Allied Vision

Vimba

Vimba

Vimba C++ Function Reference

1.9.1

Function reference

In this chapter you can find a complete list of all methods of the following classes/interfaces: VimbaSystem, Interface, FeatureContainer, IRegisterDevice, IInterfaceListObserver, ICameraListObserver, IFrameObserver, IFeatureObserver, ICameraFactory, Camera, Frame, Feature, EnumEntry and AncillaryData.

Methods in this chapter are always described in the same way:

- The caption states the name of the function without parameters
- The first item is a brief description
- The parameters of the function are listed in a table (with type, name, and description)
- The return values or the returned type is listed
- Finally, a more detailed description about the function is given

VimbaSystem

GetInstance()

Returns a reference to the System singleton.

- **VimbaSystem&**

QueryVersion()

Retrieve the version number of VmbAPI.

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

- **VmbErrorSuccess:** always returned

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

Startup()

Initialize the VmbAPI module.

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

On successful return, the API is initialized; this is a necessary call. This method must be called before any other VmbAPI function is run.

Shutdown()

Perform a shutdown on the API module.

- **VmbErrorSuccess:** always returned

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

GetInterfaces()

List all the interfaces currently visible to VmbAPI.

	Type	Name	Description
out	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 were returned than space was provided
- **VmbErrorInternalFault:** An internal fault occurred

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 can be adapter cards or frame grabber cards, for instance.

GetInterfaceByID()

Gets a specific interface identified by an ID.

	Type	Name	Description
in	const char*	pID	The ID of the interface to get (returned by GetInterfaces())
out	InterfacePtr&	pInterface	Shared pointer to Interface object

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

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

OpenInterfaceByID()

Open an interface for feature access.

	Type	Name	Description
in	const char*	pID	The ID of the interface to open (returned by GetInterfaces())
out	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
- **VmbErrorBadParameter:** "pID" is NULL.

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.

GetCameras()

Retrieve a list of all cameras.

	Type	Name	Description
out	CameraPtrVector&	cameras	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 were returned than space was provided

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

GetCameraByID()

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

	Type	Name	Description
in	const char*	pID	The ID of the camera to get
out	CameraPtr&	pCamera	Shared pointer to camera object

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

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 "DEV_1234567890" for an ID as reported by Vimba

OpenCameraByID()

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

	Type	Name	Description
in	const char*	pID	The unique ID of the camera to get
in	VmbAccessModeType	eAccessMode	The requested access mode
out	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
- **VmbErrorBadParameter:** "pID" is NULL.

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 "DEV_1234567890" for an ID as reported by Vimba

RegisterCameraListObserver()

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

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

- **VmbErrorSuccess:** If no error
- **VmbErrorBadParameter:** "pObserver" is NULL.
- **VmbErrorInvalidCall:** If the very same observer is already registered

UnregisterCameraListObserver()

Unregisters a camera observer

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

- **VmbErrorSuccess:** If no error
- **VmbErrorNotFound:** If the observer is not registered
- **VmbErrorBadParameter:** "pObserver" is NULL.

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
in	const	IInterfaceListObserverPtr&	pObserver	A shared pointer to an object derived from IInterfaceListObserver

- **VmbErrorSuccess:** If no error
- **VmbErrorBadParameter:** "pObserver" is NULL.
- **VmbErrorInvalidCall:** If the very same observer is already registered

UnregisterInterfaceListObserver()

Unregisters an interface observer

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

- **VmbErrorSuccess:** If no error
- **VmbErrorNotFound:** If the observer is not registered
- **VmbErrorBadParameter:** "pObserver" is NULL.

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
in	const ICameraFactoryPtr&	pCameraFactory	A shared pointer to an object derived from ICameraFactory

- **VmbErrorSuccess:** If no error
- **VmbErrorBadParameter:** "pCameraFactory" is NULL.

UnregisterCameraFactory()

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

- **VmbErrorSuccess:** If no error

Interface

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

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.

Close()

Close an interface.

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

GetID()

Gets the ID of an interface.

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

- **VmbErrorSuccess:** If no error

This information remains static throughout the object's lifetime

GetType()

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

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

- **VmbErrorSuccess:** If no error

This information remains static throughout the object's lifetime

GetName()

Gets the name of an interface.

