Allied Vision

Vimba

Vimba Image Transform Manual

1.6.1

Legal Notice

Trademarks

Unless stated otherwise, all trademarks appearing in this document are brands protected by law.

Warranty

The information provided by Allied Vision is supplied without any guarantees or warranty whatsoever, be it specific or implicit. Also excluded are all implicit warranties concerning the negotiability, the suitability for specific applications or the non-breaking of laws and patents. Even if we assume that the information supplied to us is accurate, errors and inaccuracy may still occur.

Copyright

Headquarters:
Allied Vision Technologies GmbH
Taschenweg 2a
D-07646 Stadtroda, Germany
Tel.: +49 (0)36428 6770
Fax: +49 (0)36428 677-28
e-mail: info@alliedvision.com

Contents

List of Tables

1 Contacting Allied Vision

Contact information on our website

https://www.alliedvision.com/en/meta-header/contact-us

Find an Allied Vision office or distributor

https://www.alliedvision.com/en/about-us/where-we-are

Email

info@alliedvision.com
support@alliedvision.com

Sales Offices

EMEA: +49 36428-677-230
North and South America: +1 978 225 2030
California: +1 408 721 1965
Asia-Pacific: +65 6634-9027
China: +86 (21) 64861133

Headquarters

Allied Vision Technologies GmbH
Taschenweg 2a
07646 Stadtroda
Germany

Tel: +49 (0)36428 677-0
Fax: +49 (0)36428 677-28

2 Document history and conventions

This chapter includes:

2.1 Document history

Version	Date	Changes
1.0	2013-03-22	Initial version
1.2	2013-06-18	Small corrections, layout changes
1.3	2014-08-12	Rework of the whole document
1.4	2015-11-09	Renamed several Vimba components and documents (”AVT” no longer in use), added table for pixel formats and chapter for OpenMP, added return value descriptions for functions
1.5	2016-02-27	Rework of examples and supported transformation table
1.6.0	2017-01-27	Added new supported PFNC transformations
1.6.1	June 2019	Added text in chapter Supported transformations

2.2 Conventions used in this manual

To give this manual an easily understood layout and to emphasize important information, the following typographical styles and symbols are used:

2.2.1 Styles

Style	Function	Example
Emphasis	Programs, or highlighting important things	**Emphasis**
Publication title	Publication titles	Title
Web reference	Links to web pages	Link
Document reference	Links to other documents	Document
Output	Outputs from software GUI	**Output**
Input	Input commands, modes	Input
Feature	Feature names	`Feature`

2.2.2 Symbols

Practical Tip

Safety-related instructions to avoid malfunctions
Instructions to avoid malfunctions

Further information available online

3 General aspects of the library

This chapter includes:

The purpose of the Vimba Image Transform library is to transform images received via Vimba APIs into common image formats. It is part of Vimba and doesn't require a separate installation.

Applications using Vimba C or C++ API may need to integrate Vimba Image Transform library as an additional library, whereas a subset of it is seamlessly integrated into the Vimba .NET API via method `Frame.Fill()` (See the Vimba.NET Manual if installed). As of Vimba 2.0, two additional functions enable setting a compatible destination format without knowing the PFNC (Pixel Format Naming Convention) naming scheme.

3.1 Variants

The Vimba Image Transform Library is available in two variants:

- The **standard variant** executes a function call in a single thread.
- The **OpenMP variant** distributes the function calls over multiple processor cores (parallel computing).

The OpenMP variant is recommended for achieving a high performance with multi-core PCs: Although additional time is required to schedule the sub tasks, its performance roughly scales by the number of free cores.

Both library variants have the same interface and the same name, but are delivered in different folders in the Vimba installation, allowing you to use the variant you want.

The OpenMP variant of the library can be found in the subdirectory OpenMP beneath the single-threaded library variant.

Windows supports OpenMP V2.0, whereas Unix-based systems support OpenMP V4.0.

3.1.1 Optimizing OpenMP performance

The motivation behind OpenMP is to speed up computations by distributing them to the available computing cores (hyper-threading or real ones).

To minimize the additional cost of thread management (starting/stopping threads), the OpenMP runtime environment tries to reuse threads as much as possible.

Since the default values are not adjusted to your particular use case, we recommend optimizing them to achieve high performance with low CPU load.

