareMinImageTypes (const MinImg *p_image_a, const MinImg *p_image_b)

  Compares element (channel) types (MinTyp values) of two images.
- int CompareMinImages (const MinImg *p_image_a, const MinImg *p_image_b)

  Compares headers and contents of two images.
- int CheckMinImagesTangle (uint32_t *p_result, const MinImg *p_dst_image, const MinImg *p_src_image)

  Checks how two images are placed in memory respecting to each other.
- int ZeroFillMinImage (const MinImg *p_image)

Fills every element of an image with zero value.

- int FillMinImage (const MinImg *p_image, const void *p_canvas, int value_size=0)

  Fills every line of an image cyclically with a given value.
- int CopyMinImage (const MinImg *p_dst_image, const MinImg *p_src_image)

  Copies one image to another.
- int CopyMinImageFragment (const MinImg *p_dst_image, const MinImg *p_src_image, int dst_x0, int dst_y0, int src_x0, int src_y0, int width, int height)

  Copies fragment of one image to fragment of another.
- int FlipMinImage (const MinImg *p_dst_image, const MinImg *p_src_image, DirectionOption direction)

  Flips an image around vertical or horizontal axis.
- int TransposeMinImage (const MinImg *p_dst_image, const MinImg *p_src_image)

  Transposes an image.
- int RotateMinImageBy90 (const MinImg *p_dst_image, const MinImg *p_src_image, int num_rotations)

  Rotates an image by 90 degrees (clockwise).
- int CopyMinImageChannels (const MinImg *p_dst_image, const MinImg *p_src_image, const int *p_dst_channels, const int *p_src_channels, int num_channels)

  Copies specified channels of an image to another one.
- int InterleaveMinImages (const MinImg *p_dst_image, const MinImg *const *p_p_src_images, int num_src_images)

  Interleaves pixels of the source images in the resulting image.
- int DeinterleaveMinImage (const MinImg *const *p_p_dst_images, const MinImg *p_src_image, int num_dst_images)

  Deinterleaves pixels of the source image in the resulting images.
- int ResampleMinImage (const MinImg *p_dst_image, const MinImg *p_src_image, double x_phase=0.5, double y_phase=0.5)

  Changes image sample rate.

6.7.1 Detailed Description

Some of the functions also have inline versions. These versions are defined in minimgapi-inl.h and have the same names as the ordinary ones preceded with underscore.

Definition in file minimgapi.h.

6.7.2 Enumeration Type Documentation

6.7.2.1 AllocationOption

`enum AllocationOption`

The enum specifies whether the new object should be allocated. This is used in various create- and clone-functions.

Enumerator

AO_EMPTY	The object should stay empty (without allocation).
AO_PREALLOCATED	The object should be allocated.

Definition at line 194 of file minimgapi.h.

6.7.2.2 BorderOption

`enum BorderOption`

The enum specifies acceptable options for border condition. If a function needs pixels outside of an image, then they are reconstructed according to one the following modes (that is, fill the "image border").

Enumerator

BO_IGNORE	Ignores the image size and allows out of memory reading.
BO_REPEAT	The value of pixel out of the image is assumed to be equal to the nearest one in the image.
BO_SYMMETRIC	Assumes that coordinate plane is periodical with an image as a half-period.
BO_CYCLIC	Assumes that coordinate plane is periodical with an image as a period.
BO_CONSTANT	Assumes that pixels out of image have fixed value.
BO_VOID	Assumes that pixels out of image do not exist.

Definition at line 217 of file minimgapi.h.

6.7.2.3 DirectionOption

`enum DirectionOption`

The enum specifies directions which can be used in image transformation, image filtration, calculation orientation and other functions.

Enumerator

DO_VERTICAL	Vertical transformation.
DO_HORIZONTAL	Horizontal transformation.
DO_BOTH	Transformation in both directions.

Definition at line 235 of file minimgapi.h.

6.7.2.4 RulesOption

`enum RulesOption`

The enum specifies the degree of rules validation. This can be used, for example, to choose a proper way of input arguments validation.