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

- **VmbErrorSuccess:** If no error

GetSerialNumber()

Gets the serial number of an interface.

	Type	Name	Description
out	std::string&	serialNumber	The serial number of the interface

- **VmbErrorSuccess:** If no error

GetPermittedAccess()

Gets the access mode of an interface.

	Type	Name	Description
out	VmbAccessModeType&	accessMode	The possible access mode of the interface

- **VmbErrorSuccess:** If no error

FeatureContainer

FeatureContainer constructor

Creates an instance of class FeatureContainer

FeatureContainer destructor

Destroys an instance of class FeatureContainer

GetFeatureByName()

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

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

- **VmbErrorSuccess:** If no error
- **VmbErrorDeviceNotOpen:** Base feature class (e.g. Camera) was not opened.
- **VmbErrorBadParameter:** "name" is NULL.

GetFeatures()

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

	Type	Name	Description
out	FeaturePtrVector&	features	The container for all queried features

- **VmbErrorSuccess:** If no error
- **VmbErrorBadParameter:** "features" is empty.

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

IRegisterDevice

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
in	const Uint64Vector&	addresses	A list of register addresses
out	Uint64Vector&	buffer	The returned data as vector

- **VmbErrorSuccess:** If all requested registers have been read
- **VmbErrorBadParameter:** Vectors "addresses" and/or "buffer" are empty.
- **VmbErrorIncomplete:** If at least one, but not all registers have been read. See overload ReadRegisters(const Uint64Vector&, Uint64Vector&, VmbUint32_t&).

ReadRegisters()

Same as ReadRegisters(const Uint64Vector&, Uint64Vector&), but returns the number of successful read operations in case of an error.

	Type	Name	Description
in	const Uint64Vector&	addresses	A list of register addresses
out	Uint64Vector&	buffer	The returned data as vector
out	VmbUint32_t&	completedReads	The number of successfully read registers

- **VmbErrorSuccess:** If all requested registers have been read
- **VmbErrorBadParameter:** Vectors "addresses" and/or "buffer" are empty.
- **VmbErrorIncomplete:** If at least one, but not all registers have been read.

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
in	const Uint64Vector&	addresses	A list of register addresses
in	const Uint64Vector&	buffer	The data to write as vector

- **VmbErrorSuccess:** If all requested registers have been written
- **VmbErrorBadParameter:** Vectors "addresses" and/or "buffer" are empty.
- **VmbErrorIncomplete:** If at least one, but not all registers have been written. See overload WriteRegisters(const Uint64Vector&, const Uint64Vector&, VmbUint32_t&).

WriteRegisters()

Same as WriteRegisters(const Uint64Vector&, const Uint64Vector&), but returns the number of successful write operations in case of an error VmbErrorIncomplete.

	Type	Name	Description
in	const Uint64Vector&	addresses	A list of register addresses
in	const Uint64Vector&	buffer	The data to write as vector
out	VmbUint32_t&	completedWrites	The number of successfully written registers

- **VmbErrorSuccess:** If all requested registers have been written
- **VmbErrorBadParameter:** Vectors "addresses" and/or "buffer" are empty.
- **VmbErrorIncomplete:** If at least one, but not all registers have been written.

ReadMemory()

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

	Type	Name	Description
in	const VmbUint64_t&	address	The address to read from
out	UcharVector&	buffer	The returned data as vector

- **VmbErrorSuccess:** If all requested bytes have been read
- **VmbErrorBadParameter:** Vector "buffer" is empty.
- **VmbErrorIncomplete:** If at least one, but not all bytes have been read. See overload ReadMemory(const VmbUint64_t&, UcharVector&, VmbUint32_t&).

ReadMemory()

Same as ReadMemory(const Uint64Vector&, UcharVector&), but returns the number of bytes successfully read in case of an error VmbErrorIncomplete.

	Type	Name	Description
in	const VmbUint64_t&	address	The address to read from
out	UcharVector&	buffer	The returned data as vector
out	VmbUint32_t&	sizeComplete	The number of successfully read bytes

- **VmbErrorSuccess:** If all requested bytes have been read
- **VmbErrorBadParameter:** Vector "buffer" is empty.
- **VmbErrorIncomplete:** If at least one, but not all bytes have been read.

