



AVT SOFTWARE DEVELOPMENT KIT

**Vimba C++ API**  
**Function Reference Manual**

V1.2  
2013-Aug-28

## 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 08/2013

All rights reserved.

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](mailto:info@alliedvisiontec.com)

# Contents

<b>1</b>	<b>Contacting Allied Vision Technologies</b>	<b>4</b>
<b>2</b>	<b>Introduction</b>	<b>5</b>
2.1	Conventions used in this manual . . . . .	5
2.1.1	Styles . . . . .	5
2.1.2	Symbols . . . . .	5
<b>3</b>	<b>VimbaSystem</b>	<b>6</b>
3.1	GetInstance() . . . . .	6
3.2	QueryVersion() . . . . .	6
3.3	Startup() . . . . .	6
3.4	Shutdown() . . . . .	6
3.5	GetInterfaces() . . . . .	7
3.6	GetInterfaceByID() . . . . .	7
3.7	OpenInterfaceByID() . . . . .	7
3.8	GetCameras() . . . . .	8
3.9	GetCameraByID() . . . . .	8
3.10	OpenCameraByID() . . . . .	9
3.11	RegisterCameraListObserver() . . . . .	9
3.12	UnregisterCameraListObserver() . . . . .	9
3.13	RegisterInterfaceListObserver() . . . . .	9
3.14	UnregisterInterfaceListObserver() . . . . .	10
3.15	RegisterCameraFactory() . . . . .	10
3.16	UnregisterCameraFactory() . . . . .	10
<b>4</b>	<b>Interface</b>	<b>11</b>
4.1	Open() . . . . .	11
4.2	Close() . . . . .	11
4.3	GetID() . . . . .	11
4.4	GetType() . . . . .	11
4.5	GetName() . . . . .	12
4.6	GetSerialNumber() . . . . .	12
4.7	GetPermittedAccess() . . . . .	12
<b>5</b>	<b>FeatureContainer</b>	<b>13</b>
5.1	FeatureContainer constructor . . . . .	13
5.2	FeatureContainer destructor . . . . .	13
5.3	GetFeatureByName() . . . . .	13
5.4	GetFeatures() . . . . .	13
<b>6</b>	<b>IRegisterDevice</b>	<b>14</b>
6.1	ReadRegisters() . . . . .	14

6.2	ReadRegisters()	14
6.3	WriteRegisters()	14
6.4	WriteRegisters()	14
6.5	ReadMemory()	15
6.6	ReadMemory()	15
6.7	WriteMemory()	15
6.8	WriteMemory()	16
<b>7</b>	<b>IInterfaceListObserver</b>	<b>17</b>
7.1	InterfaceListChanged()	17
7.2	IInterfaceListObserver destructor	17
<b>8</b>	<b>ICameraListObserver</b>	<b>18</b>
8.1	CameraListChanged()	18
8.2	ICameraListObserver destructor	18
<b>9</b>	<b>IFrameObserver</b>	<b>19</b>
9.1	FrameReceived()	19
9.2	IFrameObserver destructor	19
<b>10</b>	<b>IFeatureObserver</b>	<b>20</b>
10.1	FeatureChanged()	20
10.2	IFeatureObserver destructor	20
<b>11</b>	<b>ICameraFactory</b>	<b>21</b>
11.1	CreateCamera()	21
11.2	ICameraFactory destructor	21
<b>12</b>	<b>Camera</b>	<b>22</b>
12.1	Camera constructor	22
12.2	Camera destructor	22
12.3	Open()	22
12.4	Close()	23
12.5	GetID()	23
12.6	GetName()	23
12.7	GetModel()	23
12.8	GetSerialNumber()	23
12.9	GetInterfaceID()	24
12.10	GetInterfaceType()	24
12.11	GetPermittedAccess()	24
12.12	ReadRegisters()	24
12.13	ReadRegisters()	24
12.14	WriteRegisters()	25
12.15	WriteRegisters()	25
12.16	ReadMemory()	25

12.17	ReadMemory()	26
12.18	WriteMemory()	26
12.19	WriteMemory()	26
12.20	AcquireSingleImage()	26
12.21	AcquireMultipleImages()	27
12.22	AcquireMultipleImages()	27
12.23	StartContinuousImageAcquisition()	27
12.24	StopContinuousImageAcquisition()	27
12.25	AnnounceFrame()	27
12.26	RevokeFrame()	28
12.27	RevokeAllFrames()	28
12.28	QueueFrame()	28
12.29	FlushQueue()	29
12.30	StartCapture()	29
12.31	EndCapture()	29
<b>13</b>	<b>Frame</b>	<b>30</b>
13.1	Frame constructor	30
13.2	Frame constructor	30
13.3	Frame destructor	30
13.4	RegisterObserver()	30
13.5	UnregisterObserver()	30
13.6	GetAncillaryData()	31
13.7	GetAncillaryData()	31
13.8	GetBuffer()	31
13.9	GetBuffer()	31
13.10	GetImage()	31
13.11	GetImage()	32
13.12	GetReceiveStatus()	32
13.13	GetImageSize()	32
13.14	GetAncillarySize()	32
13.15	GetAncillarySize()	32
13.16	GetPixelFormat()	33
13.17	GetWidth()	33
13.18	GetHeight()	33
13.19	GetOffsetX()	33
13.20	GetOffsetY()	33
13.21	GetFrameID()	34
13.22	GetTimeStamp()	34
<b>14</b>	<b>Feature</b>	<b>35</b>
14.1	GetValue()	35
14.2	GetValue()	35
14.3	GetValue()	35