Enumerator

RO_STRICT	Validate each rule in a proper way.
RO_IGNORE_BORDERS	Skip validations of image borders.
RO_REUSE_CONTAINER	Allow overwrite of allocated MinImg.

Definition at line 204 of file minimgapi.h.

6.7.2.5 TangleCheckResult

```
enum TangleCheckResult
```

The enum specifies location of destination image with respect to source one, that can restrict some operations with these images.

Enumerator

TCR_TANGLED_IMAGES	Images are tangled in a complex way; one needs to copy source.
TCR_FORWARD_PASS_POSSIBLE	Every pixel of source image has address not lower than corresponding pixel of destination image; pixel-by-pixel processing is possible.
TCR_BACKWARD_PASS_POSSIBLE	Every pixel of destination image has address not lower than corresponding pixel of source image; pixel-by-pixel processing is possible for vertically flipped images.
TCR_INDEPENDENT_LINES	Corresponding lines of the images do not intersect in memory.
TCR_INDEPENDENT_IMAGES	Images do not intersect in memory.
TCR_SAME_IMAGE	Images coincide pixel-to-pixel; copy-like actions require no processing, for other cases one needs to copy source.

Definition at line 247 of file minimgapi.h.

6.8 minimgapi.h

```
00001 /*
00002 Copyright (c) 2011-2013, Smart Engines Limited. All rights reserved.
00003
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without modification,
00007 are permitted provided that the following conditions are met:
00008
00009    1. Redistributions of source code must retain the above copyright notice,
00010       this list of conditions and the following disclaimer.
00011
00012    2. Redistributions in binary form must reproduce the above copyright notice,
00013       this list of conditions and the following disclaimer in the documentation
00014       and/or other materials provided with the distribution.
00015
00016 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDERS "AS IS" AND ANY EXPRESS OR
00017 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00018 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00019 SHALL COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
00020 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00021 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00022 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00023 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00024 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00025 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00026
00027 The views and conclusions contained in the software and documentation are those
00028 of the authors and should not be interpreted as representing official policies,
00029 either expressed or implied, of copyright holders.
00030 */
00031
00041 #pragma once
00042 #ifndef MINIMGAPI_MINIMGAPI_H_INCLUDED
00043 #define MINIMGAPI_MINIMGAPI_H_INCLUDED
00044
00045 #include <stdlib.h>
```

```
00046 #include <minbase/crossplat.h>
00047 #include <minbase/minimg.h>
00048
00131 #ifdef __cplusplus
00132 extern "C" {
00133 #endif
00134
00140 #if defined _MSC_VER && defined MINIMGAPI_EXPORTS
00141 #  define MINIMGAPI_API __declspec(dllexport)
00142 #else
00143 #  define MINIMGAPI_API
00144 #endif
00145
00153 #define GET_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) & (0x80U >> ((x) & 7)))
00154
00160 #define SET_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) |= (0x80U >> ((x) & 7)))
00161
00167 #define CLEAR_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) &= (0xFF7FU >> ((x) & 7)))
00168