WriteMemory()

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

	Type	Name	Description
in	const VmbUint64_t&	address	The address to write to
in	const UcharVector&	buffer	The data to write as vector

- **VmbErrorSuccess:** If all requested bytes have been written
- **VmbErrorBadParameter:** Vector "buffer" is empty.
- **VmbErrorIncomplete:** If at least one, but not all bytes have been written. See overload WriteMemory(const VmbUint64_t&, const UcharVector&, VmbUint32_t&).

WriteMemory()

Same as WriteMemory(const Uint64Vector&, const UcharVector&), but returns the number of bytes successfully written in case of an error VmbErrorIncomplete.

	Type	Name	Description
in	const VmbUint64_t&	address	The address to write to
in	const UcharVector&	buffer	The data to write as vector
out	VmbUint32_t&	sizeComplete	The number of successfully written bytes

- **VmbErrorSuccess:** If all requested bytes have been written
- **VmbErrorBadParameter:** Vector "buffer" is empty.
- **VmbErrorIncomplete:** If at least one, but not all bytes have been written.

IInterfaceListObserver

InterfaceListChanged()

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

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

IInterfaceListObserver destructor

Destroys an instance of class IInterfaceListObserver

ICameraListObserver

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
out	CameraPtr	pCam	The camera that triggered the event
out	UpdateTriggerType	reason	The reason why the callback routine was triggered (e.g., a new camera was plugged in)

ICameraListObserver destructor

Destroys an instance of class ICameraListObserver

IFrameObserver

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

IFrameObserver destructor

Destroys aninstance of class IFrameObserver

IFeatureObserver

FeatureChanged()

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

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

IFeatureObserver destructor

Destroys an instance of class IFeatureObserver

ICameraFactory

CreateCamera()

Factory method to create a camera that extends the Camera class

	Type	Name	Description
in	const char*	pCameraID	The ID of the camera
in	const char*	pCameraName	The name of the camera
in	const char*	pCameraModel	The model name of the camera
in	const char*	pCameraSerialNumber	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
in	const char*	pInterfaceName	The name of the interface
in	const char*	pInterfaceSerialNumber	The serial number of the interface
in	VmbAccessModeType	interfacePermittedAccess	The access privileges for the interface

The ID of the camera may be, among others, one of the following: "169.254.12.13", "000f31000001", a plain serial number: "1234567890", or the device ID of the underlying transport layer.

ICameraFactory destructor

Destroys an instance of class Camera

Camera

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

The ID of the camera may be, among others, one of the following: "169.254.12.13", "000f31000001", a plain serial number: "1234567890", or the device ID of the underlying transport layer.

Camera destructor

Destroys an instance of class Camera

Destroying a camera implicitly closes it beforehand.

Open()

Opens the specified camera.

	Type	Name	Description
in	VmbAccessMode_t	accessMode	Access mode determines the level 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

A camera may be opened in a specific access mode. This mode determines the level of control you have on a camera.

Close()

Closes the specified camera.

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

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

GetID()

Gets the ID of a camera.

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

- **VmbErrorSuccess:** If no error

GetName()

Gets the name of a camera.

	Type	Name	Description
out	std::string&	name	The name of the camera

- **VmbErrorSuccess:** If no error

GetModel()

Gets the model name of a camera.

	Type	Name	Description
out	std::string&	model	The model name of the camera

- **VmbErrorSuccess:** If no error

GetSerialNumber()

Gets the serial number of a camera.

	Type	Name	Description
out	std::string&	serialNumber	The serial number of the camera

- **VmbErrorSuccess:** If no error

GetInterfaceID()

Gets the interface ID of a camera.

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

- **VmbErrorSuccess:** If no error

GetInterfaceType()

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

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

- **VmbErrorSuccess:** If no error

GetPermittedAccess()

Gets the access modes of a camera.

	Type	Name	Description
out	VmbAccessModeType&	permittedAccess	The possible access modes of the camera

- **VmbErrorSuccess:** If no error

ReadRegisters()

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

	Type	Name	Description
in	const Uint64Vector&	addresses	A list of register addresses
out	Uint64Vector&	buffer	The returned data as vector

- **VmbErrorSuccess:** If all requested registers have been read
- **VmbErrorBadParameter:** Vectors "addresses" and/or "buffer" are empty.
- **VmbErrorIncomplete:** If at least one, but not all registers have been read. See overload ReadRegisters(const Uint64Vector&, Uint64Vector&, VmbUint32_t&).