Active waiting tasks and CPU load

Ideally, threads are created when OpenMP is first used in an application, and they are always reused without being interrupted by the system. In cases where the application has no work items ready when tasks finish, the waiting tasks consume CPU power for a certain time and then become passive.

During active waiting, OpenMP uses high CPU load while no real work is done. This may lead to situations where OpenMP consumes too many resources although the use case isn't demanding (low image resolution, slow frame rate).

Example

If frames arrive at 10 frames per second, there is one work item every 100 ms. Even if image transformation only takes 10 ms, the threads are active for about 100 ms. OpenMP is underutilized in this case because 90% of what it's doing is waiting for new work.

If frames arrive at 100 frames per second (frames arrive at 10 ms intervals), OpenMP is working to capacity, and all threads can take on new work directly after they have finished the previous one.

It is recommended to use the same OpenMP runtime environment that the Vimba Image Transform library uses (Windows: Visual Studio 2010 or higher; Linux: gomp). In this case, Vimba Image Transform and the application share OpenMP threads, workload is optimally distributed between worker tasks, and waiting threads get new work fast.

Setting up environment variables

To optimize OpenMP, set up environment variables that determine the number of threads used and OpenMP's waiting time behavior.

- Linux and Windows
  - OMP_NUM_THREADS=[number of threads for OpenMP usage]
    Limits the number of threads that are utilized by OpenMP. Lower this value to reduce the CPU load that OpenMP is taking away from other application threads.
  - OMP_WAIT_POLICY=[ACTIVE|PASSIVE]
    Changes the policy OpenMP uses when threads wait for new work to arrive. ACTIVE is the default setting. Under Windows, you can change it only after installing Microsoft KB2686593 Hotfix. If a too long active waiting time causes high CPU load, change the environment variable to PASSIVE. Then threads don't consume CPU power while waiting for new work.
- Linux
  - GOMP_SPINCOUNT=[number of spins for active waiting]
    If OpenMP is set to OMP_WAIT_POLICY=ACTIVE, this variable determines how long a thread waits actively with consuming CPU power before waiting passively without consuming CPU power. If work items arrive at a high rate, it is best to let the thread's spin wait actively for the next work load. If work items arrive sporadically, a high spin count leads to unnecessary high CPU load.
    Setting GOMP_SPINCOUNT to 0 is the same as setting OMP_WAIT_POLICY to PASSIVE.

Further reading

- GCC Environment Variables
- Visual Studio OpenMP Environment Variables

3.2 Supported image data formats

In contrast to image file formats, image data formats contain no embedded meta-data, but only pure images as rectangular arrays of pixels. These pixels have certain characteristics, e.g. a certain color or bit

depth and the endianness for a bit depth of more than eight, and they may appear in a certain order. All this is described in a pixel format name.

All implemented image transformations only accept multi-byte pixel formats in little-endian byte order.

For this library, pixel formats from the IIDC and GigE Vision interface specification, from the Pixel Format Naming Convention as well as common display formats on Windows and Linux platforms have been selected.

The names of the pixel formats mainly match the list of values of the ”PixelFormat” feature in the Standard Features Naming Convention of GenICam, but the concrete name depends on the interface standard according to the following table:

Vimba Pixel format (VmbPixelFormat...)	GigE Vision<2.0[1] FireWire	PFNC (USB3 Vision, GigE Vision>=2.0)
Mono8	Mono8	Mono8
Mono10	Mono10	Mono10
Mono10p	-	Mono10p
Mono12	Mono12	Mono12
Mono12p	-	Mono12p
Mono12Packed	Mono12Packed	-
Mono14	Mono14	Mono14
Mono16	Mono16	Mono16
BayerXY²8	BayerXY²8	BayerXY²8
BayerXY²10	BayerXY²10	BayerXY²10
BayerXY²10p	-	BayerXY²10p
BayerXY²12	BayerXY²12	BayerXY²12
BayerXY²12p	-	BayerXY²12p
BayerXY²12Packed	BayerXY²12Packed	-
BayerXY²16	BayerXY²16	BayerXY²16

Table 1: Vimba pixel formats and their counterparts in interface standards (1/2)