14.4 GetValue()	35
14.5 GetValue()	35
14.6 GetValue()	35
14.7 GetValues()	36
14.8 GetValues()	36
14.9 GetEntry()	36
14.10GetEntries()	36
14.11GetRange()	36
14.12GetRange()	36
14.13SetValue()	37
14.14SetValue()	37
14.15SetValue()	37
14.16SetValue()	37
14.17SetValue()	37
14.18SetValue()	37
14.19GetIncrement()	38
14.20IsValueAvailable()	38
14.21IsValueAvailable()	38
14.22RunCommand()	38
14.23IsCommandDone()	38
14.24GetName()	39
14.25GetDisplayName()	39
14.26GetDataType()	39
14.27GetFlags()	39
14.28GetCategory()	39
14.29GetPollingTime()	39
14.30GetUnit()	39
14.31GetRepresentation()	40
14.32GetVisibility()	40
14.33GetToolTip()	40
14.34GetDescription()	40
14.35GetSFNCNamespace()	40
14.36GetAffectedFeatures()	40
14.37GetSelectedFeatures()	41
14.38IsReadable()	41
14.39IsWritable()	41
14.40IsStreamable()	41
14.41RegisterObserver()	41
14.42UnregisterObserver()	41
<b>15 EnumEntry</b>	<b>42</b>
15.1 EnumEntry constructor	42
15.2 EnumEntry constructor	42

15.3 EnumEntry destructor . . . . .	42
15.4 GetName() . . . . .	42
15.5 GetDisplayName() . . . . .	42
15.6 GetDescription() . . . . .	42
15.7 GetTooltip() . . . . .	43
15.8 GetValue() . . . . .	43
15.9 GetVisibility() . . . . .	43
15.10 GetSNFCNamespace() . . . . .	43
<b>16 AncillaryData</b>	<b>44</b>
16.1 Open() . . . . .	44
16.2 Close() . . . . .	44
16.3 GetBuffer() . . . . .	44
16.4 GetBuffer() . . . . .	44
16.5 GetBuffer() . . . . .	45

# 1 Contacting Allied Vision Technologies

## Note



- **Technical Information**  
<http://www.alliedvisiontec.com>
- **Support**  
[support@alliedvisiontec.com](mailto: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](mailto: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](mailto: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](mailto: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](mailto: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](mailto: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	<b>bold</b>
Courier	Code listings etc.	Input
Upper case	Constants	CONSTANT
Italics	Modes, fields	<i>Mode</i>
Parentheses and/or blue	Links	( <a href="#">Link</a> )

#### 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 VimbaSystem

### 3.1 GetInstance()

Returns a reference to the singleton.

- **VimbaSystem&**

### 3.2 QueryVersion()

Retrieve the version number of VmbAPI.

Type	Name	Description
out	VmbVersionInfo_t&	version
		Reference to the struct where version information is copied

- **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

### 3.3 Startup()

Initialize the VmbApi module.

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

#### Note



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

### 3.4 Shutdown()

Perform a shutdown on the API module.

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

**Note**

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

## 3.5 GetInterfaces()

List all the interfaces currently visible to VmbApi.

Type	Name	Description
<b>out</b> InterfacePtrVector&	Interfaces	Vector of shared pointer to Interface object

- **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**

All the interfaces known via a GenTL are listed by this command and filled into the vector provided. If the vector is not empty, new elements will be appended. Interfaces may be adapter cards or frame grabber cards, for instance.

## 3.6 GetInterfaceByID()

Gets a specific interface identified by an ID.

Type	Name	Description
<b>out</b> InterfacePtr&	pInterface	Shared pointer to Interface object

- **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**

An interface known via a GenTL is listed by this command and filled into the pointer provided. Interface may be adapter card or frame grabber card, for instance.

## 3.7 OpenInterfaceByID()

Open an interface for feature access.

Type	Name	Description
<b>in</b> const char*	pID	The unique ID of the interface to get
<b>out</b> InterfacePtr&	pInterface	A shared pointer to the interface

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

**Note**

An interface can be opened if interface-specific control is required, such as I/O pins on a frame grabber card. Control is then possible via feature access methods.

## 3.8 GetCameras()

Retrieve a list of all cameras.

Type	Name	Description
<b>out</b> CameraPtrVector&	rCameras	Vector of shared pointer to Camera object

- **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**

A camera known via a GenTL is listed by this command and filled into the pointer provided.

## 3.9 GetCameraByID()

Gets a specific camera identified by an ID. The returned camera is still closed.

Type	Name	Description
<b>out</b> CameraPtr&	pCamera	Shared pointer to camera object

- **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**

A camera known via a GenTL is listed by this command and filled into the pointer provided. Only static properties of the camera can be fetched until the camera has been opened. "pID" might be one of the following: "169.254.12.13" for an IP address, "000F314C4BE5" for a MAC address or "1234567890" for a plain serial number.

## 3.10 OpenCameraByID()

Gets a specific camera identified by an ID. The returned camera is already open.

Type	Name	Description
<b>in</b> const char*	pID	The unique ID of the camera to get
<b>in</b> VmbAccessModeType	eAccessMode	The requested access mode
<b>out</b> CameraPtr&	pCamera	A shared pointer to the camera

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

### Note



A camera can be opened if camera-specific control is required, such as I/O pins on a frame grabber card. Control is then possible via feature access methods. "pID" might be one of the following: "169.254.12.13" for an IP address, "000F314C4BE5" for a MAC address or "1234567890" for a plain serial number.