ReadRegisters()

Same as ReadRegisters(const Uint64Vector&, Uint64Vector&), but returns the number of successful read operations in case of an error.

	Type	Name	Description
in	const Uint64Vector&	addresses	A list of register addresses
out	Uint64Vector&	buffer	The returned data as vector
out	VmbUint32_t&	completedReads	The number of successfully read registers

- **VmbErrorSuccess:** If all requested registers have been read
- **VmbErrorBadParameter:** Vectors "addresses" and/or "buffer" are empty.
- **VmbErrorIncomplete:** If at least one, but not all registers have been read.

WriteRegisters()

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

	Type	Name	Description
in	const Uint64Vector&	addresses	A list of register addresses
in	const Uint64Vector&	buffer	The data to write as vector

- **VmbErrorSuccess:** If all requested registers have been written
- **VmbErrorBadParameter:** Vectors "addresses" and/or "buffer" are empty.
- **VmbErrorIncomplete:** If at least one, but not all registers have been written. See overload WriteRegisters(const Uint64Vector&, const Uint64Vector&, VmbUint32_t&).

WriteRegisters()

Same as WriteRegisters(const Uint64Vector&, const Uint64Vector&), but returns the number of successful write operations in case of an error.

	Type	Name	Description
in	const Uint64Vector&	addresses	A list of register addresses
in	const Uint64Vector&	buffer	The data to write as vector
out	VmbUint32_t&	completedWrites	The number of successfully read registers

- **VmbErrorSuccess:** If all requested registers have been written
- **VmbErrorBadParameter:** Vectors "addresses" and/or "buffer" are empty.
- **VmbErrorIncomplete:** If at least one, but not all registers have been written.

ReadMemory()

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

	Type	Name	Description
in	const VmbUint64_t&	address	The address to read from
out	UcharVector&	buffer	The returned data as vector

- **VmbErrorSuccess:** If all requested bytes have been read
- **VmbErrorBadParameter:** Vector "buffer" is empty.
- **VmbErrorIncomplete:** If at least one, but not all bytes have been read. See overload ReadMemory(const VmbUint64_t&, UcharVector&, VmbUint32_t&).

ReadMemory()

Same as ReadMemory(const Uint64Vector&, UcharVector&), but returns the number of bytes successfully read in case of an error VmbErrorIncomplete.

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

- **VmbErrorSuccess:** If all requested bytes have been read
- **VmbErrorBadParameter:** Vector "buffer" is empty.
- **VmbErrorIncomplete:** If at least one, but not all bytes have been read.

WriteMemory()

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

	Type	Name	Description
in	const VmbUint64_t&	address	The address to write to
in	const UcharVector&	buffer	The data to write as vector

- **VmbErrorSuccess:** If all requested bytes have been written
- **VmbErrorBadParameter:** Vector "buffer" is empty.
- **VmbErrorIncomplete:** If at least one, but not all bytes have been written. See overload WriteMemory(const VmbUint64_t&, const UcharVector&, VmbUint32_t&).

WriteMemory()

Same as WriteMemory(const Uint64Vector&, const UcharVector&), but returns the number of bytes successfully written in case of an error VmbErrorIncomplete.

	Type	Name	Description
in	const VmbUint64_t&	address	The address to write to
in	const UcharVector&	buffer	The data to write as vector
out	VmbUint32_t&	sizeComplete	The number of successfully written bytes

- **VmbErrorSuccess:** If all requested bytes have been written
- **VmbErrorBadParameter:** Vector "buffer" is empty.
- **VmbErrorIncomplete:** If at least one, but not all bytes have been written.

AcquireSingleImage()

Gets one image synchronously.

	Type	Name	Description
out	FramePtr&	pFrame	The frame that gets filled
in	VmbUint32_t	timeout	The time to wait until the frame got filled
in	FrameAllocationMode	allocationMode	The frame allocation mode

- **VmbErrorSuccess:** If no error
- **VmbErrorBadParameter:** "pFrame" is NULL.
- **VmbErrorTimeout:** Call timed out

AcquireMultipleImages()

Gets a certain number of images synchronously.