Vimba Pixel format (VmbPixelFormat...)	GigE Vision<2.0[1] FireWire	PFNC (USB3 Vision, GigE Vision>=2.0)
Rgb8	RGB8Packed	RGB8
Bgr8	BGR8Packed	BGR8
Rgba8	RGBA8Packed	RGBa8
Bgra8	BGRA8Packed	BGRa8
Rgb10	RGB10Packed	RGB10
Bgr10	BGR10Packed	BGR10
Rgba10	-	RGBa10
Bgra10	-	BGRa10
Rgb12	RGB12Packed	RGB12
Bgr12	BGR12Packed	BGR12
Rgba12	-	RGBa12
Bgra12	-	BGRa12
Rgb16	RGB16Packed	RGB16
Bgr16	-	BGR16
Rgba16	-	RGBa16
Bgra16	-	BGRa16
Yuv411	YUV411Packed	YUV411_8_UYYVYY
Yuv422	YUV422Packed	YUV422_8_UYVY
Yuv444	YUV444Packed	YUV8_UYV
YCbCr411_8_CbYYCrYY	-	YCbCr411_8_CbYYCrYY (layout/values same as Yuv411)
YCbCr422_8_CbYCrY	-	YCbCr422_8_CbYCrY (layout/values same as Yuv422)
YCbCr8_CbYCr	-	YCbCr8_CbYCr (layout/values same as Yuv444)

Table 2: Vimba pixel formats and their counterparts in interface standards (2/2)

[1]Allied Vision GigE cameras comply with GigE Vision 1.2
[2]BayerXY is any of BayerGR, BayerRG, BayerGB or BayerBG

Since the following pixel formats span a few bytes, or in some cases several lines, their width and height or size (in pixels) must be divisible by the following values:

Pixel format	Width	Height	Size
`Mono12Packed,Mono12p`	1	1	2
`BayerXY`[1]`...`	2	2	1
`Yuv411/YCbCr411_8_CbYYCrYY`	4	1	1
`Yuv422/YCbCr422_8_CbYCrY`	2	1	1

Table 3: Minimum sizes for certain pixel formats

3.3 Supported transformations

The ”Target” values from table 4 can be used within the helper functions 6.3.1 and 6.3.2. Please see an example in section 5.1 for the application of a transformation operation.

Converting an image from a lower to a higher bit depth aligns the image data to the least significant bit.The remaining bits are set to zero.

Additionally to the lossless transformations in table 4, lossy transformations are supported: Mono10p, Mono12p, and Mono12Packed have a lossy conversion to 8-bit formats and a lossless conversion to Mono10 and Mono12 (both LSB and Little Endian). The same is true for Bayer[1] and RGB formats. Additionally, they have a lossy conversion to Mono formats.

[1]BayerXY is any of BayerGR, BayerRG, BayerGB or BayerBG

Source	Target	Mono 8Bit	Mono 16Bit	Color[2] 8Bit	Color[2] 16Bit
Mono8		✔	-	✔	-
Mono10		✔	-	✔	-
Mono10p		✔	-	✔	-
Mono12		✔	-	✔	-
Mono12p		✔	-	✔	-
Mono12Packed		✔	-	✔	-
Mono14		✔	-	✔	-
Mono16		✔	✔	✔	-
BayerXY[1]8		✔	-	✔	-
BayerXY[1]10		✔	✔	✔	✔
BayerXY[1]10p		✔	✔	✔	✔
BayerXY[1]12		✔	✔	✔	✔
BayerXY[1]12p		✔	✔	✔	✔
BayerXY[1]12Packed		✔	✔	✔	✔
BayerXY[1]16		✔	✔	✔	✔
Rgb8		✔	-	✔	-
Bgr8		✔	-	✔	-
Rgba8		✔	-	✔	-
Bgra8		✔	-	✔	-
Yuv411/YCbCr411_8_CbYYCrYY		✔	-	✔	-
Yuv422/YCbCr422_8_CbYCrY		✔	-	✔	-
Yuv444/YCbCr8_CbYCr		✔	-	✔	-

Table 4: Supported transformation formats

[1]BayerXY is any of BayerGR, BayerRG, BayerGB or BayerBG
[2]Color is any of RGB, RGBA, BGR or BGRA format

4 API usage

This chapter includes:

4.1 General

Every concrete image transformation must specify its three parameters:

- The fully-qualified input image (including complete format specification)
- The complete output format (plus a pointer to enough memory to hold the transformed image)
- The transformation itself (including transformation options if there is more than one possible way from the input format to the output format)

`VmbImageTransform()` is the main function, which covers all three transformation parameters. It uses two `VmbImage` pointers to specify the source and destination image and a list of `VmbTransformInfo` structs to specify the transformation.

