Vimba C API

Function Reference Manual

V1.2
2013-Jun-25

Allied Vision Technologies GmbH
Taschenweg 2a
D-07646 Stadtroda / Germany

Legal Notice

Trademarks

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

Warranty

The information provided by Allied Vision Technologies 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

All texts, pictures and graphics are protected by copyright and other laws protecting intellectual property. It is not permitted to copy or modify them for trade use or transfer, nor may they be used on websites.

Allied Vision Technologies GmbH 06/2013

Managing Director: Mr. Frank Grube
Tax ID: DE 184383113

Headquarters:

Taschenweg 2a
D-07646 Stadtroda, Germany
Tel.: +49 (0)36428 6770
Fax: +49 (0)36428 677-28
e-mail: info@alliedvisiontec.com

Contents

1 Contacting Allied Vision Technologies 4

2 Introduction 5
- 2.1 Conventions used in this manual . . . 5
  - 2.1.1 Styles . . . 5
  - 2.1.2 Symbols . . . 5

3 Callbacks 6
- 3.1 VmbInvalidationCallback . . . 6
- 3.2 VmbFrameCallback . . . 6

4 API Version 7
- 4.1 VmbVersionQuery() . . . 7

5 API Initialization 8
- 5.1 VmbStartup() . . . 8
- 5.2 VmbShutdown() . . . 8

6 Camera Enumeration & Information 9
- 6.1 VmbCamerasList() . . . 9
- 6.2 VmbCameraInfoQuery() . . . 9
- 6.3 VmbCameraOpen() . . . 10
- 6.4 VmbCameraClose() . . . 10

7 Features 11
- 7.1 VmbFeaturesList() . . . 11
- 7.2 VmbFeatureInfoQuery() . . . 11
- 7.3 VmbFeatureListAffected() . . . 12
- 7.4 VmbFeatureListSelected() . . . 12
- 7.5 VmbFeatureAccessQuery() . . . 13

8 Integer 14
- 8.1 VmbFeatureIntGet() . . . 14
- 8.2 VmbFeatureIntSet() . . . 14
- 8.3 VmbFeatureIntRangeQuery() . . . 14
- 8.4 VmbFeatureIntIncrementQuery() . . . 15

9 Float 16
- 9.1 VmbFeatureFloatGet() . . . 16
- 9.2 VmbFeatureFloatSet() . . . 16
- 9.3 VmbFeatureFloatRangeQuery() . . . 16

10 Enum 18

10.1 VmbFeatureEnumGet() . . . 18
10.2 VmbFeatureEnumSet() . . . 18
10.3 VmbFeatureEnumRangeQuery() . . . 18
10.4 VmbFeatureEnumIsAvailable() . . . 19
10.5 VmbFeatureEnumAsInt() . . . 19
10.6 VmbFeatureEnumAsString() . . . 20
10.7 VmbFeatureEnumEntryGet() . . . 20

11 String **22**
11.1 VmbFeatureStringGet() . . . 22
11.2 VmbFeatureStringSet() . . . 22
11.3 VmbFeatureStringMaxlengthQuery() . . . 22

12 Boolean **24**
12.1 VmbFeatureBoolGet() . . . 24
12.2 VmbFeatureBoolSet() . . . 24

13 Command **25**
13.1 VmbFeatureCommandRun() . . . 25
13.2 VmbFeatureCommandIsDone() . . . 25

14 Raw **26**
14.1 VmbFeatureRawGet() . . . 26
14.2 VmbFeatureRawSet() . . . 26
14.3 VmbFeatureRawLengthQuery() . . . 27

15 Feature invalidation **28**
15.1 VmbFeatureInvalidationRegister() . . . 28
15.2 VmbFeatureInvalidationUnregister() . . . 28

16 Image preparation and acquisition **30**
16.1 VmbFrameAnnounce() . . . 30
16.2 VmbFrameRevoke() . . . 30
16.3 VmbFrameRevokeAll() . . . 31
16.4 VmbCaptureStart() . . . 31
16.5 VmbCaptureEnd() . . . 31
16.6 VmbCaptureFrameQueue() . . . 32
16.7 VmbCaptureFrameWait() . . . 32
16.8 VmbCaptureQueueFlush() . . . 32

