Allied Vision

Vimba

Vimba .NET Manual

1.8.3

Legal Notice

Trademarks

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

Warranty

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

Copyright

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

Contents

List of Tables

Listings

1 Contacting Allied Vision

Contact information on our website

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

Find an Allied Vision office or distributor

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

Email

info@alliedvision.com
support@alliedvision.com

Sales Offices

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

Headquarters

Allied Vision Technologies GmbH
Taschenweg 2a
07646 Stadtroda
Germany

Tel: +49 (0)36428 677-0
Fax: +49 (0)36428 677-28
Managing Directors (Geschäftsführer): Andreas Gerk, Peter Tix

2 Document history and conventions

This chapter includes:

2.1 Document history

Version	Date	Changes
1.0	2012-09-27	Initial version
1.1	2013-03-05	Added which methods can be called within a callback, fixed wording and formatting issues
1.2	2013-06-18	Small corrections, layout changes
1.2.1	2013-08-25	Corrected error code chapter, small changes
1.3	2014-08-11	Appended the function reference, re-structured and made corrections
1.4	2015-11-09	Added USB compatibility, renamed several Vimba components and documents ("AVT" no longer in use), links to new Allied Vision website
1.5	2016-01-31	Added Goldeye CL compatibility, new document layout
1.6	2017-04-05	Addded chapter Triggering cameras (including Action Commands), changed the position of `Camera.FlushQueue()`, several minor changes, updated document layout
1.7	September 2017	Added information to chapter Trigger over Ethernet – Action Commands, updated Troubleshooting, section Goldeye CL cameras
1.7.1	May 2018	Bug fixes
1.8.0	June 2019	Bug fixes Bug fixes, added a note for code examples
1.8.1	October 2019	Updated for use with GenTL 1.5, bug fixes
1.8.2	May 2020	Bug fixes
1.8.3	October 2020	Added standard-compliant ForceIP features

2.2 Conventions used in this manual

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

2.2.1 Styles

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

2.2.2 Symbols

Practical Tip

Safety-related instructions to avoid malfunctions
Instructions to avoid malfunctions

Further information available online

3 General aspects of the API

The Vimba .NET API is an object-oriented API for .NET languages that enables programmers to interact with Allied Vision cameras independent of the interface technology (Gigabit Ethernet, USB, 1394). It utilizes GenICam transport layer modules to connect to the various camera interfaces and is therefore generic in terms of camera interfaces.

The entry point to Vimba API is the `Vimba` class. The `Vimba` class allows to control the API's behavior and to query for interfaces and cameras.

The Vimba Manual contains a description of the API concepts.

By default, code examples are located at:
C:\Users\Public\Documents\Allied Vision\Vimba_x.x

To build the .NET code examples with Visual Studio 2019 and Windows 10:
In Visual Studio, open the properties of a solution and go to Application. In the Target Framework drop-down list, select an installed .NET framework version, for example, 4.7.2.

4 API usage

This chapter includes:

The entry point to the Vimba .NET API is the `Vimba` object. Use `new Vimba()` to obtain a .NET reference to it. All Vimba .NET classes reside in the namespace `AVT.VmbAPINET`, so you might want to employ the using declaration `using AVT.VmbAPINET`.

4.1 API Setup

Vimba .NET API is a .NET 2.0 assembly. Hence, a C# application using the Vimba .NET API should ideally be built with the .NET 2.0 framework. If a newer .NET framework version is used, add the following text to the app.config file (example given for .NET framework 4.5):

```
<configuration>
    <startup useLegacyV2RuntimeActivationPolicy="true">
        <supportedRuntime version="v4.5" sku=".NETFramework, Version=v4.5" />
    </startup>
</configuration>
```

4.2 API Version

Even if new features are introduced to Vimba .NET API, your software remains backwards compatible. Use the property `Vimba.Version` to check the version number of Vimba .NET API.

4.3 API Startup and Shutdown

In order to start and shut down Vimba .NET API, use these paired methods:

- `Vimba.Startup()` initializes Vimba API.
- `Vimba.Shutdown()` shuts down Vimba API (as soon as all observers are finished).