## 3.11 RegisterCameraListObserver()

Registers an instance of camera observer who's CameraListChanged() method gets called as soon as a camera is plugged in, plugged out or changes its access status

Type	Name	Description
<b>in</b> const ICameraListObserverPtr	&pObserver	A shared pointer to an object derived from ICameraListObserver

- **VmbErrorSuccess:** If no error
- **VmbErrorInvalidCall:** If the very same observer is already registered

## 3.12 UnregisterCameraListObserver()

Unregisters a camera observer

Type	Name	Description
<b>in</b> const ICameraListObserverPtr	&pObserver	A shared pointer to an object derived from ICameraListObserver

- **VmbErrorSuccess:** If no error
- **VmbErrorNotFound:** If the observer is not registered

## 3.13 RegisterInterfaceListObserver()

Registers an instance of interface observer whose InterfaceListChanged() method gets called as soon as an interface is plugged in, plugged out, or changes its access status

Type	Name	Description
<b>in</b> const IInterfaceListObserverPtr	&pObserver	A shared pointer to an object derived from IInterfaceListObserver

- **VmbErrorSuccess:** If no error
- **VmbErrorInvalidCall:** If the very same observer is already registered

## 3.14 UnregisterInterfaceListObserver()

Unregisters an interface observer

Type	Name	Description
<b>in</b> const IInterfaceListObserverPtr	&pObserver	A shared pointer to an object derived from IInterfaceListObserver

- **VmbErrorSuccess:** If no error
- **VmbErrorNotFound:** If the observer is not registered

## 3.15 RegisterCameraFactory()

Registers an instance of camera factory. When a custom camera factory is registered, all instances of type camera will be set up accordingly.

Type	Name	Description
<b>in</b> const ICameraFactoryPtr	&cameraFactory	A shared pointer to an object derived from ICameraFactory

- **VmbErrorSuccess:** If no error

## 3.16 UnregisterCameraFactory()

Unregisters the camera factory. After unregistering the default camera class is used.

- **VmbErrorSuccess:** If no error

## 4 Interface

### 4.1 Open()

Open an interface handle for feature access.

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

#### Note



An interface can be opened if interface-specific control is required, such as I/O pins on a frame grabber card. Control is then possible via feature access methods.

### 4.2 Close()

Close an interface.

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

#### Note



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

### 4.3 GetID()

Gets the ID of an interface.

Type	Name	Description
out std::string	&ID	The ID of the interface

- **VmbErrorSuccess:** If no error

### 4.4 GetType()

Gets the type, e.g. FireWire, Ethernet, USB, of an interface.

Type	Name	Description
out VmbInterfaceType	&type	The type of the interface

- **VmbErrorSuccess:** If no error

## 4.5 GetName()

Gets the name of an interface.

Type	Name	Description
<b>out</b> std::string	&name	The name of the interface

- **VmbErrorSuccess:** If no error

## 4.6 GetSerialNumber()

Gets the serial number of an interface.

Type	Name	Description
<b>out</b> std::string	&serialNumber	The serial number of the interface

- **VmbErrorSuccess:** If no error

## 4.7 GetPermittedAccess()

Gets the access mode of an interface.

Type	Name	Description
<b>out</b> VmbAccessModeType	&accessMode	The possible access mode of the interface

- **VmbErrorSuccess:** If no error



## 5 FeatureContainer

### 5.1 FeatureContainer constructor

Creates an instance of class FeatureContainer

### 5.2 FeatureContainer destructor

Destroys an instance of class FeatureContainer

### 5.3 GetFeatureByName()

Gets one particular feature of a feature container (e.g. a camera)

Type	Name	Description
<b>in</b> const char	*name	The name of the feature to get
<b>out</b> FeaturePtr	&pFeature	The queried feature

### 5.4 GetFeatures()

Gets all features of a feature container (e.g. a camera)

Type	Name	Description
<b>out</b> FeaturePtrVector	&features	The container for all queried features

#### Note



Once queried this information remains static throughout the object's lifetime

## 6 IRegisterDevice

### 6.1 ReadRegisters()

Reads one or more registers consecutively. The number of registers to be read is determined by the number of provided addresses.

Type	Name	Description
<b>in</b> Uint64Vector	&addresses	A list of register addresses
<b>out</b> Uint64Vector	&buffer	The returned data as vector

- **VmbErrorSuccess:** If all requested registers have been read
- **VmbErrorIncomplete:** If at least one, but not all registers have been read. See overload `ReadRegisters( const Uint64Vector &addresses, Uint64Vector &buffer, VmbUint32_t &completeReads )`.

### 6.2 ReadRegisters()

Reads one or more registers consecutively. The number of registers to be read is determined by the number of provided addresses.

Type	Name	Description
<b>in</b> const Uint64Vector	&addresses	A list of register addresses
<b>out</b> Uint64Vector	&buffer	The returned data as vector
<b>out</b> VmbUint64_t	&completeReads	The number of successfully read registers

- **VmbErrorSuccess:** If all requested registers have been read
- **VmbErrorIncomplete:** If at least one, but not all registers have been read.

### 6.3 WriteRegisters()

Writes one or more registers consecutively. The number of registers to be written is determined by the number of provided addresses.

Type	Name	Description
<b>in</b> Uint64Vector	&addresses	A list of register addresses
<b>in</b> Uint64Vector	&buffer	The data to write as vector

- **VmbErrorSuccess:** If all requested registers have been written
- **VmbErrorIncomplete:** If at least one, but not all registers have been written. See overload `WriteRegisters( const Uint64Vector &addresses, Uint64Vector &buffer, VmbUint32_t &completeWrites )`.