00174 #define INVERT_IMAGE_LINE_BIT(p, x) (((p)[(x) >> 3]) ^= (0x80U >> ((x) & 7)))
00175
00194 typedef enum {
00195   AO_EMPTY,
00196   AO_PREALLOCATED
00197 } AllocationOption;
00198
00204 typedef enum {
00205   RO_STRICT          = 0x00,
00206   RO_IGNORE_BORDERS  = 0x01,
00207   RO_REUSE_CONTAINER = 0x02
00208 } RulesOption;
00209
00217 typedef enum {
00218   BO_IGNORE,
00219   BO_REPEAT,
00220   BO_SYMMETRIC,
00222   BO_CYCLIC,
00224   BO_CONSTANT,
00226   BO_VOID
00227 } BorderOption;
00228
00235 typedef enum {
00236   DO_VERTICAL,
00237   DO_HORIZONTAL,
00238   DO_BOTH
00239 } DirectionOption;
00240
00247 typedef enum {
00248   TCR_TANGLED_IMAGES         = 0x00,
00250   TCR_FORWARD_PASS_POSSIBLE  = 0x01,
00253   TCR_BACKWARD_PASS_POSSIBLE = 0x02,
00257   TCR_INDEPENDENT_LINES      = 0x04,
00259   TCR_INDEPENDENT_IMAGES     = TCR_FORWARD_PASS_POSSIBLE |
00260                                TCR_BACKWARD_PASS_POSSIBLE |
00261                                TCR_INDEPENDENT_LINES,
00263   TCR_SAME_IMAGE             = TCR_BACKWARD_PASS_POSSIBLE |
00264                                TCR_FORWARD_PASS_POSSIBLE
00267 } TangleCheckResult;
00268
00285 MINIMGAPI_API int NewMinImagePrototype(
00286     MinImg          *p_image,
00287     int              width,
00288     int              height,
00289     int              channels,
00290     MinTyp           element_type,
00291     int              address_space IS_BY_DEFAULT(0),
00292     AllocationOption allocation    IS_BY_DEFAULT(AO_PREALLOCATED));
00293
00307 MINIMGAPI_API int AllocMinImage(
00308     MinImg *p_image,
00309     int     alignment IS_BY_DEFAULT(16));
00310
00320 MINIMGAPI_API int FreeMinImage(
00321     MinImg *p_image);
00322
00335 MINIMGAPI_API int CloneMinImagePrototype(
00336     MinImg          *p_dst_image,
00337     const MinImg    *p_src_image,
00338     AllocationOption allocation IS_BY_DEFAULT(AO_PREALLOCATED));
00339
00355 MINIMGAPI_API int CloneTransposedMinImagePrototype(
00356     MinImg          *p_dst_image,
00357     const MinImg    *p_src_image,
00358     AllocationOption allocation IS_BY_DEFAULT(AO_PREALLOCATED));
00359
00375 MINIMGAPI_API int CloneRetypifiedMinImagePrototype(
00376     MinImg          *p_dst_image,
```

```
00377     const MinImg    *p_src_image,
00378     MinTyp           type,
00379     AllocationOption allocation IS_BY_DEFAULT(AO_PREALLOCATED));
00380
00395 MINIMGAPI_API int CloneDimensionedMinImagePrototype(
00396     MinImg          *p_dst_image,
00397     const MinImg    *p_src_image,
00398     int              channels,
00399     AllocationOption allocation IS_BY_DEFAULT(AO_PREALLOCATED));
00400
00416 MINIMGAPI_API int CloneResizedMinImagePrototype(
00417     MinImg          *p_dst_image,
00418     const MinImg    *p_src_image,
00419     int              width,
00420     int              height,
00421     AllocationOption allocation IS_BY_DEFAULT(AO_PREALLOCATED));
00422
00438 MINIMGAPI_API int WrapScalarWithMinImage(
00439     MinImg     *p_image,
00440     void       *p_scalar,
00441     MinTyp      element_type,
00442     RulesOption rules IS_BY_DEFAULT(RO_STRICT));
00443
00460 MINIMGAPI_API int WrapPixelWithMinImage(
00461     MinImg     *p_image,
00462     void       *p_pixel,
00463     int         channels,
00464     MinTyp      element_type,
00465     RulesOption rules IS_BY_DEFAULT(RO_STRICT));
00466
00485 MINIMGAPI_API int WrapScalarVectorWithMinImage(
00486     MinImg          *p_image,
00487     void            *p_vector,