`Vimba.Startup()` and `Vimba.Shutdown()` must always be paired. Calling the pair several times within the same program is possible, but not recommended. In order to free resources, shut down Vimba API when your application doesn't use it anymore. This will close all opened cameras.

Successive calls of `Vimba.Startup()` or `Vimba.Shutdown()` are ignored, and the first `Vimba.Shutdown()` after a `Vimba.Startup()` will close the API.

4.4 Object Lifetime

All Vimba .NET objects are connected to a corresponding internal object through a shared pointer. That means a Vimba .NET object will not be destroyed until the last reference within the user application and Vimba is dropped.

The best example for this is a `Camera` object, which is created during Vimba startup (if the camera is already plugged in) or during runtime (when plugged in later). The `Camera` object will not be destroyed by Vimba until it is physically unplugged or Vimba is shut down. Consequently, for a camera that stays connected, the returned `Camera` object reference will always stay the same.

Another example is the `Frame` object created by the user application. This object will live as long as Vimba works with it and will be destroyed when Vimba and the user application hold no reference to it anymore.

4.5 Listing available cameras

For a quick start, see ListCameras example of the Vimba SDK.

`Vimba.Cameras` enumerates all cameras recognized by the underlying transport layers. With this property, the programmer can fetch the list of all connected cameras. Before opening any camera, the camera object contains all static details of a physical camera that do not change throughout the object's lifetime such as:

- Camera ID
- Camera model
- Name or ID of the connected interface (for example, the network or 1394 adapter)

The order in which the detected cameras are listed is determined by the order of camera discovery and therefore not deterministic. Normally, Vimba recognizes cameras in the following order: USB - 1394 - GigE - Camera Link. However, this order may change depending on your system configuration and the accessories (for example, hubs or long cables).

GigE cameras:
For GigE cameras, discovery has to be initiated by the host software. This is done automatically at `Vimba.Startup()` in the Vimba class. Any access to the `Vimba.Cameras` property or calls to `Vimba.GetCameraByID()` return immediately.

USB and 1394 cameras:
Changes to the plugged cameras are detected automatically. Consequently, any changes to the camera list are announced via discovery event.

Goldeye CL cameras:
The Camera Link specification does not support plug & play or discovery events. To detect changes to the camera list, call `Vimba.Shutdown()` and `Vimba.Startup()` consecutively.

See Listing 1 for an example of **getting the camera list**.

Listing 1: List Cameras

```
string strName;
Vimba sys = new Vimba();
CameraCollection cameras = null;

try
{
  sys.Startup();
  cameras = sys.Cameras;

  Console.WriteLine( "Cameras found: " + cameras.Count );
  Console.WriteLine();

  foreach (Camera camera in cameras)
  {
    try
    {
      strName = camera.Name;
    }
    catch ( VimbaException ve )
    {
      strName = ve.Message;
    }
    Console.WriteLine( "Camera Name: " + strName );
  }
}
finally
{
  sys.Shutdown();
}
```

The `Camera` class provides the properties listed in Table 4.5 to obtain information about a camera.

Type	Name	Purpose
`string`	`Id`	The unique ID
`string`	`Name`	The name
`string`	`Model`	The model name
`string`	`SerialNumber`	The serial number
`VmbAccessModeType`	`PermittedAccess`	The mode to open the camera
`string`	`InterfaceID`	The ID of the interface the camera is connected to

Table 1: Basic properties of the Camera class

Notifications of changed camera states

To **get notified whenever a camera is detected, disconnected, or changes its open state**, define and

add a delegate for `CameraListChanged` events (see chapter Using Events). The delegate has to be of type `OnCameraListChangeHandler()`.

`Vimba.Shutdown()` blocks until all events have finished execution.

Calling certain methods within the event routine would counteract the reason of the event or lead to a deadlock. Therefore these methods must **not** be called within the event routine:

- `Feature.xxxValue.set()`
- `Feature.RunCommand()`
- `Camera.Open()`
- `Camera.Close()`
- `Vimba.Startup()`
- `Vimba.Shutdown()`
- `Vimba.Cameras.get()`
- `Vimba.OpenCameraByID()`

4.6 Opening and closing a camera

A camera must be opened to control it and to capture images.