### 6.4 WriteRegisters()

Writes one or more registers consecutively. The number of registers to be written is determined by the number of provided addresses.

Type	Name	Description
<b>in</b> Uint64Vector	&addresses	A list of register addresses
<b>in</b> Uint64Vector	&buffer	The data to write as vector
<b>out</b> VmbUint64_t	&completeWrites	The number of successfully read registers

- **VmbErrorSuccess:** If all requested registers have been written
- **VmbErrorIncomplete:** If at least one, but not all registers have been written.

## 6.5 ReadMemory()

Reads a block of memory. The number of bytes to read is determined by the size of the provided buffer.

Type	Name	Description
<b>in</b> VmbUint64_t	&address	The address to read from
<b>out</b> UcharVector	&buffer	The returned data as vector

- **VmbErrorSuccess:** If all requested bytes have been read
- **VmbErrorIncomplete:** If at least one, but not all bytes have been read. See overload ReadMemory( const VmbUint64\_t &address, UcharVector &buffer, VmbUint32\_t &completeReads ).

## 6.6 ReadMemory\_()

Reads a block of memory. The number of bytes to read is determined by the size of the provided buffer.

Type	Name	Description
<b>in</b> VmbUint64_t	&address	The address to read from
<b>out</b> UcharVector	&buffer	The returned data as vector
<b>out</b> VmbUint32_t	&completeReads	The number of successfully read bytes

- **VmbErrorSuccess:** If all requested bytes have been read
- **VmbErrorIncomplete:** If at least one, but not all bytes have been read.

## 6.7 WriteMemory()

Writes a block of memory. The number of bytes to write is determined by the size of the provided buffer.

Type	Name	Description
<b>in</b> VmbUint64_t	&address	The address to write to
<b>in</b> UcharVector	&buffer	The data to write as vector

- **VmbErrorSuccess:** If all requested bytes have been written
- **VmbErrorIncomplete:** If at least one, but not all bytes have been written. See overload ReadMemory( const VmbUint64\_t &address, UcharVector &buffer, VmbUint32\_t &completeWrites ).

## 6.8 WriteMemory()

Writes a block of memory. The number of bytes to write is determined by the size of the provided buffer.

Type	Name	Description
<b>in</b> VmbUInt64_t	&address	The address to write to
<b>in</b> UcharVector	&buffer	The data to write as vector
<b>out</b> VmbUInt32_t	&completeWrites	The number of successfully written bytes

- **VmbErrorSuccess:** If all requested bytes have been written
- **VmbErrorIncomplete:** If at least one, but not all bytes have been written.

## 7 IInterfaceListObserver

### 7.1 InterfaceListChanged()

The event handler function that gets called whenever an IInterfaceListObserver is triggered.

Type	Name	Description
<b>out</b> InterfacePtr	pInterface	The interface that triggered the event
<b>out</b> UpdateTriggerType	reason	The reason why the callback routine was triggered

### 7.2 IInterfaceListObserver destructor

Destroys an instance of class IInterfaceListObserver

## 8 ICameraListObserver

### 8.1 CameraListChanged()

The event handler function that gets called whenever an ICameraListObserver is triggered. This occurs most likely when a camera was plugged in or out

Type	Name	Description
<b>out</b> CameraPtr	pCam	The camera that triggered the event
<b>out</b> UpdateTriggerType	reason	The reason why the callback routine was triggered (e.g., a new camera was plugged in)

### 8.2 ICameraListObserver destructor

Destroys an instance of class ICameraListObserver

## 9 IFrameObserver

### 9.1 FrameReceived()

The event handler function that gets called whenever a new frame is received

Type	Name	Description
in const FramePtr	pFrame	The frame that was received

### 9.2 IFrameObserver destructor

Destroys an instance of class IFrameObserver

## 10 IFeatureObserver

### 10.1 FeatureChanged()

The event handler function that gets called whenever a feature's has changed

Type	Name	Description
in const FeaturePtr&	pFeature	The frame that has changed

### 10.2 IFeatureObserver destructor

Destroys an instance of class IFeatureObserver



# 11 ICameraFactory

## 11.1 CreateCamera()

FactorMethod to create a camera that implements Camera.h

Type	Name	Description
<b>in</b> const char	*pCameraID	The ID of the camera
<b>in</b> const char	*pCameraName	The name of the camera
<b>in</b> const char	*pCameraModel,	The model name of the camera
<b>in</b> const char	*pCameraSerialNumber	The serial number of the camera
<b>in</b> const char	*pInterfaceID	The ID of the interface the camera is connected to
<b>in</b> VmbInterfaceType	interfaceType	The type of the interface the camera is connected to
<b>in</b> const char	*pInterfaceName	The name of the interface
<b>in</b> const char	*pInterfaceSerialNumber	The serial number of the interface
<b>in</b> VmbAccessModeType	interfacePermittedAccess	The access privileges for the interface

### Note



The ID of the camera might be one of the following: "IP:169.254.12.13", "MAC:00f3100001", or a plain serial number: "1234567890".

## 11.2 ICameraFactory destructor

Destroys an instance of class camera

## 12 Camera

### 12.1 Camera constructor

Creates an instance of class camera

Type	Name	Description
in const char	*pID	The ID of the camera
in const char	*pName	The name of the camera
in const char	*pModel,	The model name of the camera
in const char	*pSerialNumber	The serial number of the camera
in const char	*pInterfaceID	The ID of the interface the camera is connected to
in VmbInterfaceType	interfaceType	The type of the interface the camera is connected to

#### Note



The ID might be one of the following: "IP:169.254.12.13", "MAC:000f31000001", or a plain serial number: "1234567890".