17 Interface Enumeration & Information **34**
17.1 VmbInterfacesList() . . . 34
17.2 VmbInterfaceOpen() . . . 34
17.3 VmbInterfaceClose() . . . 35

18 Ancillary data **36**
18.1 VmbAncillaryDataOpen() 36
18.2 VmbAncillaryDataClose() 36

19 Raw memory/register access **37**
19.1 VmbMemoryRead() 37
19.2 VmbMemoryWrite() 37
19.3 VmbRegistersRead() 37
19.4 VmbRegistersWrite() 38

1 Contacting Allied Vision Technologies

Note

- **Technical Information**
  http://www.alliedvisiontec.com
- **Support**
  support@alliedvisiontec.com

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

Allied Vision Technologies Canada Inc.
101-3750 North Fraser Way
Burnaby, BC, V5J 5E9, Canada
Tel: +1 604-875-8855
Fax: +1 604-875-8856
Email: info@alliedvisiontec.com

Allied Vision Technologies Inc.
38 Washington Street
Newburyport, MA 01950, USA
Toll Free number +1 877-USA-1394
Tel.: +1 978-225-2030
Fax: +1 978-225-2029
Email: info@alliedvisiontec.com

Allied Vision Technologies Asia Pte. Ltd.
82 Playfair Road
#07-02 D'Lithium
Singapore 368001
Tel. +65 6634-9027
Fax:+65 6634-9029
Email: info@alliedvisiontec.com

Allied Vision Technologies (Shanghai) Co., Ltd.
2-2109 Hongwell International Plaza
1602# ZhongShanXi Road
Shanghai 200235, China
Tel: +86 (21) 64861133
Fax: +86 (21) 54233670
Email: info@alliedvisiontec.com

2 Introduction

2.1 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:

2.1.1 Styles

Style	Function	Example
Bold	Programs, inputs or highlighting important things	**bold**
Courier	Code listings etc.	`Input`
Upper case	Constants	CONSTANT
Italics	Modes, fields	*Mode*
Parentheses and/or blue	Links	(Link)

2.1.2 Symbols

Note

This symbol highlights important information.

Caution

This symbol highlights important instructions. You have to follow these instructions to avoid malfunctions.

www

This symbol highlights URLs for further information. The URL itself is shown in blue.
Example: http://www.alliedvisiontec.com

3 Callbacks

3.1 VmbInvalidationCallback

Invalidation Callback type

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
in	`void*`	pUserContext	Pointer to the user context, see VmbFeatureInvalidationRegister

3.2 VmbFrameCallback

Frame Callback type

	Type	Name	Description
in	`const VmbHandle_t`	cameraHandle	Handle of the camera
out	`VmbFrame_t*`	pFrame	Frame completed

4 API Version

4.1 VmbVersionQuery()

Retrieve the version number of VimbaC.

	Type	Name	Description
out	`VmbVersionInfo_t*`	pVersionInfo	Pointer to the struct where version information is copied
in	`VmbUint32_t`	sizeofVersionInfo	Size of structure in bytes

- **VmbErrorSuccess:** If no error
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API
- **VmbErrorBadParameter:** "pVersionInfo" is NULL.

Note

This function can be called at anytime, even before the API is initialized. All other version numbers may be queried via feature access

5 API Initialization

5.1 VmbStartup()

Initialize the VimbaC API.

- **VmbErrorSuccess:** If no error
- **VmbErrorInternalFault:** An internal fault occurred

Note

On successful return, the API is initialized; this is a necessary call.

5.2 VmbShutdown()

Perform a shutdown on the API.

Note

This will free some resources and deallocate all physical resources if applicable.

6 Camera Enumeration & Information

6.1 VmbCamerasList()

Retrieve a list of all cameras.