Call `Camera.Open()` on the camera list entry of your choice, or call `Vimba.OpenCameraByID()` with the ID of the camera. In both cases, also provide the desired access mode for the camera.
Vimba API provides several **access modes**:

- `VmbAccessModeFull` - read and write access. Use this mode to configure the camera features and to acquire images (Goldeye CL cameras: configuration only)
- `VmbAccessModeRead` – read-only access. Setting features is not possible. However, for GigE cameras that are already in use by another application, the acquired images can be transferred to Vimba API (Multicast).
- `VmbAccessModeConfig` – GigE only, enables configuring the IP address of the camera

An example for **opening and closing a camera** retrieved from the camera list is shown in Listing 2.

Listing 2: Open and Close Camera

```
Vimba sys = new Vimba();
CameraCollection cameras = null;

try
{
  sys.Startup();
  cameras = sys.Cameras;

  foreach (Camera camera in cameras)
  {
    try
    {
      camera.Open( VmbAccessModeType.VmbAccessModeFull );
      Console.WriteLine( "Camera opened" );
      camera.Close();
    }
    catch ( VimbaException ve )
    {
      Console.WriteLine(  "Error : " + ve.MapReturnCodeToString() );
    }
  }
}
finally
{
  sys.Shutdown();
}
```

Listing 3 shows how to **open a GigE camera by its IP address**.

Listing 3: Open Camera by IP

```
Vimba sys = new Vimba();
Camera camera=null;

try
{
  sys.Startup();

  try
  {
    camera = sys.OpenCameraByID( "192.168.0.42",
                                 VmbAccessModeType.VmbAccessModeFull );
    Console.WriteLine( "Camera opened" );
  }
  catch ( VimbaException ve )
  {
    Console.WriteLine( "Camera open error : " +
                       ve.MapReturnCodeToString() );
  }
}
finally
{
  sys.Shutdown();
}
```

Instead of the IP address, the ID of the camera provided by the `Camera.Id` property can be used to open the camera.

4.7 Accessing Features

For a quick start, see ListFeatures example of the Vimba SDK.

GenICam-compliant features control and monitor various aspects of the drivers and cameras. For more details on features, see (if installed):

- GigE Features Reference (GigE camera features)
- USB Features Reference (USB camera features)
- Vimba 1394 TL Features Manual (1394 camera and TL features)
- Goldeye G/CL Features Reference
  https://www.alliedvision.com/en/support/technical-documentation.html
- Vimba Manual (Vimba System features)

There are several feature types with type-specific properties and type-specific functionality. Vimba API provides its own set of access methods and properties for each of these feature types.

Table 4.7 lists Vimba methods and properties of the `Feature` class used to access feature values.

Feature Type	Get / Set	Range	Other
`Enum`	`EnumValue`	`EnumRange`	`IsEnumValueAvailable`
			`EnumEntries`
	`EnumIntValue`		`EnumValues`
			`EnumIntValues`
`long`	`IntValue`	`IntRangeMin`	`IntIncrement`
		`IntRangeMax`	
`double`	`FloatValue`	`FloatRangeMin`	
		`FloatRangeMax`	
`string`	`StringValue`		
`bool`	`BoolValue`		
`Command`	`RunCommand`		
	`IsCommandDone`		
`Raw`	`RawValue`		

Table 2: Properties and methods for accessing a `Feature`

With the properties `XXXValue`, a feature's value can be accessed.

Integer and double features support read-only properties `XXXRangeMin` and `XXXRangeMax`. They return the minimum and maximum value that a feature can have. Integer features also support the `IntIncrement` property to query the step size of feature changes. In some cases, e.g. the actual maximum value cannot be set because it is not a multiple of the increment added to the minimum. In general, valid values for integer features are `min <= val <= min + [(max-min)/increment] * increment`.

Enumeration features support properties `EnumValues` and `EnumIntValues` that return an array of strings or integers. The values returned in these arrays can be used to set the feature according to the result of `IsEnumValueAvailable`. An enumeration feature can also be used in a similar way as an integer feature.