To ease filling the structs that are needed for a successful call of `VmbImageTransform()`, several helper functions for initializing them are available:

- Methods for filling the necessary format information of either the input or the output image (described in chapter Filling Image Information).
- Methods for filling optional transformation parameters to invoke special and additional functionality, as can be seen in chapter Specifying Transformation Options.

4.2 Filling Image Information

The transformation library offers an easy way of setting the appropriate compatible destination format. The first two functions, also described in section 6.3.1 and 6.3.2, set a compatible output format and relieves the programmer from learning the PFNC naming scheme.
Initialize the `VmbImage` with:

1. `VmbSetImageInfoFromInputImage()` by providing a Vimba input image, the output layout, and bit depth
2. `VmbSetImageInfoFromInputParameters()` by providing the input format, the input image size, the output layout, and bit depth
3. `VmbSetImageInfoFromPixelFormat()` by providing a `VmbPixelFormat_t` value and the size
4. `VmbSetImageInfoFromString()` by providing a string that describes the pixel format and the size (expert)
5. Filling the fields explicitly one by one (not recommended).

4.3 Specifying Transformation Options

Depending on your application and the desired image characteristics, you can choose between several transformation options.

4.3.1 Adding color transformations

A common way to transform color images is using a 3x3 matrix with method `VmbSetColorCorrectionMatrix3x3()`. The matrix you have to specify is a 3x3 row order float matrix.

$$\begin{pmatrix} rr & rg & rb \\ gr & gg & gb \\ br & bg & bb \end{pmatrix}$$

4.3.2 Changing the debayering algorithm

Every transformation of a Bayer image requires defining how to interpolate or combine the color pixels. Because of better system performance, a simple 2x2 demosaicing algorithm is the default. Changing this algorithm is possible with the help of an additional transformation parameter to `VmbImageTransform()`, which can be filled with the help of function `VmbSetDebayerMode()` (using a `VmbDebayerMode` as input).

`VmbDebayerMode` can be one of the following values:

Value	Description
`VmbDebayerMode2x2`	2x2 with green averaging (this is the default if no debayering algorithm is added as transformation option). Supported by the OpenMP variant.
`VmbDebayerMode3x3`	3x3 with equal green weighting per line
`VmbDebayerModeLCAA`	Debayering with horizontal local color anti-aliasing[1]
`VmbDebayerModeLCAAV`	Debayering with horizontal and vertical local color anti-aliasing[1]
`VmbDebayerModeYUV422`	Debayering with YUV422-alike sub-sampling[1]

If one of `VmbDebayerModeLCAA`, `VmbDebayerModeLCAAV`, or `VmbDebayerModeYUV422` is used, the input buffers are used as intermediate buffers for the transformation for performance reasons. If you intend to use the input buffer for another purpose, debayer afterwards or copy the input buffer beforehand.

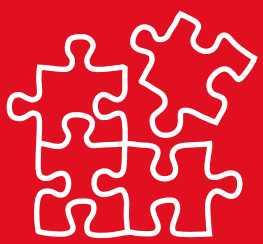
[1] For 8-bit images only

4.4 Transforming images

When you've prepared all the parameters of a transformation, use the pre-filled structs and call `VmbImageTransform()`.

Inapplicable transformation options are ignored (e.g. debayering options to a monochrome input image).