	Type	Name	Description
out	`VmbCameraInfo_t*`	pCameraInfo	Array of VmbCameraInfo_t, allocated by the caller. The camera list is copied here. May be NULL.
in	`VmbUint32_t`	listLength	Number of VmbCameraInfo_t elements provided
out	`VmbUint32_t*`	pNumFound	Number of VmbCameraInfo_t elements found. May be NULL.
in	`VmbUint32_t`	sizeofCameraInfo	Size of the structure

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorStructSize:** The given struct size is not valid for this API version
- **VmbErrorMoreData:** More data was returned than space was provided

Note

Camera detection is started with the first call of VmbCamerasList() or the registration of the "DiscoveryInterfaceEvent" event. The first call of VmbCamerasList() might be delayed if no "DiscoveryInterfaceEvent" event is registered (see GigE Discovery procedure). If "pCameraInfo" is NULL on entry, only the number of interfaces is returned in "pNumFound".

6.2 VmbCameraInfoQuery()

Retrieve information on a camera given by an ID.

	Type	Name	Description
in	`const char*`	idString	Unique ID of the camera
out	`VmbCameraInfo_t*`	pInfo	Structure where information will be copied. May be NULL.
in	`VmbUint32_t`	sizeofCameraInfo	Size of the structure

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorNotFound:** The designated camera cannot be found
- **VmbErrorStructSize:** The given struct size is not valid for this API version
- **VmbErrorMoreData:** More data was returned than space was provided

Note

May be called if a camera is not yet under control of the application. "idString" might be one of the following:"169.254.12.13" or a plain serial number: "1234567890"

6.3 VmbCameraOpen()

Open the specified camera.

	Type	Name	Description
in	`const char*`	idString	Unique ID of the camera
in	`VmbAccessMode_t`	accessMode	Access mode determines the amount of control you have on the camera
out	`VmbHandle_t*`	pCameraHandle	A camera handle

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorNotFound:** The designated camera cannot be found
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode

Note

A camera may be opened in a specific access mode. This mode determines the amount of control you have on a camera. "idString" might be one of the following:"169.254.12.13" or a plain serial number: "1234567890".

6.4 VmbCameraClose()

Close the specified camera.

	Type	Name	Description
in	`const VmbHandle_t`	cameraHandle	A valid camera handle

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command

Note

Depending on the access mode this camera was opened with, events are killed, callbacks are unregistered, and camera control is released.

7 Features

7.1 VmbFeaturesList()

List all the features for this module.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
out	`VmbFeatureInfo_t*`	pFeatureInfoList	An array of VmbFeatureInfo_t to be filled by the API. May be NULL.
in	`VmbUint32_t`	listLength	Number of VmbFeatureInfo_t elements provided
out	`VmbUint32_t*`	pNumFound	Number of VmbFeatureInfo_t elements found. May be NULL.
in	`VmbUint32_t`	featureInfoSize	Size of an VmbFeatureInfo_t entry

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API

Note

This method lists all implemented features, whether they are currently available or not. The list of features does not change as long as the camera/interface is connected. "pNumFound" returns the number of VmbFeatureInfo elements.

7.2 VmbFeatureInfoQuery()

Query information about the constant properties of a feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
out	`VmbFeatureInfo_t*`	pFeatureInfo	The feature info to query.
in	`VmbUint32_t`	featureInfoSize	Size of the structure

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API

Note

Users provide a pointer to VmbFeatureInfo_t which is then set to the internal representation.

7.3 VmbFeatureListAffected()

List all the features that might be affected by changes to this feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
out	`VmbFeatureInfo_t*`	pFeatureInfoList	An array of VmbFeatureInfo_t to be filled by the API. May be NULL.
in	`VmbUint32_t`	listLength	Number of VmbFeatureInfo_t elements provided
out	`VmbUint32_t*`	pNumFound	Number of VmbFeatureInfo_t elements found. May be NULL.
in	`VmbUint32_t`	featureInfoSize	Size of an VmbFeatureInfo_t entry

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API

Note

This method lists all affected features, whether they are currently available or not. The value of affected features depends directly or indirectly on this feature (including all selected features). The list of features does not change as long as the camera/interface is connected. "pNumFound" returns the number of VmbFeatureInfo elements.