Since not all the features are available all the time, the current accessibility of features may be queried via methods `IsReadable()` and `IsWritable()`. The availability of Enum values may be queried with method `IsEnumValueAvailable()`.

With property `Camera.Features`, you can list all features available for a camera. To query for the presence of a specific feature, use the `ContainsName()` method of the returned feature collection. This list remains static while the camera is opened. The `Feature` class of the entries in this list also provides information about the features that always stay the same for this camera. Use the following properties of class `Feature` to access them:

For an example of **reading a camera feature**, see Listing 4.

Listing 4: Reading a camera feature

```
FeatureCollection features=null;
Feature feature=null;
long width;

try
{
  features = camera.Features;
  if ( features.ContainsName( "Width" )
  {
    feature = features[ "Width" ];
    width = feature.IntValue;
    Console.WriteLine( "Width = " + width.ToString() );
  }
}
catch ( VimbaException ve )
{
  Console.WriteLine( "Feature error: " + ve.Message );
}
```

As an example for **writing features to a camera** and **running a command feature**, see Listing 5.

[1]For the current availability, query methods `IsReadable()` and `IsWritable()`. The availability of Enum values may be queried with method `IsEnumValueAvailable()`.

Property	Purpose
`Name`	Name of the feature
`DisplayName`	Name to display in GUI
`DataType`	Data type of the feature. Gives information about the available methods for the feature. See table 4.7
`Flags`	Static feature flags, containing information about the actions available for a feature and how changes might affect it. `Read`[1] and `Write`[1] flags determine whether get and set methods succeed. `Volatile` features may change with every successive read. When writing `ModifyWrite` features, they will be adjusted to valid values.
`Category`	Category the feature belongs to, used for structuring the features
`PollingTime`	The suggested time to poll the feature
`Unit`	The unit of the feature, if available
`Representation`	The scale to represent the feature, used as a hint for feature control
`Visibility`	The audience the feature is for
`ToolTip`	Short description of the feature, used for bubble help
`Description`	Description of the feature, used as extended explanation
`SFNCNamespace`	The SFNC namespace of the feature
`AffectedFeatures`	Features that change if the feature is changed
`SelectedFeatures`	Features that are selected by the feature

Table 3: Static properties of a `Feature`

Listing 5: Writing features and running command features

```
FeatureCollection features=null;
Feature feature=null;

try
{
  features = camera.Features;
  feature = features[ "AcquisitionMode" ];
  feature.EnumValue = "Continuous";
  feature = features[ "AcquisitionStart" ];
  feature.RunCommand();
  Console.WriteLine( "Acquisition started" );
}
catch ( VimbaException ve )
{
  Console.WriteLine( "Feature error: " + ve.Message );
}
```

Table 4 introduces the basic features of all cameras. A feature has a name, a type, and access flags such as `read-permitted` and `write-permitted`.

Feature	Type	Access Flags	Description
`AcquisitionMode`	`Enumeration`	R/W	The acquisition mode of the camera. Value set: Continuous, SingleFrame, MultiFrame.
`AcquisitionStart`	`Command`		Start acquiring images.
`AcquisitionStop`	`Command`		Stop acquiring images.
`PixelFormat`	`Enumeration`	R/W	The image format. Possible values are e.g.: Mono8, RGB8Packed, YUV411Packed, BayerRG8, …
`Width`	`Uint32`	R/W	Image width, in pixels.
`Height`	`Uint32`	R/W	Image height, in pixels.
`PayloadSize`	`Uint32`	R	Number of bytes in the camera payload, including the image.

Table 4: Basic features found on all cameras

To **get notified when a feature's value changes**, use `Feature.OnFeatureChangeHandler()` (see chapter Using Events). In the implementation of this delegate, it is possible to react on updated feature values as it will get called by Vimba .NET API on the according event.

`Vimba.Shutdown()` blocks until all events have finished execution.

Calling certain methods within the event routine would counteract the reason of the event or lead to a deadlock. Therefore these methods must **not** be called within the event routine:

- `Feature.xxxValue.set()`
- `Feature.RunCommand()`
- `Camera.Open()`
- `Camera.Close()`
- `Vimba.Startup()`
- `Vimba.Shutdown()`
- `Vimba.Cameras.get()`
- `Vimba.OpenCameraByID()`

4.8 Image Capture (API) and Acquisition (Camera)

The Vimba Manual describes the principles of synchronous and asynchronous image acquisition.