### 12.2 Camera destructor

Destroys an instance of class camera

#### Note



Destroying a camera implicitly closes it beforehand.

### 12.3 Open()

Opens the specified camera.

Type	Name	Description
in VmbAccessMode_t	accessMode	Access mode determines the amount of control you have on the camera

- **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.

## 12.4 Close()

Close the specified camera.

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

### Note



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

## 12.5 GetID()

Gets the ID of a camera.

Type	Name	Description
<b>out</b> std::string	&ID	The ID of the camera

- **VmbErrorSuccess:** If no error

## 12.6 GetName()

Gets the name of a camera.

Type	Name	Description
<b>out</b> std::string	&name	The name of the camera

- **VmbErrorSuccess:** If no error

## 12.7 GetModel()

Gets the model name of a camera.

Type	Name	Description
<b>out</b> std::string	&model	The model name of the camera

- **VmbErrorSuccess:** If no error

## 12.8 GetSerialNumber()

Gets the serial number of a camera.

Type	Name	Description
<b>out</b> std::string	&serialNumber	The serial number of the camera

- **VmbErrorSuccess:** If no error

## 12.9 GetInterfaceID()

Gets the interface ID of a camera.

Type	Name	Description
<b>out</b> std::string	&interfaceID	The interface ID of the camera

- **VmbErrorSuccess:** If no error

## 12.10 GetInterfaceType()

Gets the type of the interface the camera is connected to. And therefore the type of the camera itself.

Type	Name	Description
<b>out</b> VmbInterfaceType	&interfaceType	The interface type of the camera

- **VmbErrorSuccess:** If no error

## 12.11 GetPermittedAccess()

Gets the access mode of an camera.

Type	Name	Description
<b>out</b> VmbAccessModeType	&permittedAccess	The possible access mode of the camera

- **VmbErrorSuccess:** If no error

## 12.12 ReadRegisters()

Reads one or more registers consecutively. The amount of registers to read is determined by the amount of provided addresses.

Type	Name	Description
<b>in</b> Uint64Vector	&addresses	A list of register addresses
<b>out</b> Uint64Vector	&buffer	The returned data as vector

- **VmbErrorSuccess:** If all requested registers have been read
- **VmbErrorIncomplete:** If at least one, but not all registers have been read. See overload `ReadRegisters( const Uint64Vector &addresses, Uint64Vector &buffer, VmbUint32_t &completeReads )`.

## 12.13 ReadRegisters()

Reads one or more registers consecutively. The amount of registers to read is determined by the amount of provided addresses.

Type	Name	Description
<b>in</b> const Uint64Vector	&addresses	A list of register addresses
<b>out</b> Uint64Vector	&buffer	The returned data as vector
<b>out</b> VmbUint64_t	&completedReads	The number of successfully read registers

- **VmbErrorSuccess:** If all requested registers have been read
- **VmbErrorIncomplete:** If at least one, but not all registers have been read.

## 12.14 WriteRegisters()

Writes one or more registers consecutively. The amount of registers to write is determined by the amount of provided addresses.

Type	Name	Description
<b>in</b> Uint64Vector	&addresses	A list of register addresses
<b>in</b> Uint64Vector	&buffer	The data to write as vector

- **VmbErrorSuccess:** If all requested registers have been written
- **VmbErrorIncomplete:** If at least one, but not all registers have been written. See overload WriteRegisters( const Uint64Vector &addresses, Uint64Vector &buffer, VmbUint32\_t &completeWrites ).

## 12.15 WriteRegisters()

Writes one or more registers consecutively. The amount of registers to write is determined by the amount of provided addresses.

Type	Name	Description
<b>in</b> Uint64Vector	&addresses	A list of register addresses
<b>in</b> Uint64Vector	&buffer	The data to write as vector
<b>out</b> VmbUint64_t	&completedWrites	The number of successfully read registers

- **VmbErrorSuccess:** If all requested registers have been written
- **VmbErrorIncomplete:** If at least one, but not all registers have been written.

## 12.16 ReadMemory()

Reads a block of memory. The number of bytes to read is determined by the size of the provided buffer.

Type	Name	Description
<b>in</b> VmbUint64_t	&address	The address to read from
<b>out</b> UcharVector	&buffer	The returned data as vector

- **VmbErrorSuccess:** If all requested bytes have been read
- **VmbErrorIncomplete:** If at least one, but not all bytes have been read. See overload ReadMemory( const VmbUint64\_t &address, UcharVector &buffer, VmbUint32\_t &completeReads ).

## 12.17 ReadMemory()

Reads a block of memory. The number of bytes to read is determined by the size of the provided buffer.

Type	Name	Description
<b>in</b> VmbUInt64_t	&address	The address to read from
<b>out</b> UcharVector	&buffer	The returned data as vector
<b>out</b> VmbUInt32_t	&completeReads	The number of successfully read bytes

- **VmbErrorSuccess:** If all requested bytes have been read
- **VmbErrorIncomplete:** If at least one, but not all bytes have been read.

## 12.18 WriteMemory()

Writes a block of memory. The number of bytes to write is determined by the size of the provided buffer.

Type	Name	Description
<b>in</b> VmbUInt64_t	&address	The address to write to
<b>in</b> UcharVector	&buffer	The data to write as vector

- **VmbErrorSuccess:** If all requested bytes have been written
- **VmbErrorIncomplete:** If at least one, but not all bytes have been written. See overload `ReadMemory( const VmbUInt64_t &address, UcharVector &buffer, VmbUInt32_t &completeWrites )`.