7.4 VmbFeatureListSelected()

List all the features selected by a given feature for this module.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
out	`VmbFeatureInfo_t*`	pFeatureInfoList	An array of VmbFeatureInfo_t to be filled by the API. May be NULL.
in	`VmbUint32_t`	listLength	Number of VmbFeatureInfo_t elements provided
out	`VmbUint32_t*`	pNumFound	Number of VmbFeatureInfo_t elements found. May be NULL.
in	`VmbUint32_t`	featureInfoSize	Size of an VmbFeatureInfo_t entry

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API

Note

This method lists all selected features, whether they are currently available or not. Features having selected features ("selectors") have no direct impact on the camera, but only have an influence on the register address that selected features point to. The list of features does not change as long as the camera/interface is connected. "pNumFound" returns the number of VmbFeatureInfo elements.

7.5 VmbFeatureAccessQuery()

Return the dynamic read and write capabilities of this feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features.
in	`const char *`	name	Name of the feature.
out	`VmbBool_t *`	pIsReadable	Indicates if this feature is readable. May be NULL.
out	`VmbBool_t *`	pIsWriteable	Indicates if this feature is writable. May be NULL.

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorBadParameter:** pIsReadable and pIsWriteable were both NULL

Note

The access mode of a feature may change. For example, if "PacketSize" is locked while image data is streamed, it is only readable.

8 Integer

8.1 VmbFeatureIntGet()

Get the value of an integer feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
out	`VmbInt64_t*`	pValue	Value to get

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Integer

8.2 VmbFeatureIntSet()

Set the value of an integer feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
in	`VmbInt64_t`	value	Value to set

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Integer
- **VmbErrorInvalidValue:** "value" is either out of bounds or not an increment of the minimum

8.3 VmbFeatureIntRangeQuery()

Query the range of an integer feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
out	`VmbInt64_t*`	pMin	Minimum value to be returned. May be NULL.
out	`VmbInt64_t*`	pMax	Maximum value to be returned. May be NULL.

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Integer

8.4 VmbFeatureIntIncrementQuery()

Query the increment of an integer feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
out	`VmbInt64_t*`	pValue	Value of the increment to get.

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Integer

9 Float

9.1 VmbFeatureFloatGet()

Get the value of a float feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
out	`double*`	pValue	Value to get

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Float

9.2 VmbFeatureFloatSet()

Set the value of a float feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
in	`double`	value	Value to set

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Float
- **VmbErrorInvalidValue:** "value" is not within valid bounds

9.3 VmbFeatureFloatRangeQuery()

Query the range of a float feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
out	`double*`	pMin	Minimum value to be returned. May be NULL.
out	`double*`	pMax	Maximum value to be returned. May be NULL.

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Float

Note

Only one of the values may be queried if the other parameter is set to NULL, but if both parameters are NULL, an error is returned.

10 Enum

10.1 VmbFeatureEnumGet()

Get the value of an enumeration feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
out	`const char**`	pValue	The current enumeration value

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Enumeration

10.2 VmbFeatureEnumSet()

Set the value of an enumeration feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
in	`const char*`	value	Value to set

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Enumeration
- **VmbErrorInvalidValue:** "value" is not within valid bounds

10.3 VmbFeatureEnumRangeQuery()

Query the value range of an enumeration feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
out	`const char*`	const*	pNameArray An Array of enumeration value names
in	`VmbUint32_t`	arrayLength	Number of elements in the array
out	`VmbUint32_t *`	pNumFilled	Number of filled elements

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorMoreData:** More data was returned than space was provided
- **VmbErrorWrongType:** The type of feature "name" is not Enumeration

10.4 VmbFeatureEnumIsAvailable()

Check if a certain value of an enumeration is available.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
in	`const char*`	value	Value to check
out	`VmbBool_t *`	pIsAvailable	Indicates if the given enumeration value is available

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Enumeration

10.5 VmbFeatureEnumAsInt()

Get the integer value for a given enumeration string value.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
in	`const char*`	value	The enumeration value to get the integer value for
out	`VmbInt64_t*`	pIntVal	The integer value for this enumeration entry

- **VmbErrorSuccess:** If no error

- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Enumeration

Note

Converts a name of an enum member into an int value ("Mono12Packed" to 0x10C0006)

10.6 VmbFeatureEnumAsString()

Get the enumeration string value for a given integer value.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the feature
in	`VmbInt64_t`	intValue	The numeric value
out	`const char**`	pStringValue	The string value for the numeric value