00488     int              size,
00489     DirectionOption  direction,
00490     MinTyp           element_type,
00491     RulesOption      rules IS_BY_DEFAULT(RO_STRICT));
00492
00511 MINIMGAPI_API int WrapPixelVectorWithMinImage(
00512     MinImg          *p_image,
00513     void            *p_vector,
00514     int              size,
00515     DirectionOption  direction,
00516     int              channels,
00517     MinTyp           element_type,
00518     RulesOption      rules IS_BY_DEFAULT(RO_STRICT));
00519
00538 MINIMGAPI_API int WrapSolidBufferWithMinImage(
00539     MinImg     *p_image,
00540     void       *p_buffer,
00541     int         width,
00542     int         height,
00543     int         channels,
00544     MinTyp      element_type,
00545     RulesOption rules IS_BY_DEFAULT(RO_STRICT));
00546
00567 MINIMGAPI_API int WrapAlignedBufferWithMinImage(
00568     MinImg     *p_image,
00569     void       *p_buffer,
00570     int         width,
00571     int         height,
00572     int         channels,
00573     MinTyp      element_type,
00574     int         stride,
00575     RulesOption rules IS_BY_DEFAULT(RO_STRICT));
00576
00593 MINIMGAPI_API int GetMinImageRegion(
00594     MinImg       *p_dst_image,
00595     const MinImg *p_src_image,
00596     int           x0,
00597     int           y0,
00598     int           width,
00599     int           height,
00600     RulesOption   rules IS_BY_DEFAULT(RO_STRICT));
00601
00614 MINIMGAPI_API int FlipMinImageVertically(
00615     MinImg       *p_dst_image,
00616     const MinImg *p_src_image,
00617     RulesOption   rules IS_BY_DEFAULT(RO_STRICT));
00618
00634 MINIMGAPI_API int UnfoldMinImageChannels(
00635     MinImg       *p_dst_image,
00636     const MinImg *p_src_image,
00637     RulesOption   rules IS_BY_DEFAULT(RO_STRICT));
00638
00658 MINIMGAPI_API int SliceMinImageVertically(
00659     MinImg       *p_dst_image,
```

```
00660     const MinImg *p_src_image,
00661     int           begin,
00662     int           period,
00663     int           end IS_BY_DEFAULT(-1),
00664     RulesOption   rules IS_BY_DEFAULT(RO_STRICT));
00665
00680 MINIMGAPI_API int UnrollSolidMinImage(
00681     MinImg       *p_dst_image,
00682     const MinImg *p_src_image,
00683     RulesOption   rules IS_BY_DEFAULT(RO_STRICT));
00684
00695 MINIMGAPI_API int GetFmtByTyp(
00696     MinTyp typ);
00697
00710 MINIMGAPI_API int GetDepthByTyp(
00711     MinTyp typ);
00712
00725 MINIMGAPI_API int GetTypByFmtAndDepth(
00726     MinFmt fmt,
00727     int    depth);
00728
00740 MINIMGAPI_API int GetMinImageType(
00741     const MinImg *p_image);
00742
00753 MINIMGAPI_API int SetMinImageType(
00754     MinImg *p_image,
00755     MinTyp  element_type);
00756
00767 MINIMGAPI_API int GetMinImageBitsPerPixel(
00768     const MinImg *p_image);
00769
00781 MINIMGAPI_API int GetMinImageBytesPerLine(
00782     const MinImg *p_image);
00783
00796 MINIMGAPI_API int AssureMinImagePrototypeIsValid(
00797     const MinImg *p_image);
00798
00811 MINIMGAPI_API int AssureMinImageIsValid(
00812     const MinImg *p_image);
00813
00827 MINIMGAPI_API int AssureMinImageIsEmpty(
00828     const MinImg *p_image);
00829
00844 MINIMGAPI_API int AssureMinImageIsSolid(
00845     const MinImg *p_image);
00846
00859 MINIMGAPI_API int AssureMinImageIsScalar(
00860     const MinImg *p_image);
00861
00873 MINIMGAPI_API int AssureMinImageIsPixel(
00874     const MinImg *p_image);
00875
00895 MINIMGAPI_API int AssureMinImageFits(