	Type	Name	Description
out	FramePtrVector&	frames	The frames that get filled
in	VmbUint32_t	timeout	The time to wait until one frame got filled
in	FrameAllocationMode	allocationMode	The frame allocation mode

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

- **VmbErrorSuccess:** If no error
- **VmbErrorInternalFault:** Filling all the frames was not successful.
- **VmbErrorBadParameter:** Vector "frames" is empty.

AcquireMultipleImages()

Same as AcquireMultipleImages(FramePtrVector&, VmbUint32_t), but returns the number of frames that were filled completely.

	Type	Name	Description
out	FramePtrVector&	frames	The frames that get filled
in	VmbUint32_t	timeout	The time to wait until one frame got filled
out	VmbUint32_t&	numFramesCompleted	The number of frames that were filled completely
in	FrameAllocationMode	allocationMode	The frame allocation mode

The size of the frame vector determines the number of frames to use. On return, "numFramesCompleted" holds the number of frames actually filled.

- **VmbErrorSuccess:** If no error
- **VmbErrorBadParameter:** Vector "frames" is empty.

StartContinuousImageAcquisition()

Starts streaming and allocates the needed frames

	Type	Name	Description
in	int	bufferCount	The number of frames to use
out	const IFrameObserverPtr&	pObserver	The observer to use on arrival of new frames
in	FrameAllocationMode	allocationMode	The frame allocation mode

- **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

StopContinuousImageAcquisition()

Stops streaming and deallocates the needed frames

AnnounceFrame()

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

	Type	Name	Description
in	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
- **VmbErrorBadParameter:** "pFrame" is NULL.
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API

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.

RevokeFrame()

Revoke a frame from the API.

	Type	Name	Description
in	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
- **VmbErrorBadParameter:** "pFrame" is NULL.
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API

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

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

QueueFrame()

Queues a frame that may be filled during frame capturing.

	Type	Name	Description
in	const FramePtr&	pFrame	A shared pointer to a frame

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given frame is not valid
- **VmbErrorBadParameter:** "pFrame" is NULL.
- **VmbErrorStructSize:** The given struct size is not valid for this version of the API
- **VmbErrorInvalidCall:** StopContinuousImageAcquisition is currently running in another thread

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 ensure that the frame is also revoked by calling RevokeFrame() or RevokeAll() when cleaning up.

FlushQueue()

Flushes the capture queue.

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

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.

StartCapture()

Prepare the APIor 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

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

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

SaveCameraSettings()

Saves the current camera setup to an XML file

	Type	Name	Description
in	std::string	pStrFileName	xml file name
in	VmbFeaturePersistSettings_t*	pSettings	pointer to settings struct

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInternalFault:** When something unexpected happens in VimbaC function
- **VmbErrorOther:** Every other failure in load/save settings implementation class

LoadCameraSettings()

Loads the current camera setup from an XML file into the camera

	Type	Name	Description
in	std::string	pStrFileName	xml file name
in	VmbFeaturePersistSettings_t*	pSettings	pointer to settings struct

- **VmbErrorSuccess:** If no error
- **VmbErrorApiNotStarted:** VmbStartup() was not called before the current command
- **VmbErrorBadHandle:** The given handle is not valid
- **VmbErrorInternalFault:** When something unexpected happens in VimbaC function
- **VmbErrorOther:** Every other failure in load/save settings implementation class

LoadSaveSettingsSetup()

Sets Load/Save settings behaviour (alternative to settings struct)

	Type	Name	Description
in	VmbFeaturePersist_t	persistType	determines which feature shall be considered during load/save settings
in	VmbUint32_t	maxIterations	determines how many 'tries' during loading feature values shall be performed
in	VmbUint32_t	loggingLevel	determines level of detail for load/save settings logging

Frame

Frame constructor

Creates an instance of class Frame of a certain size

	Type	Name	Description
in	VmbInt64_t	bufferSize	The size of the underlying buffer
in	FrameAllocationMode	allocationMode	Indicates if announce frame or alloc and announce frame is used

Frame constructor

Creates an instance of class Frame with the given user buffer of the given size

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

Frame destructor

Destroys an instance of class Frame

RegisterObserver()

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

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

- **VmbErrorSuccess:** If no error
- **VmbErrorBadParameter:** "pObserver" is NULL.