## 12.19 WriteMemory()

Writes a block of memory. The number of bytes to write is determined by the size of the provided buffer.

Type	Name	Description
<b>in</b> VmbUInt64_t	&address	The address to write to
<b>in</b> UcharVector	&buffer	The data to write as vector
<b>out</b> VmbUInt32_t	&completeWrites	The number of successfully written bytes

- **VmbErrorSuccess:** If all requested bytes have been written
- **VmbErrorIncomplete:** If at least one, but not all bytes have been written.

## 12.20 AcquireSingleImage()

Gets one image synchronously.

Type	Name	Description
<b>out</b> FramePtr	&frame	The frame that gets filled
<b>in</b> VmbUInt32_t	timeout	The time to wait until the frame got filled

## 12.21 AcquireMultipleImages()

Gets multiple images synchronously.

Type	Name	Description
<b>out</b> FramePtrVector	&frames	The frames that get filled
<b>in</b> VmbUInt32_t	timeout	The time to wait until one frame got filled

## 12.22 AcquireMultipleImages()

Gets multiple images synchronously.

Type	Name	Description
<b>out</b> FramePtrVector	&frames	The frames that get filled
<b>in</b> VmbUInt32_t	timeout	The time to wait until one frame got filled
<b>out</b> VmbUInt32_t	&numFramesCompleted	The number of frames that were filled completely

### Note



The size of the frame vector determines the number of frames to use

## 12.23 StartContinuousImageAcquisition()

Starts streaming and allocates the needed frames

Type	Name	Description
<b>in</b> VmbUInt32_t	bufferCount	The number of frames to use
<b>out</b> const IFrameObserverPtr	&pObserver	The observe to use on arrival of new frames

- **VmbErrorSuccess:** If no error
- **VmbErrorDeviceNotOpen:** The camera has not been opened before
- **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

## 12.24 StopContinuousImageAcquisition()

Stops streaming and frees the needed frames

## 12.25 AnnounceFrame()

Announces a frame to the API that may be queued for frame capturing later

Type	Name	Description
<b>in</b> const FramePtr	&pFrame	Shared pointer to a frame to announce

- **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 in consideration by the API.

## 12.26 RevokeFrame()

Revoke a frame from the API.

Type	Name	Description
<b>in</b> const FramePtr	&pFrame	Shared pointer to a frame that is 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.

## 12.27 RevokeAllFrames()

Revoke all frames assigned to this certain camera

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

## 12.28 QueueFrame()

Queues a frame that may be filled during frame capturing

Type	Name	Description
<b>in</b> const FramePtr	&frame	A shared pointer to a frame

- **VmbErrorSuccess:** If no error



- **VmbErrorApiNotStarted:** Startup 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 `AnnounceFrame()` before, the application has to take care that the frame is also revoked by calling `RevokeFrame()` or `RevokeAll()` when cleaning up.

## 12.29 FlushQueue()

Flushes the capture queue.

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** Startup 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.

## 12.30 StartCapture()

Prepare the API for incoming frames from this 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

## 12.31 EndCapture()

Stop the API from being able to receive frames from this 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 frame queue is flushed - The frame callback will not be called any more

## 13 Frame

### 13.1 Frame constructor

Creates an instance of class Frame

Type	Name	Description
in VmbInt64_t	bufferSize	The size of the underlying buffer

### 13.2 Frame constructor

Creates an instance of class Frame

Type	Name	Description
in VmbUchar_t	*pBuffer	A pointer to an allocated buffer
in VmbInt64_t	bufferSize	The size of the underlying buffer

### 13.3 Frame destructor

Destroys an instance of class Frame

### 13.4 RegisterObserver()

Registers an observer that will be called whenever a new frame arrives

Type	Name	Description
in IFrameObserverPtr	&observer	An object that implements the IObserver interface

- **VmbErrorSuccess:** If no error
- **VmbErrorResources:** The observer was in use

#### Note



As new frames arrive, the observer's FrameReceived method will be called. Only one observer can be registered.

### 13.5 UnregisterObserver()

Unregisters the observer that was called whenever a new frame arrived

## 13.6 GetAncillaryData()

Returns the part of a frame that describes the chunk data as an object

- **VmbErrorSuccess:** If no error
- **VmbErrorNotFound:** No chunk data present

Type	Name	Description
<b>out</b> AncillaryDataPtr	&ancillaryData	The wrapped chunk data

## 13.7 GetAncillaryData()

Returns the part of a frame that describes the chunk data as an object

- **VmbErrorSuccess:** If no error
- **VmbErrorNotFound:** No chunk data present

Type	Name	Description
<b>out</b> ConstAncillaryDataPtr	&ancillaryData	The wrapped chunk data

## 13.8 GetBuffer()

Returns the complete buffer including image and chunk data

- **VmbErrorSuccess:** If no error

Type	Name	Description
<b>out</b> VmbUchar_t	*pBuffer	A pointer to the buffer

## 13.9 GetBuffer()

Returns the complete buffer including image and chunk data

- **VmbErrorSuccess:** If no error

Type	Name	Description
<b>out</b> const VmbUchar_t	*pBuffer	A pointer to the buffer

## 13.10 GetImage()

Returns only the image data

- **VmbErrorSuccess:** If no error

Type	Name	Description
<b>out</b> VmbUchar_t	*pBuffer	A pointer to the buffer

## 13.11 GetImage()

Returns only the image data

- **VmbErrorSuccess:** If no error

Type	Name	Description
out const VmbUchar_t	*pBuffer	A pointer to the buffer

## 13.12 GetReceiveStatus()

Returns the receive status of a frame

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbFrameStatusType	&status	The receive status