5 Transformation examples



This chapter includes:

5.1 Debayering into an Rgb8 image

Example for a transformation of a BayerGR8 image with 640x480 pixels into an Rgb8 image of the same size:

```
VmbImage                sourceImage;
VmbImage                destinationImage;
VmbTransformInfo        info;

// set size member for verification inside API
sourceImage.Size        = sizeof ( sourceImage );
destinationImage.Size   = sizeof ( destinationImage );

// attach the data buffers
sourceImage.Data        = pInBuffer;
destinationImage.Data   = pOutBuffer;

// fill image info from pixel format
VmbSetImageInfoFromPixelFormat ( VmbPixelFormatBayerGR8,
                                 640,
                                 480,
                                 &sourceImage);

// fill destination image info from input image
VmbSetImageInfoFromInputImage ( &sourceImage,
                                VmbPixelLayoutRGB,
                                8,
                                &destinationImage);

// set the debayering algorithm to simple 2 by 2
VmbSetDebayerMode ( VmbDebayerMode2x2, &info );

// perform the transformation
VmbImageTransform ( &sourceImage, &destinationImage, &info, 1 );
```

5.2 Debayering into a Mono8 image

Example for preparing a transformation of a BayerRG8 image with 640x480 pixels into a Mono8 image of the same size:

```
VmbImage                sourceImage;
VmbImage                destinationImage;
VmbTransformInfo        info;

// set size member for verification inside API
sourceImage.Size        = sizeof ( sourceImage );
destinationImage.Size   = sizeof ( destinationImage );

// attach the data buffers
sourceImage.Data        = pInBuffer;
```

```
destinationImage.Data    = pOutBuffer;

// fill image info from pixel format string
std::string name ( "PixelFormatBayerRG8" );
VmbSetImageInfoFromString ( name.c_str(),
                            static_cast<VmbUint32_t>( name.size() ),
                            640,
                            480,
                            &sourceImage );

// fill image info from pixel known input parameters
VmbSetImageInfoFromInputParameters ( VmbPixelFormatBayerRG8,
                                     640,
                                     480,
                                     VmbPixelLayoutMono,
                                     8
                                    &destinationImage );

// set the debayering algorithm to 3 by 3
VmbSetDebayerMode ( VmbDebayerMode3x3, &info );

// perform the transformation
VmbImageTransform ( &sourceImage, &destinationImage, &info, 1 );
```

5.3 Applying a color correction to an Rgb8 image

Example for applying a color correction to an Rgb8 image with 640x480 pixels:

```
VmbImage           sourceImage;
VmbImage           destinationImage;
VmbTransformInfo   info;
const VmbFloat_t   mat[] = { 1.0f, 0.0f, 0.0f,
                             0.0f, 1.0f, 0.0f,
                             0.0f, 0.0f, 1.0f };

// set size member for verification inside API
sourceImage.Size        = sizeof ( sourceImage );
destinationImage.Size   = sizeof ( destinationImage );

// attach the data buffers
sourceImage.Data        = pInBuffer;
destinationImage.Data   = pOutBuffer;

// fill image info from pixel format
VmbSetImageInfoFromPixelFormat ( VmbPixelFormatRgb8,
                                 640,
                                 480,
                                 &sourceImage );
```

```
// fill destination image info from input image
VmbSetImageInfoFromInputImage ( &sourceImage,
                                VmbPixelLayoutRGB,
                                8,
                                &destinationImage);

// set the transformation matrix
VmbSetColorCorrectionMatrix3x3 ( mat, &info );

// perform the transformation
VmbImageTransform ( &sourceImage, &destinationImage, &info, 1 );
```

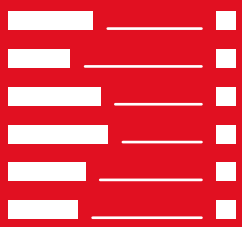
5.4 Debayering a 12-bit image into a Mono16 image with additional color correction

Example for preparing a transformation of a BayerGR12 image with 640x480 pixels into a Mono16 image of the same size:

```
VmbImage           sourceImage;
VmbImage           destinationImage;
VmbTransformInfo   info[2];
const VmbFloat_t   mat[] = { 1.0f, 0.0f, 0.0f,
                             0.0f, 1.0f, 0.0f,
                             0.0f, 0.0f, 1.0f };

// set size member for verification inside API
sourceImage.Size        = sizeof ( sourceImage );
destinationImage.Size   = sizeof ( destinationImage );

// attach the data buffers
sourceImage.Data        = pInBuffer;
destinationImage.Data   = pOutBuffer;

// fill image info from pixel format
VmbSetImageInfoFromPixelFormat ( VmbPixelFormatBayerGR12Packed,
                                 640,
                                 480,
                                 &sourceImage );

// fill destination image info from input image
VmbSetImageInfoFromInputImage ( &sourceImage,
                                VmbPixelLayoutMono,
                                16,
                                &destinationImage);

// set the debayering algorithm to 2 by 2
```

```
VmbSetDebayerMode ( VmbDebayerMode2x2, &info[0] );

// set the transformation matrix
VmbSetColorCorrectionMatrix3x3 ( mat, &info[1] );

// perform the transformation
VmbImageTransform ( &sourceImage, &destinationImage, &info, 2 );
```

6 Function reference



This chapter includes:

6.1 Information

6.1.1 VmbGetVersion()

Inquire the library version. This function can be called at anytime, even before the library is initialized.