- **VmbErrorResources:** The observer was in use

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

UnregisterObserver()

Unregisters the observer that was called whenever a new frame arrived

GetAncillaryData()

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

	Type	Name	Description
out	AncillaryDataPtr&	pAncillaryData	The wrapped chunk data

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

GetAncillaryData()

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

	Type	Name	Description
out	ConstAncillaryDataPtr&	pAncillaryData	The wrapped chunk data

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

GetBuffer()

Returns the complete buffer including image and chunk data

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

- **VmbErrorSuccess:** If no error

GetBuffer()

Returns the complete buffer including image and chunk data

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

- **VmbErrorSuccess:** If no error

GetImage()

Returns only the image data

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

- **VmbErrorSuccess:** If no error

GetImage()

Returns only the image data

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

- **VmbErrorSuccess:** If no error

GetReceiveStatus()

Returns the receive status of a frame

	Type	Name	Description
out	VmbFrameStatusType&	status	The receive status

- **VmbErrorSuccess:** If no error

GetImageSize()

Returns the memory size of the image

	Type	Name	Description
out	VmbUint32_t&	imageSize	The size in bytes

- **VmbErrorSuccess:** If no error

GetAncillarySize()

Returns memory size of the chunk data

	Type	Name	Description
out	VmbUint32_t&	ancillarySize	The size in bytes

- **VmbErrorSuccess:** If no error

GetBufferSize()

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

	Type	Name	Description
out	VmbUint32_t&	bufferSize	The size in bytes

- **VmbErrorSuccess:** If no error

GetPixelFormat()

Returns the GenICam pixel format

	Type	Name	Description
out	VmbPixelFormatType&	pixelFormat	The GenICam pixel format

- **VmbErrorSuccess:** If no error

GetWidth()

Returns the width of the image

	Type	Name	Description
out	VmbUint32_t&	width	The width in pixels

- **VmbErrorSuccess:** If no error

GetHeight()

Returns the height of the image

	Type	Name	Description
out	VmbUint32_t&	height	The height in pixels

- **VmbErrorSuccess:** If no error

GetOffsetX()

Returns the x offset of the image

	Type	Name	Description
out	VmbUint32_t&	offsetX	The x offset in pixels

- **VmbErrorSuccess:** If no error

GetOffsetY()

Returns the y offset of the image

	Type	Name	Description
out	VmbUint32_t&	offsetY	The y offset in pixels

- **VmbErrorSuccess:** If no error

GetFrameID()

Returns the frame ID

	Type	Name	Description
out	VmbUint64_t&	frameID	The frame ID

- **VmbErrorSuccess:** If no error

GetTimeStamp()

Returns the time stamp

	Type	Name	Description
out	VmbUint64_t&	timestamp	The time stamp

- **VmbErrorSuccess:** If no error

Feature

GetValue()

Queries the value of a feature of type VmbInt64

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

GetValue()

Queries the value of a feature of type double

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

GetValue()

Queries the value of a feature of type string

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

When an empty string is returned, its size indicates the maximum length

GetValue()

Queries the value of a feature of type bool

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

GetValue()

Queries the value of a feature of type UcharVector

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

GetValue()

Queries the value of a feature of type const UcharVector

	Type	Name	Description
out	UcharVector&	value	The feature's value
out	VmbUint32_t&	sizeFilled	The number of actually received values

GetValues()

Queries the values of a feature of type Int64Vector

	Type	Name	Description
out	Int64Vector&	values	The feature's values

GetValues()

Queries the values of a feature of type StringVector

	Type	Name	Description
out	StringVector&	values	The feature's values

GetEntry()

Queries a single enum entry of a feature of type Enumeration

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

GetEntries()

Queries all enum entries of a feature of type Enumeration

	Type	Name	Description
out	EnumEntryVector&	entries	An enum feature's enum entries

GetRange()

Queries the range of a feature of type double

	Type	Name	Description
out	double&	minimum	The feature's min value
out	double&	maximum	The feature's max value

GetRange()

Queries the range of a feature of type VmbInt64

	Type	Name	Description
out	VmbInt64_t&	minimum	The feature's min value
out	VmbInt64_t&	maximum	The feature's max value

SetValue()