## 13.13 GetImageSize()

Returns the memory size of the image

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbUInt32_t	&imageSize	The size in bytes

## 13.14 GetAncillarySize()

Returns memory size of the chunk data

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbUInt32_t	&ancillarySize	The size in bytes

## 13.15 GetAncillarySize()

Returns the memory size of the frame buffer holding both the image data and the ancillary data

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbUInt32_t	&bufferSize	The size in bytes

## 13.16 GetPixelFormat()

Returns the GeV compliant pixel format

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbPixelFormatType	&pixelFormat	The GeV pixel format

## 13.17 GetWidth()

Returns the width of the image

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbUInt32_t	&width	The width in pixels

## 13.18 GetHeight()

Returns the height of the image

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbUInt32_t	&height	The height in pixels

## 13.19 GetOffsetX()

Returns the x offset of the image

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbUInt32_t	&offsetX	The x offset in pixels

## 13.20 GetOffsetY()

Returns the y offset of the image

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbUInt32_t	&offsetY	The y offset in pixels

## 13.21 GetFrameID()

Returns the frame ID

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbUInt64_t	&frameID	The frame ID

## 13.22 GetTimeStamp()

Returns the time stamp

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbUInt64_t	&timestamp	The time stamp

## 14 Feature

### 14.1 GetValue()

Queries the value of a feature of type VmbInt64

Type	Name	Description
out VmbInt64_t&	value	The feature's value

### 14.2 GetValue()

Queries the value of a feature of type double

Type	Name	Description
out double&	value	The feature's value

### 14.3 GetValue()

Queries the value of a feature of type string

Type	Name	Description
out std::string&	value	The feature's value

### 14.4 GetValue()

Queries the value of a feature of type bool

Type	Name	Description
out bool&	value	The feature's value

### 14.5 GetValue()

Queries the value of a feature of type UcharVector

Type	Name	Description
out UcharVector&	value	The feature's value

### 14.6 GetValue()

Queries the value of a feature of type const UcharVector

Type	Name	Description
out const UcharVector&	value	The feature's value
out VmbUInt32&	sizeFilled	The amount of actually received values

## 14.7 GetValues()

Queries the values of a feature of type Int64Vector

Type	Name	Description
<b>out</b> Int64Vector&	values	The feature's values

## 14.8 GetValues()

Queries the values of a feature of type StringVector

Type	Name	Description
<b>out</b> StringVector&	values	The feature's values

## 14.9 GetEntry()

Queries a single enum entry of a feature of type Enumeration

Type	Name	Description
<b>out</b> EnumEntry&	entry	An enum feature's enum entry
<b>in</b> const char*	pEntryName	The name of the enum entry

## 14.10 GetEntries()

Queries all enum entries of a feature of type Enumeration

Type	Name	Description
<b>out</b> EnumEntryVector	entries	An enum feature's enum entries

## 14.11 GetRange()

Queries the range of a feature of type double

Type	Name	Description
<b>out</b> double&	min	The feature's min value
<b>out</b> double&	max	The feature's max value

## 14.12 GetRange()

Queries the range of a feature of type VmbInt64

Type	Name	Description
<b>out</b> VmbInt64&	min	The feature's min value
<b>out</b> VmbInt64&	max	The feature's max value



## 14.13 SetValue()

Sets the value of a feature of type VmbInt32

Type	Name	Description
<b>in</b> const VmbInt32_t&	value	The feature's value

## 14.14 SetValue()

Sets the value of a feature of type VmbInt64

Type	Name	Description
<b>in</b> const VmbInt64&	value	The feature's value

## 14.15 SetValue()

Sets the value of a feature of type double

Type	Name	Description
<b>in</b> const double&	value	The feature's value

## 14.16 SetValue()

Sets the value of a feature of type char\*

Type	Name	Description
<b>in</b> const char*	pValue	The feature's value

## 14.17 SetValue()

Sets the value of a feature of type bool

Type	Name	Description
<b>in</b> bool	value	The feature's value

## 14.18 SetValue()

Sets the value of a feature of type UcharVector

Type	Name	Description
<b>in</b> const UcharVector&	value	The feature's value

## 14.19 GetIncrement()

Gets the increment of a feature of type VmbInt64

Type	Name	Description
<b>out</b> VmbInt64_t&	increment	The feature's increment

## 14.20 IsValueAvailable()

Indicates whether an existing enumeration-value is currently available. An enumeration-value might not be selectable due to the camera's current configuration.

Type	Name	Description
<b>in</b> const char*	pStrValue	The enumeration-value as string
<b>out</b> bool&	available	True when the given value is available

- **VmbErrorSuccess:** If no error
- **VmbErrorInvalidValue:** If the given value is not a valid enumeration-value for this enum
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The feature is not an enumeration

## 14.21 IsValueAvailable()

Indicates whether an existing enumeration-value is currently available. An enumeration-value might not be selectable due to the camera's current configuration.