	Type	Name	Description
out	`VmbUint32_t*`	pValue	Contains the library version (Major,Minor,Sub,Build)

- **VmbErrorBadParameter:** if the given pointer is NULL

6.1.2 VmbGetErrorInfo()

Translate Vimba error codes into a human-readable string.

	Type	Name	Description
in	`VmbError_t`	errorCode	The error code to get a readable string for
out	`VmbANSIChar_t*`	pInfo	Pointer to a zero terminated string that will contain the error information on return
in	`VmbUint32_t`	maxInfoLength	The length of the pInfo buffer

- **VmbErrorBadParameter:** if the given pointer is NULL, or if maxInfoLength is 0
- **VmbErrorMoreData:** if maxInfoLength is too small to hold the complete error information

6.1.3 VmbGetApiInfoString()

Get information about the currently loaded Vimba ImageTransform API.

	Type	Name	Description
in	`VmbAPIInfo_t`	infoType	Type of information to return
out	`VmbANSIChar_t*`	pInfo	Pointer to a zero terminated string that will contain the information on return
in	`VmbUint32_t`	maxInfoLength	The length of the info buffer

- **VmbErrorBadParameter:** if the given pointer is NULL

- **VmbErrorMoreData:** if maxInfoLength is too small to hold the complete information

Parameter infoType may be one of the following values:

Value	Description
`VmbAPIInfoAll`	Return all information about the API
`VmbAPIInfoPlatform`	Return information about the platform the API was built for (x86 or x64)
`VmbAPIInfoBuild`	Return info about the API build (debug or release)
`VmbApiInfoTechnology`	Return info about the supported technologies the API was built for (e.g. OpenMP)

6.2 Transformation

6.2.1 VmbImageTransform()

Transform images from one pixel format to another with possible transformation options. The transformation is defined by the provided images and the desired transformation.

	Type	Name	Description
in	`const VmbImage*`	pSource	Image to transform
in/out	`VmbImage*`	pDestination	Destination image
in	`const VmbTransformInfo*`	pParameter	Optional transform parameters
in	`VmbUint32_t`	parameterCount	Number of transform parameters

- **VmbErrorBadParameter:**
  - if any image pointer or their ”Data” members is NULL
  - if Width or Height don’t match between source and destination
  - if one of the parameters for the conversion does not fit
- **VmbErrorStructSize:** if the image structs size don’t match their ”Size” member
- **VmbErrorNotImplemented:** if there is no transformation between source and destination format

Fill source and destination image info structure with `VmbSetImageInfoFromPixelFormat()` or `VmbSetImageInfoFromString()` and keep those structures as template. For calls to transform, attach the image to the ”Data” member. When set, the optional parameters are constraints on the transform.

6.3 Helper functions

6.3.1 VmbSetImageInfoFromInputImage()

Set image info member values in `VmbImage` from input `VmbImage`. Use simple target layout and bit per pixel values to set the desired target format.

	Type	Name	Description
in	`const VmbImage*`	pInputmage	Source image for transformation operation
in	`VmbPixelLayout_t`	OutputPixelLayout	Pixel layout of the target image
in	`VmbUint32_t`	BitPerPixel	Bits per pixel of the target image
in/out	`VmbImage*`	pOutputImage	Pointer to Vimba image to set the info to

- **VmbErrorBadParameter:**
  - if the given pointer is NULL
  - if one of the parameters for the conversion does not fit
- **VmbErrorNotImplemented:** transformation is not supported

6.3.2 VmbSetImageInfoFromInputParameters()

Set image info member values in `VmbImage` from pixel format. Use simple target layout and bit per pixel values to set the desired target format.