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Enumeration

Note

Converts an int value to a name of an enum member (e.g. 0x10C0006 to "Mono12Packed")

10.7 VmbFeatureEnumEntryGet()

Get infos about an entry of an enumeration feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	featureName	Name of the feature
in	`const char*`	entryName	Name of the enum entry of that feature
out	`VmbFeatureEnumEntry_t*`	pFeatureEnumEntry	Infos about that entry returned by the API
in	`VmbUint32_t`	featureEnumEntrySize	Size of the structure

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Enumeration

11 String

11.1 VmbFeatureStringGet()

Get the value of a string feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the string feature
out	`char*`	buffer	String buffer to fill
in	`VmbUint32_t`	bufferSize	Size of the input buffer
out	`VmbUint32_t*`	pSizeFilled	Size actually filled

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorMoreData:** More data was returned than space was provided
- **VmbErrorWrongType:** The type of feature "name" is not String

11.2 VmbFeatureStringSet()

Set the value of a string feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the string feature
in	`const char*`	value	Value to set

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not String
- **VmbErrorInvalidValue:** Length of "value" exceeded the maximum length

11.3 VmbFeatureStringMaxlengthQuery()

Get the maximum length of a string feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the string feature
out	`VmbUint32_t*`	pMaxLength	Maximum length of this string feature

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not String

12 Boolean

12.1 VmbFeatureBoolGet()

Get the value of a boolean feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the boolean feature
out	`VmbBool_t *`	pValue	Value to be read

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Boolean

12.2 VmbFeatureBoolSet()

Set the value of a boolean feature.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the boolean feature
in	`VmbBool_t`	value	Value to write

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Boolean
- **VmbErrorInvalidValue:** "value" is not within valid bounds

13 Command

13.1 VmbFeatureCommandRun()

Run a feature command.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the command feature

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Command

13.2 VmbFeatureCommandIsDone()

Check if a feature command is done.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the command feature
out	`VmbBool_t *`	pIsDone	State of the command.

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Command

14 Raw

14.1 VmbFeatureRawGet()

Read the memory contents of an area given by a feature name.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the raw feature
out	`char*`	pBuffer	Buffer to fill
in	`VmbUint32_t`	bufferSize	Size of the buffer to be filled
out	`VmbUint32_t*`	pSizeFilled	Number of bytes actually filled

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorMoreData:** More data was returned than space was provided
- **VmbErrorWrongType:** The type of feature "name" is not Register

Note

This feature type corresponds to a first-level "Register" node in the XML file. Data transfer is split up by the transport layer if the feature length is too large. You can get the size of the memory area addressed by the feature "name" by VmbFeatureRawLengthQuery().

14.2 VmbFeatureRawSet()

Write to a memory area given by a feature name.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the raw feature
in	`const char*`	pBuffer	Data buffer to use
in	`VmbUint32_t`	bufferSize	Size of the buffer

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Register

Note

This feature type corresponds to a first-level "Register" node in the XML file. Data transfer is split up by the transport layer if the feature length is too large. You can get the size of the memory area addressed by the feature "name" by VmbFeatureRawLengthQuery().

14.3 VmbFeatureRawLengthQuery()

Get the length of a raw feature for memory transfers.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that exposes features
in	`const char*`	name	Name of the raw feature
out	`VmbUint32_t*`	pLength	Length of the raw feature area (in bytes)

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The type of feature "name" is not Register

Note

This feature type corresponds to a first-level "Register" node in the XML file.

15 Feature invalidation

15.1 VmbFeatureInvalidationRegister()

Register a callback for feature invalidation signaling.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that emits events
in	`const char*`	name	Name of the event (NULL to register for any feature)
in	`VmbInvalidationCallback`	callback	Callback to be run, when invalidation occurs
in	`void*`	pUserContext	User context passed to function

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode

Note

Registering multiple callbacks for one feature invalidation event is possible because only the combination of handle, name, and callback is used as key. If the same combination of handle, name, and callback is registered a second time, it overwrites the previous one.

Caution

Consider if it could make sense to register the same combination of handle, name, and callback (with different user context) so that the same callback is called multiple times. Callbacks would have to be stored in a queue/stack.