For a quick start, see SynchronousGrab or AsynchronousGrab examples of the Vimba SDK.

4.8.1 Image Capture and Image Acquisition

Image capture and image acquisition are two independent operations: **Vimba API captures** images, the **camera acquires** images.

To obtain an image from your camera, setup Vimba API to capture images before starting the acquisition on the camera:

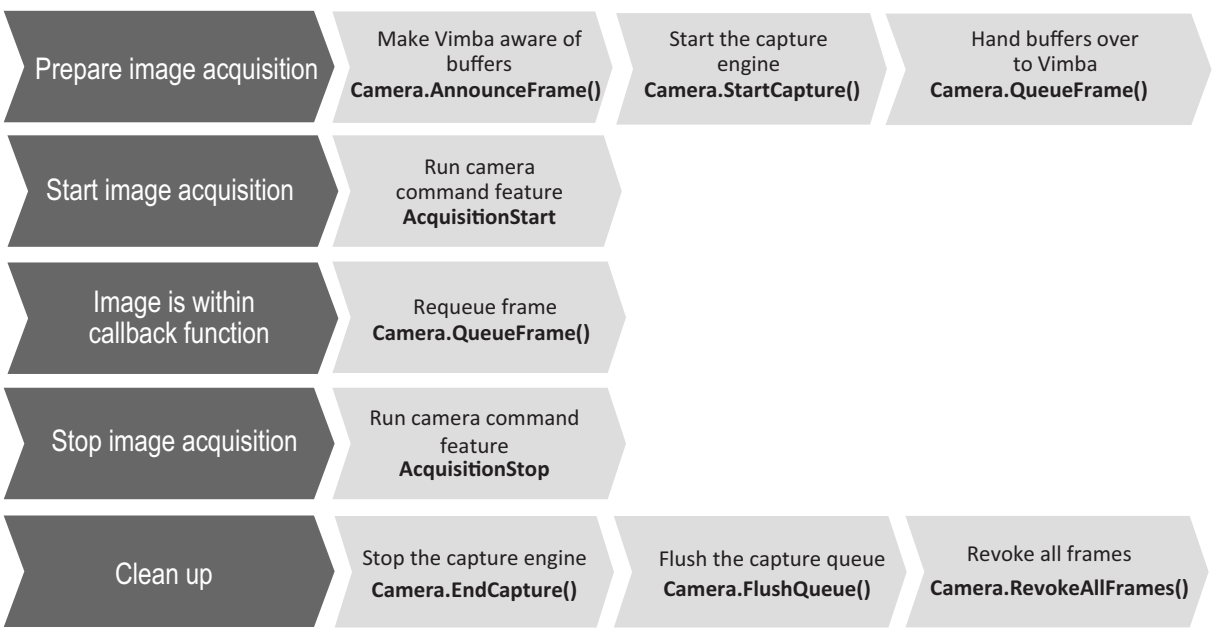

Figure 1: Typical asynchronous application using Vimba.NET

Note that the .NET API provides convenience methods for standard applications. These methods perform several procedures in just one step. However, for complex applications with special requirements, manual programming as described here is still required.

4.8.2 Image Capture

To enable image capture, frame buffers must be allocated, and the API must be prepared for incoming frames.

This is done in convenience method `Camera.StartContinuousAcquisition()` (`Camera.StopContinuousAcquisition()` stops acquisition.).