00896     const MinImg *p_image,
00897     MinTyp        element_type,
00898     int           channels      IS_BY_DEFAULT(-1),
00899     int           width         IS_BY_DEFAULT(-1),
00900     int           height        IS_BY_DEFAULT(-1));
00901
00915 MINIMGAPI_API uint8_t *GetMinImageLine(
00916     const MinImg *p_image,
00917     int           y,
00918     BorderOption  border   IS_BY_DEFAULT(BO_VOID),
00919     void         *p_canvas IS_BY_DEFAULT(NULL));
00920
00933 MINIMGAPI_API int CompareMinImagePrototypes(
00934     const MinImg *p_image_a,
00935     const MinImg *p_image_b);
00936
00948 MINIMGAPI_API int CompareMinImage2DSizes(
00949     const MinImg *p_image_a,
00950     const MinImg *p_image_b);
00951
00964 MINIMGAPI_API int CompareMinImage3DSizes(
00965     const MinImg *p_image_a,
00966     const MinImg *p_image_b);
00967
00980 MINIMGAPI_API int CompareMinImagePixels(
00981     const MinImg *p_image_a,
00982     const MinImg *p_image_b);
00983
00995 MINIMGAPI_API int CompareMinImageTypes(
00996     const MinImg *p_image_a,
00997     const MinImg *p_image_b);
00998
01010 MINIMGAPI_API int CompareMinImages(
01011     const MinImg *p_image_a,
```

```
01012     const MinImg *p_image_b);
01013
01027 MINIMGAPI_API int CheckMinImagesTangle(
01028     uint32_t     *p_result,
01029     const MinImg *p_dst_image,
01030     const MinImg *p_src_image);
01031
01041 MINIMGAPI_API int ZeroFillMinImage(
01042     const MinImg *p_image);
01043
01059 MINIMGAPI_API int FillMinImage(
01060     const MinImg *p_image,
01061     const void   *p_canvas,
01062     int           value_size IS_BY_DEFAULT(0));
01063
01077 MINIMGAPI_API int CopyMinImage(
01078     const MinImg *p_dst_image,
01079     const MinImg *p_src_image);
01080
01104 MINIMGAPI_API int CopyMinImageFragment(
01105     const MinImg *p_dst_image,
01106     const MinImg *p_src_image,
01107     int           dst_x0,
01108     int           dst_y0,
01109     int           src_x0,
01110     int           src_y0,
01111     int           width,
01112     int           height);
01113
01131 MINIMGAPI_API int FlipMinImage(
01132     const MinImg *p_dst_image,
01133     const MinImg *p_src_image,
01134     DirectionOption direction);
01135
01148 MINIMGAPI_API int TransposeMinImage(
01149     const MinImg *p_dst_image,
01150     const MinImg *p_src_image);
01151
01165 MINIMGAPI_API int RotateMinImageBy90(
01166     const MinImg *p_dst_image,
01167     const MinImg *p_src_image,
01168     int           num_rotations);
01169
01186 MINIMGAPI_API int CopyMinImageChannels(
01187     const MinImg *p_dst_image,
01188     const MinImg *p_src_image,
01189     const int    *p_dst_channels,
01190     const int    *p_src_channels,
01191     int           num_channels);
01192
01205 MINIMGAPI_API int InterleaveMinImages(
01206     const MinImg        *p_dst_image,
01207     const MinImg *const *p_p_src_images,
01208     int                  num_src_images);
01209
01224 MINIMGAPI_API int DeinterleaveMinImage(
01225     const MinImg *const *p_p_dst_images,
01226     const MinImg        *p_src_image,
01227     int                  num_dst_images);
01228
01245 MINIMGAPI_API int ResampleMinImage(
01246     const MinImg *p_dst_image,
01247     const MinImg *p_src_image,
01248     double        x_phase IS_BY_DEFAULT(0.5),
01249     double        y_phase IS_BY_DEFAULT(0.5));
01250
01251 #ifdef __cplusplus
01252 } // extern "C"
01253 #endif
01254
01255 #endif // #ifndef MINIMGAPI_MINIMGAPI_H_INCLUDED
```

Index