	Type	Name	Description
in	`VmbPixelFormat_t`	InputPixelFormat	PixelFormat describes the pixel format used by the image data member
in	`VmbUint32_t`	Width	Width of the image in pixels
in	`VmbUint32_t`	Height	Height of the image in pixels
in	`VmbPixelLayout_t`	OutputPixelLayout	Pixel layout of the target image
in	`VmbUint32_t`	BitsPerPixel	Bits per pixel of the target image
in/out	`VmbImage*`	pOutputImage	Pointer to Vimba image to set the info to

- **VmbErrorBadParameter:** if the given pointer is NULL or one of the image struct members is invalid
- **VmbErrorNotImplemented:** transformation is not supported

6.3.3 VmbSetImageInfoFromPixelFormat()

This is an expert function. Set image info member values in `VmbImage` from pixel format. Please use table 5 from appendix as reference.

	Type	Name	Description
in	`VmbPixelFormat_t`	PixelFormat	PixelFormat describes the pixel format used by the image data member.
in	`VmbUint32_t`	Width	Width of the image in pixels
in	`VmbUint32_t`	Height	Height of the image in pixels
in/out	`VmbImage*`	pImage	Pointer to Vimba image to set the info to

- **VmbErrorBadParameter:** if the given pointer is NULL or one of the image struct members is invalid
- **VmbErrorStructSize:** if the image struct size doesn't match its "Size" member

`VmbPixelFormat_t` can be obtained from Vimba C/C++ APIs frame or from the PixelFormat feature. For displaying images, it is suggested to use `VmbSetImageInfoFromString()` or to look up a matching `VmbPixelFormat`.

6.3.4 VmbSetImageInfoFromString()

This is an expert function. Set image info member values in `VmbImage` from string.

	Type	Name	Description
in	`const VmbANSIChar_t*`	imageFormat	Image format as a (const) case insensitive string that is either a PixelFormat (Vmb is optional) or a pixel struct name
in	`VmbUint32_t`	StringLength	The length of the pixel format string
in	`VmbUint32_t`	Width	Width of the image in pixels
in	`VmbUint32_t`	Height	Height of the image in pixels
in/out	`VmbImage*`	pImage	Pointer to Vimba image to set the info to

- **VmbErrorBadParameter:**
  - if one of the given pointers or the ”Data” member in the image is NULL
  - if Width or Height don’t match between source and destination
  - if one of the parameters for the conversion does not fit
- **VmbErrorStructSize:** if the image struct size does not match its ”Size” member
- **VmbErrorResources:** if the there was a memory fault during processing

6.3.5 VmbSetDebayerMode()

Set transformation options to a predefined debayering mode.

	Type	Name	Description
in	`VmbDebayerMode_t`	DebayerMode	The mode used for debayering the source raw image, default mode is 2x2 debayering. Debayering modes only work for image widths and heights divisible by two.
in/out	`VmbTransformInfo_t*`	pTransformInfo	Parameter that contains information about special transform functionality.

- **VmbErrorBadParameter:** if the given pointer is NULL

Debayering is only applicable to image formats with both an even width and an even height.

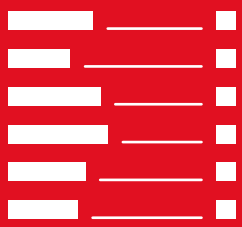
6.3.6 VmbSetColorCorrectionMatrix3x3()

Set transformation options to a 3x3 color matrix transformation.

	Type	Name	Description
in	`const VmbFloat_t*`	pMatrix	Color correction matrix
in/out	`VmbTransformInfo_t*`	pTransformInfo	Parameter that is filled with information about special transform functionality.

- **VmbErrorBadParameter:** if one of the given pointers is NULL

7 Structs



This chapter includes:

7.1 VmbImage

`VmbImage` encapsulates image data for the transformation function.

Struct entry		Purpose
`VmbUint32_t`	Size	Size of the structure
`void*`	Data	Pointer to the payload received from Vimba C/C++ API or to the display image data
`VmbImageInfo`	ImageInfo	Internal information data used for mapping the images to the correct transformation, imageInfo data can be set with `VmbSetImageInfo()` helper functions from `VmbPixelFormat_t` or format string

7.2 VmbImageInfo

`VmbImageInfo` contains image information needed for the transformation function.