Please find below how to **asynchronously capture images** step by step:

1. Open the camera as described in chapter Opening and closing a camera.
2. Query the necessary buffer size through the feature *PayloadSize* (A)[1]. Allocate frame buffers of this size (B).
3. Announce the frame buffers (1). Depending on the underlying technology, this performs e.g. DMA preparation.
4. Start the capture engine (2). This sets the host ready to receive incoming frames.
5. Queue the frame (3) you have just created with `Camera.QueueFrame()`, so that the buffer can be filled when the acquisition has started.
   The API is now ready. Start and stop image acquisition on the camera as described in chapter Image Acquisition.
6. Register a delegate (C) that gets executed when capturing is complete.
   The delegate has to be of type `Camera.OnFrameReceivedHandler`. During event handling, queue the frame again after you have processed it. Reduce processing to a minimum (e.g. copying) within the event handler and have another thread perform expensive image processing.
7. Stop the capture engine with `Camera.EndCapture()`.
8. Call `Camera.FlushQueue()` to cancel all frames on the queue. In case the API has done memory allocation, this memory is not released until the camera class' `RevokeAllFrames()` / `RevokeFrame()`, `EndCapture()` or `Close()` method has been called.
9. Revoke the frames with `Camera.RevokeAllFrames()` to clear the buffers.

To **synchronously capture images** (blocking your execution thread), follow these steps:

1. Open the camera as described in chapter Opening and closing a camera.
2. How you proceed depends on the number of frames you need:
   - **A single frame:** Use `Camera.AcquireSingleImage()` to receive a frame.
   - **Multiple frames:** Use `Camera.AcquireMultipleImages()` to receive several frames (determined by the size of your vector of `FramePtrs`).

[1]The bracketed tokens in this chapter refer to Listing 6.

To assure correct continuous image capture, use at least two or three frames. The appropriate number of frames to be queued in your application depends on the frames per second the camera delivers and on the speed in which you are able to re-queue frames (also taking into consideration the operating system load). If your application cannot hand back (re-queue) the received and processed frames to the API fast enough, the API will run out of buffers. Incoming images from the camera will then be dropped. The image frames are filled in the same order in which they were queued.

Always check that `Frame.ReceiveStatus` returns `VmbFrameStatusComplete` when a frame is returned to ensure the data is valid.

Calling certain methods within the event routine would counteract the reason of the event or lead to a deadlock. Therefore these methods must **not** be called within the frame received event routine:

- `Camera.Open()`
- `Camera.Close()`
- `Camera.AcquireSingleImage()`
- `Camera.AcquireMultipleImages()`
- `Camera.StartContinuousImageAcquisition()`
- `Camera.StopContinuousImageAcquisition()`
- `Camera.StartCapture()`
- `Camera.EndCapture()`
- `Camera.AnnounceFrame()`
- `Camera.RevokeFrame()`
- `Camera.RevokeAllFrames()`
- `Vimba.Startup()`
- `Vimba.Shutdown()`
- `Vimba.OpenCameraByID()`

4.8.3 Image Acquisition

If you use one of the methods `AcquireSingleImage()`, `AcquireMultipleImages()`, or `StartContinuousImageAcquisition()` of the `Camera` class, no further actions have to be taken.

Only if you have setup capture step by step as described in chapter Image Capture, you have to start image acquisition on your camera:

1. Set the feature *AcquisitionMode* (e.g. to *Continuous*).
2. Run the command *AcquisitionStart* (4).

To stop image acquisition, run command *AcquisitionStop*.

Listing 6 shows a **simplified streaming example** (without error handling). For a more comprehensive example, see the AsynchronousGrab example application of the Vimba SDK.