15.2 VmbFeatureInvalidationUnregister()

Unregister a previously registered feature invalidation callback.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that emits events
in	`const char*`	name	Name of the event
in	`VmbInvalidationCallback`	callback	Callback to be removed

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode

Note

Since multiple callbacks may be registered for a feature invalidation event, a combination of handle, name, and callback is needed for unregistering, too.

16 Image preparation and acquisition

16.1 VmbFrameAnnounce()

Announce frames to the API that may be queued for frame capturing later.

	Type	Name	Description
in	`const VmbHandle_t`	cameraHandle	Handle for a camera
in	`const VmbFrame_t*`	pFrame	Frame buffer to announce
in	`VmbUint32_t`	sizeofFrame	Size of the frame structure

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API

Note

Allows some preparation for frames like DMA preparation depending on the transport layer. The order in which the frames are announced is not taken into consideration by the API.

16.2 VmbFrameRevoke()

Revoke a frame from the API.

	Type	Name	Description
in	`const VmbHandle_t`	cameraHandle	Handle for a camera
in	`const VmbFrame_t*`	pFrame	Frame buffer to be removed from the list of announced frames

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given frame pointer is not valid
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API

Note

The referenced frame is removed from the pool of frames for capturing images.

16.3 VmbFrameRevokeAll()

Revoke all frames assigned to a certain camera.

	Type	Name	Description
in	`const VmbHandle_t`	cameraHandle	Handle for a camera

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid

16.4 VmbCaptureStart()

Prepare the API for incoming frames.

	Type	Name	Description
in	`const VmbHandle_t`	cameraHandle	Handle for a camera

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorDeviceNotOpen:** Camera was not opened for usage
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode

16.5 VmbCaptureEnd()

Stop the API from being able to receive frames.

	Type	Name	Description
in	`const VmbHandle_t`	cameraHandle	Handle for a camera

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid

Note

Consequences of VmbCaptureEnd(): - The input queue is flushed - The frame callback will not be called any more

16.6 VmbCaptureFrameQueue()

Queue frames that may be filled during frame capturing.

	Type	Name	Description
in	`const VmbHandle_t`	cameraHandle	Handle of the camera
in	`const VmbFrame_t*`	pFrame	Pointer to an already announced frame
in	`VmbFrameCallback`	callback	Callback to be run when the frame is complete. NULL is Ok.

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given frame is not valid
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API

Note

The given frame is put into a queue that will be filled sequentially. The order in which the frames are filled is determined by the order in which they are queued. If the frame was announced with VmbFrameAnnounce() before, the application has to ensure that the frame is also revoked by calling VmbFrameRevoke or VmbFrameRevokeAll when cleaning up.

16.7 VmbCaptureFrameWait()

Wait for a queued frame to be filled (or dequeued).

	Type	Name	Description
in	`const VmbHandle_t`	cameraHandle	Handle of the camera
in	`const VmbFrame_t*`	pFrame	Pointer to an already announced & queued frame
in	`VmbUint32_t`	timeout	Timeout (in milliseconds)

- **VmbErrorSuccess:** If no error
- **VmbErrorTimeout:** Call timed out
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid

16.8 VmbCaptureQueueFlush()

Flush the capture queue.

	Type	Name	Description
in	`const VmbHandle_t`	cameraHandle	Handle of the camera to flush

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid

Note

All the currently queued frames will be returned to the user, leaving no frames in the input queue. After this call, no frame notification will occur until frames are queued again.

17 Interface Enumeration & Information

17.1 VmbInterfacesList()

List all the interfaces currently visible to VimbaC.