Sets the value of a feature of type VmbInt32

	Type		Name	Description
in	const	VmbInt32_t&	value	The feature's value

SetValue()

Sets the value of a feature of type VmbInt64

	Type		Name	Description
in	const	VmbInt64_t&	value	The feature's value

SetValue()

Sets the value of a feature of type double

	Type		Name	Description
in	const	double&	value	The feature's value

SetValue()

Sets the value of a feature of type char*

	Type		Name	Description
in	const	char*	pValue	The feature's value

SetValue()

Sets the value of a feature of type bool

	Type	Name	Description
in	bool	value	The feature's value

SetValue()

Sets the value of a feature of type UcharVector

	Type	Name	Description
in	const UcharVector&	value	The feature's value

HasIncrement()

Gets the support state increment of a feature

	Type	Name	Description
out	VmbBool_t&	incrementsupported	The feature's increment support state

GetIncrement()

Gets the increment of a feature of type VmbInt64

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

GetIncrement()

Gets the increment of a feature of type double

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

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
in	const char*	pValue	The enumeration value as string
out	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

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
in	const VmbInt64_t	value	The enumeration value as int
out	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

RunCommand()

Executes a feature of type Command

IsCommandDone()

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

	Type	Name	Description
out	bool&	isDone	True when execution has finished

GetName()

Queries a feature's name

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

GetDisplayName()

Queries a feature's display name

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

GetDataType()

Queries a feature's type

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

GetFlags()

Queries a feature's access status

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

GetCategory()

Queries a feature's category in the feature tress

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

GetPollingTime()

Queries a feature's polling time

	Type	Name	Description
out	VmbUint32_t&	pollingTime	The interval to poll the feature

GetUnit()

Queries a feature's unit

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

GetRepresentation()

Queries a feature's representation

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

GetVisibility()

Queries a feature's visibility

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

GetToolTip()

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

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

GetDescription()

Queries a feature's description

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

GetSFNCNamespace()

Queries a feature's Standard Feature Naming Convention namespace

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

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

GetSelectedFeatures()

Gets the features that get selected by the current feature

	Type	Name	Description
out	FeaturePtrVector&	selectedFeatures	The selected features

IsReadable()

Queries the read access status of a feature

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

IsWritable()

Queries the write access status of a feature

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

IsStreamable()

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

	Type	Name	Description
out	bool&	isStreamable	True when streamable

RegisterObserver()

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

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

- **VmbErrorSuccess:** If no error
- **VmbErrorBadParameter:** "pObserver" is NULL.

UnregisterObserver()

Unregisters an observer

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

- **VmbErrorSuccess:** If no error
- **VmbErrorBadParameter:** "pObserver" is NULL.

EnumEntry

EnumEntry constructor

Creates an instance of class EnumEntry

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

EnumEntry constructor

Creates an instance of class EnumEntry

EnumEntry copy constructor

Creates a copy of class EnumEntry

EnumEntry assignment operator

assigns EnumEntry to existing instance

EnumEntry destructor

Destroys an instance of class EnumEntry

GetName()

Gets the name of an enumeration

	Type	Name	Description
out	std::string&	name	The name of the enumeration

GetDisplayName()

Gets a more declarative name of an enumeration

	Type	Name	Description
out	std::string&	displayName	The display name of the enumeration

GetDescription()

Gets the description of an enumeration

	Type	Name	Description
out	std::string&	description	The description of the enumeration

GetTooltip()

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

	Type	Name	Description
out	std::string&	tooltip	The tooltip as string

GetValue()

Gets the integer value of an enumeration

	Type	Name	Description
out	VmbInt64_t&	value	The integer value of the enumeration

GetVisibility()

Gets the visibility of an enumeration

	Type	Name	Description
out	VmbFeatureVisibilityType&	value	The visibility of the enumeration

GetSNFCNamespace()

Gets the standard feature naming convention namespace of the enumeration

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

AncillaryData

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

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

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

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

GetBuffer()

Returns the underlying buffer

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

- **VmbErrorSuccess:** If no error

GetBuffer()

Returns the underlying buffer

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

- **VmbErrorSuccess:** If no error

GetSize()

Returns the size of the underlying buffer

	Type	Name	Description
out	VmbUint32_t&	size	The size of the buffer

- **VmbErrorSuccess:** If no error