Listing 6: Streaming

```
Camera m_Camera=null;

private void StartCamera()
{
  Vimba sys = new Vimba();
  CameraCollection cameras=null;
  FeatureCollection features=null;
  Feature feature=null;
  long payloadSize;
  Frame[] frameArray = new Frame[3];

  sys.Startup();

  cameras = sys.Cameras;

  m_Camera = cameras[0];
  m_Camera.Open(VmbAccessModeType.VmbAccessModeFull );

  m_Camera.OnFrameReceived +=
          new Camera.OnFrameReceivedHandler(OnFrameReceived);      (C)

  features = m_Camera.Features;
  feature = features[ "PayloadSize" ];                             (A)
  payloadSize = feature.IntValue;

  for ( int index=0; index<frameArray.Length; ++index )
  {
    frameArray[index] = new Frame( payloadSize );                  (B)
    m_Camera.AnnounceFrame( frameArray[index] );                   (1)
  }

  m_Camera.StartCapture();                                         (2)

  for ( int index=0; index<frameArray.Length; ++index )
  {
    m_Camera.QueueFrame( frameArray[index] );                      (3)
  }
```

```
  feature = features[ "AcquisitionMode" ];                          (4)
  feature.EnumValue = "Continuous";

  feature = features[ "AcquisitionStart" ];                         (5)
  feature.RunCommand();
}

private void StopCamera()
{
  FeatureCollection features = m_Camera.Features;
  Feature feature = features[ "AcquisitionStop" ];
  feature.RunCommand();

  m_Camera.EndCapture();
  m_Camera.FlushQueue();
  m_Camera.RevokeAllFrames();
  m_Camera.Close();
}

private void OnFrameReceived(Frame frame)                             (C)
{
  if (InvokeRequired) // if not from this thread invoke it in our context
  {
    // In case of a separate thread (e.g. GUI ) use BeginInvoke to avoid a deadlock
    Invoke(new Camera.OnFrameReceivedHandler(OnFrameReceived), frame);
  }

  if (VmbFrameStatusType.VmbFrameStatusComplete == frame.ReceiveStatus)
  {
    Console.WriteLine( "Frame status complete" );
  }

  m_Camera.QueueFrame(frame);
}
```

While this listing demonstrates a general behavior, calls to time-consuming methods like `Console.WriteLine()` are not recommended within the frame event handling routine. Spending much time inside this routine might reduce the frame rate.

4.9 Converting Frames into Bitmaps

After acquiring a `Frame` object, you can convert the contained image buffer into a `Bitmap` object, either to display it or to perform further transformations. This can easily be done by providing a `Bitmap` object of the correct size to the `Frame.Fill()` method.

See Listing 7 for a simple example of **converting a frame to an Rgb8 bitmap**. For saving a frame as bitmap to disk, see the SynchronousGrab example application of the Vimba SDK.

Listing 7: Converting an acquired frame to an Rgb8 bitmap

```
// Convert frame data into a Bitmap
Bitmap bitmap = null;

bitmap = new Bitmap( (int)myframe.Width, (int)myframe.Height,
                     PixelFormat.Format24bppRgb);

myframe.Fill(ref bitmap);
```

Since the `Bitmap` class of .NET 2.0 supports different image formats than the Vimba ImageTransform target formats, only a subset of the capabilities of the Vimba ImageTransform library is implemented. The usable transformations are shown in Table 5 (for a complete list see the Vimba Image Transform Manual):

[1] BayerXX is any of BayerGR, BayerRG, BayerGB or BayerBG
[2] This format cannot be used for displaying

Source \ Target	Format8bppIndexed	Format16bppGrayScale[2]	Format24bppRgb	Format32bppRgb[2]
Mono8	✔	✔	✔	✔
Mono10	✔	-	✔	✔
Mono10p	✔	-	✔	✔
Mono12	✔	-	✔	✔
Mono12p	✔	-	✔	✔
Mono12Packed	✔	-	✔	✔
Mono14	✔	-	✔	✔
Mono16	✔	✔	✔	✔
BayerXX[1]8	✔	-	✔	✔
BayerXX[1]10	✔	✔	✔	✔
BayerXX[1]12	✔	✔	✔	✔
BayerXX[1]12Packed	✔	✔	✔	✔
BayerXX[1]12p	✔	✔	✔	✔
BayerXX[1]16	✔	✔	✔	✔
Rgb8	✔	-	✔	✔
Bgr8	✔	-	✔	-
Argb8	✔	-	✔	✔
Bgra8	✔	-	✔	✔
Yuv411	✔	-	✔	✔
Yuv422	✔	-	✔	✔
Yuv444	✔	-	✔	✔

Converting an image from a lower to a higher bit depth aligns the image data to the most significant bit.