	Type	Name	Description
out	`VmbInterfaceInfo_t*`	pInterfaceInfo	Array of VmbInterfaceInfo_t, allocated by the caller. The interface list is copied here. May be NULL.
in	`VmbUint32_t`	listLength	Number of entries in the caller's pList array
out	`VmbUint32_t*`	pNumFound	Number of interfaces found (may be more than listLength!) returned here. May be NULL.
in	`VmbUint32_t`	sizeofInterfaceInfo	Size of one VmbInterfaceInfo_t entry

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorStructSize:** The given struct size is not valid for this API version
- **VmbErrorMoreData:** More data was returned than space was provided
- **VmbErrorBadParameter:** pInterfaceInfo and pNumFound were both NULL

Note

All the interfaces known via GenICam TransportLayers are listed by this command and filled into the provided array. Interfaces may correspond to adapter cards or frame grabber cards or, in the case of FireWire to the whole 1394 infrastructure, for instance. If "pInterfaceInfo" is NULL on entry, only the number of interfaces is returned in "pNumFound".

17.2 VmbInterfaceOpen()

Open an interface handle for feature access.

	Type	Name	Description
in	`const char*`	idString	The unique ID of the interface to get the handle for
out	`VmbHandle_t*`	pInterfaceHandle	The handle for this interface.

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorNotFound:** The designated interface cannot be found
- **VmbErrorBadParameter:** pInterfaceHandle was NULL

Note

An interface can be opened if interface-specific control or information is required, e.g. the number of devices attached to a specific interface. Access is then possible via feature access methods.

17.3 VmbInterfaceClose()

Close an interface.

	Type	Name	Description
in	`const VmbHandle_t`	interfaceHandle	The handle of the interface to close.

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid

Note

After configuration of the interface, close it by calling this function.

18 Ancillary data

18.1 VmbAncillaryDataOpen()

Get a working handle to allow access to the elements of the ancillary data via feature access.

	Type	Name	Description
in	`VmbFrame_t*`	pFrame	Pointer to a filled frame
out	`VmbHandle_t*`	pAncillaryDataHandle	Handle to the ancillary data inside the frame

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command

Note

This function can only succeed if the given frame has been filled by the API.

18.2 VmbAncillaryDataClose()

Destroy the working handle to the ancillary data inside a frame.

	Type	Name	Description
in	`VmbHandle_t`	ancillaryDataHandle	Handle to ancillary frame data

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid

Note

After reading the ancillary data and before re-queuing the frame, ancillary data must be closed.

19 Raw memory/register access

19.1 VmbMemoryRead()

Read an array of bytes.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that allows memory access
in	`VmbUint64_t`	address	Address to be used for this read operation
in	`VmbUint32_t`	bufferSize	Size of the data buffer to read
out	`char*`	dataBuffer	Buffer to be filled
out	`VmbUint32_t*`	pSizeComplete	Size of the data actually read

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode

19.2 VmbMemoryWrite()

Write an array of bytes.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that allows memory access
in	`VmbUint64_t`	address	Address to be used for this read operation
in	`VmbUint32_t`	bufferSize	Size of the data buffer to write
in	`const char*`	dataBuffer	Data to write
out	`VmbUint32_t*`	pSizeComplete	Number of bytes successfully written; if an error occurs this is less than size

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorMoreData:** Not all data was written; see pSizeComplete value for the number of bytes written

19.3 VmbRegistersRead()

Read an array of registers.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that allows register access
in	`VmbUint32_t`	readCount	Number of registers to be read
in	`const VmbUint64_t*`	pAddressArray	Array of addresses to be used for this read operation
out	`VmbUint64_t*`	pDataArray	Array of registers to be used for this read operation
out	`VmbUint32_t*`	pNumCompleteReads	Number of reads completed

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorIncomplete:** Not all the requested reads could be completed

Note

Two arrays of data must be provided: an array of register addresses and one for corresponding values to be read. The registers are read consecutively until an error occurs or all registers are written successfully.

19.4 VmbRegistersWrite()

Write an array of registers.

	Type	Name	Description
in	`const VmbHandle_t`	handle	Handle for a module that allows register access
in	`VmbUint32_t`	writeCount	Number of registers to be written
in	`const VmbUint64_t*`	pAddressArray	Array of addresses to be used for this write operation
in	`const VmbUint64_t*`	pDataArray	Array of reads to be used for this write operation
out	`VmbUint32_t*`	pNumCompleteWrites	Number of writes completed

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorIncomplete:** Not all the requested writes could be completed

Note

Two arrays of data must be provided: an array of register addresses and one with the corresponding values to be written to these addresses. The registers are written consecutively until an error occurs or all registers are written successfully.