Type	Name	Description
<b>in</b> const VmbInt64_t	pStrValue	The enumeration-value as int
<b>out</b> bool&	available	True when the given value is available

- **VmbErrorSuccess:** If no error
- **VmbErrorInvalidValue:** If the given value is not a valid enumeration-value for this enum
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorInvalidAccess:** Operation is invalid with the current access mode
- **VmbErrorWrongType:** The feature is not an enumeration

## 14.22 RunCommand()

Executes a feature of type Command

## 14.23 IsCommandDone()

Indicates whether the execution of a feature of type Command has finished

Type	Name	Description
<b>out</b> bool&	isDone	True when execution has finished

## 14.24 GetName()

Queries a feature's name

Type	Name	Description
out std::string&	name	The feature's name

## 14.25 GetDisplayName()

Queries a feature's display name

Type	Name	Description
out std::string&	displayName	The feature's display name

## 14.26 GetDataType()

Queries a feature's type

Type	Name	Description
out VmbFeatureDataType&	dataType	The feature's type

## 14.27 GetFlags()

Queries a feature's access status

Type	Name	Description
out VmbFeatureFlagsType&	flags	The feature's access status

## 14.28 GetCategory()

Queries a feature's category in the feature tree

Type	Name	Description
out std::string&	category	The feature's position in the feature tree

## 14.29 GetPollingTime()

Queries a feature's polling time

Type	Name	Description
out VmbUInt32_t&	pollingTime	The interval to poll the feature

## 14.30 GetUnit()

Queries a feature's unit

Type	Name	Description
out std::string&	unit	The feature's unit

## 14.31 GetRepresentation()

Queries a feature's representation

Type	Name	Description
out std::string&	representation	The feature's representation

## 14.32 GetVisibility()

Queries a feature's visibility

Type	Name	Description
out VmbFeatureVisibilityType&	visibility	The feature's visibility

## 14.33 GetToolTip()

Queries a feature's tool tip to display in the GUI

Type	Name	Description
out std::string&	toolTip	The feature's tool tip

## 14.34 GetDescription()

Queries a feature's description

Type	Name	Description
out std::string&	description	The feature's description

## 14.35 GetSFNCNamespace()

Queries a feature's Standard Feature Naming Convention namespace

Type	Name	Description
out std::string&	SFNCNamespace	The feature's SFNC namespace

## 14.36 GetAffectedFeatures()

Queries the feature's that are dependent from the current feature

Type	Name	Description
out FeaturePtrVector&	affectedFeatures	The features that get invalidated through the current feature

## 14.37 GetSelectedFeatures()

Gets the features that get selected by the current feature

Type	Name	Description
out FeaturePtrVector&	selectedFeatures	The selected features

## 14.38 IsReadable()

Queries the read access status of a feature

Type	Name	Description
out bool&	isReadable	True when feature can be read

## 14.39 IsWritable()

Queries the write access status of a feature

Type	Name	Description
out bool&	isWritable	True when feature can be written

## 14.40 IsStreamable()

Queries whether a feature's value can be transferred as a stream

Type	Name	Description
out bool&	isStreamable	True when streamable

## 14.41 RegisterObserver()

Registers an observer that notifies the application whenever a features value changes

Type	Name	Description
out const IFeatureObserverPtr&	observer	The observer to be registered

## 14.42 UnregisterObserver()

Unregisters an observer

Type	Name	Description
out const IFeatureObserverPtr&	observer	The observer to be unregistered

## 15 EnumEntry

### 15.1 EnumEntry constructor

Creates an instance of class EnumEntry

Type	Name	Description
<b>in</b> const char	*pName	The name of the enum
<b>in</b> const char	*pDisplayName	The declarative name of the enum
<b>in</b> const char	*pDescription,	The descripton of the enum
<b>in</b> const char	*pTooltip	A tooltip that can be used by a GUI
<b>in</b> const char	*pSNFCNamespace	the SFNC namespace of the enum
<b>in</b> VmbFeatureVisibility_t	visibility	The visibility of the enum
<b>in</b> VmbInt64_t	value	The integer value of the enum

### 15.2 EnumEntry constructor

Creates an instance of class EnumEntry

### 15.3 EnumEntry destructor

Destroys an instance of class EnumEntry

### 15.4 GetName()

Gets the name of an enumeration

Type	Name	Description
<b>out</b> std::string&	name	The name of the enumeration

### 15.5 GetDisplayName()

Gets a more declarative name of an enumeration

Type	Name	Description
<b>out</b> std::string&	displayName	The display name of the enumeration

### 15.6 GetDescription()

Gets the description of an enumeration

Type	Name	Description
<b>out</b> std::string&	description	The description of the enumeration

## 15.7 GetTooltip()

Gets a tooltip that can be used as pop up help in a GUI

Type	Name	Description
<b>out</b> std::string&	tooltip	The tooltip as string

## 15.8 GetValue()

Gets the integer value of an enumeration

Type	Name	Description
<b>out</b> VmbInt64_t&	value	The integer value of the enumeration

## 15.9 GetVisibility()

Gets the visibility of an enumeration

Type	Name	Description
<b>out</b> bool&	value	True when visible otherwise false

## 15.10 GetSNFCNamespace()

Gets the standard feature naming convention namespace of the enumeration

Type	Name	Description
<b>out</b> std::string&	tooltip	The namespace as string

## 16 AncillaryData

### 16.1 Open()

Opens the ancillary data to allow access to the elements of the ancillary data via feature access.

- **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.

### 16.2 Close()

Closes the ancillary data inside a frame.

- **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.

### 16.3 GetBuffer()

Returns the underlying buffer

- **VmbErrorSuccess:** If no error

Type	Name	Description
out VmbUchar_t	*pBuffer	A pointer to the buffer

### 16.4 GetBuffer()

Returns the underlying buffer

- **VmbErrorSuccess:** If no error

Type	Name	Description
out const VmbUchar_t	*pBuffer	A pointer to the buffer



## 16.5 GetBuffer()

Returns the size of the underlying buffer

- **VmbErrorSuccess:** If no error

	Type	Name	Description
out	VmbUInt32_t	&nSize	The size of the buffer