Struct entry		Purpose
`VmbUint32_t`	Width	The width of the image in pixels. For macro pixel formats like YUV, it is the width in sub pixels
`VmbUint32_t`	Height	The height of the image in pixels
`VmbInt32_t`	Stride	The offset from the current line to the next line, a value not equal to Width is currently not supported
`VmbPixelInfo`	PixelInfo	Information about the pixel format

7.3 VmbPixelInfo

`VmbPixelInfo` describes the pixel format of an image.

Struct entry		Purpose
`VmbUint32_t`	BitsPerPixel	Number of bits for one image pixel, or sub pixel in macro pixel formats
`VmbUint32_t`	BitsUsed	Number of bits used per pixel, e.g. RGB12 has 48 bits per pixel and 36 bits used
`VmbAlignment_t`	Alignment	For image formats where BitsPerPixel is not equal to bitsUsed, the alignment specifies the bit layout of the pixel
`VmbEndianness_t`	Endianness	Specifies the endianness of pixels that are larger than one byte
`VmbPixelLayout_t`	PixelLayout	Describes the layout of the pixel component, e.g., RGB or BGR layout
`VmbBayerPattern_t`	BayerPattern	For raw image data, this field specifies the color filter array layout of the sensor

7.4 VmbTransformInfo

Transformation parameters given as a pointer to a `VmbTransformInfo` struct are used to invoke special and additional functionality, which is applied while interpreting the source image (debayering) or transforming the images (color correction).

Struct entry		Purpose
`VmbTransformType_t`	TransformType	Transform info type, may be VmbTransformTypeDebayerMode or VmbTransformTypeColorCorrectionMatrix
`VmbTransformParameter`	Parameter	Transform info variant, may be VmbTransformParameterDebayer or VmbTransformParameterMatrix3x3

A Supported PFNC transformations

Source	Target	Mono8	Mono10	Mono12	Mono14	Mono16	Rgb/Bgr8	Rgba/Bgra8	Rgb/Bgr10	Rgba/Bgra10	Rgb/Bgr12	Rgba/Bgra12	Rgb/Bgr14	Rgba/Bgra14	Rgb/Bgr16	Rgba/Bgra16	Yuv422
Mono8		✔	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	✔
Mono10		✔	✔	-	-	-	✔	✔	✔	✔	-	-	-	-	-	-	-
Mono10p		✔	✔	-	-	-	✔	✔	✔	✔	-	-	-	-	-	-	-
Mono12		✔	-	✔	-	-	✔	✔	-	-	✔	✔	-	-	-	-	-
Mono12p		✔	-	✔	-	-	✔	✔	-	-	✔	✔	-	-	-	-	-
Mono12Packed		✔	-	✔	-	-	✔	✔	-	-	✔	✔	-	-	-	-	-
Mono14		✔	-	-	✔	-	✔	✔	-	-	-	-	✔	✔	-	-	-
Mono16		✔	-	-	-	✔	✔	✔	-	-	-	-	-	-	✔	✔	-
BayerXY[1]8		✔	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	✔
BayerXY[1]10		✔	✔	-	-	-	✔	✔	✔	✔	-	-	-	-	-	-	-
BayerXY[1]10p		✔	✔	-	-	-	✔	✔	✔	✔	-	-	-	-	-	-	-
BayerXY[1]12		✔	-	✔	-	-	✔	✔	-	-	✔	✔	-	-	-	-	-
BayerXY[1]12p		✔	-	✔	-	-	✔	✔	-	-	✔	✔	-	-	-	-	-
BayerXY[1]12Packed		✔	-	✔	-	-	✔	✔	-	-	✔	✔	-	-	-	-	-
BayerXY[1]16		✔	-	-	-	✔	✔	✔	-	-	-	-	-	-	✔	✔	-
Rgb8		✔	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	-
Bgr8		✔	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	-
Rgba8 / Argb8		✔	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	-
Bgra8		✔	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	-
Yuv411/YCbCr411_8_CbYYCrYY		✔	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	-
Yuv422/YCbCr422_8_CbYCrY		✔	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	✔
Yuv444/YCbCr8_CbYCr		✔	-	-	-	-	✔	✔	-	-	-	-	-	-	-	-	-

Table 5: Supported PFNC transformations

[1]BayerXY is any of BayerGR, BayerRG, BayerGB or BayerBG