Table 5: Possible transformations by method `Frame.Fill()`

If you omit the creation of a suitable bitmap and just provide a null pointer reference to `Frame.Fill()`, a default conversion is performed that can be looked up in Table 6:

Source Format	Target Format
Mono8	Format8bppIndexed
Mono10	Format8bppIndexed
Mono10p	Format8bppIndexed
Mono12	Format8bppIndexed
Mono12p	Format8bppIndexed
Mono12Packed	Format8bppIndexed
Mono14	Format8bppIndexed
Mono16	Format8bppIndexed
BayerXX[3]8	Format8bppIndexed
BayerXX[3]10	Format24bppRgb
BayerXX[3]12	Format24bppRgb
BayerXX[3]12Packed	Format24bppRgb
BayerXX[3]12p	Format24bppRgb
BayerXX[3]16	Format24bppRgb
Rgb8	Format24bppRgb
Bgr8	Format24bppRgb
Argb8	Format24bppRgb
Bgra8	Format24bppRgb
Yuv411	Format24bppRgb
Yuv422	Format24bppRgb
Yuv444	Format24bppRgb

Table 6: Default transformations done by method `Frame.Fill()`

[3]BayerXX is any of BayerGR, BayerRG, BayerGB or BayerBG

4.10 Using Events

Events serve a multitude of purposes and can have several origins: The Vimba System, an Interface and cameras.

In Vimba, notifications are issued as a result to a feature invalidation of either its value or its state. Consequently, to get notified about any feature change, add a delegate of the desired type (`OnCameraListChangeHandler`, `OnInterfaceListChangeHandler`, or `OnFeatureChangeHandler`) with the appropriate OnXXXChanged method (`OnCameraListChanged()`, `OnInterfaceListChanged`, or `OnFeatureChanged`), which gets called if there is a change to that feature.

Three examples are listed in this chapter. Notifications about:

- Camera list changes
- Feature changes
- Occurring camera events

See Listing 8 for an example of being notified about **camera list changes**.

Listing 8: Getting notified about camera list changes

```
// 1. Implement the delegate for OnCameraListChanged event
private void CamListChanged(VmbUpdateTriggerType reason)
{
  // React to the reason the event was fired. Possible values are:
  // UpdateTriggerPluggedIn (0), a new camera was discovered
  // UpdateTriggerPluggedOut (1), a known camera disappeared from the bus
  // UpdateTriggerOpenStateChanged (3), a known camera was opened or closed
  // by another application
  Console.WriteLine(reason);
}

{
  // 2. Start Vimba
  Vimba sys = new Vimba();
  sys.Startup();

  // 3. Register your event handler to the desired event
  sys.OnCameraListChanged += new Vimba.OnCameraListChangeHandler(CamListChanged);
}
```

See Listing 9 for an example of being notified about **feature changes**.

Listing 9: Getting notified about feature changes

```
// 1. Implement the delegate for OnFeatureChanged event
private void FeatureChanged(Feature feature)
{
  Console.WriteLine(feature.Name);
}

{
  // 2. Get camera features
  Camera     camera;
  FeatureCollection features = camera.Features;

  // 3. Register your event handler to the desired event
  Feature feature = features["Width"];
  feature.OnFeatureChanged += new Feature.OnFeatureChangeHandler(FeatureChanged);

  // As an example, binning is changed (which in turn influences the image width),
  // so the change event will be fired
  Feature featureBin = features["BinningHorizontal"];
  featureBin.Value = 8;
}
```

GigE camera events are also handled with the same mechanism of feature invalidation. See Listing 10 for an example. For more details about camera events, see (if installed):

- GigE Features Reference (GigE camera features)

Listing 10: Getting notified about occurring camera events

```
// 1. Implement the delegate for FeatureChanged event
private void FeatureChanged(Feature feature)
{
  Console.WriteLine(feature.Name);
}

{
  // 2. Get camera features
  Camera     camera;
  FeatureCollection features = camera.Features;

  // 3. Select the desired camera event
  Feature feature = features["EventSelector"];
  feature.EnumValue = "ExposureEnd";

  // 4. Enable the corresponding camera event notification
  feature = features["EventNotification"];
  feature.EnumValue = "On";

  // 5. Register your event handler to this event
  feature = features["EventExposureEnd"];
  feature.OnFeatureChanged += new Feature.OnFeatureChangeHandler(FeatureChanged);
}
```

4.11 Saving and loading settings

Additionally to the user sets stored inside the cameras, you can save the feature values as an XML file to your host PC. For example, you can configure your camera with Vimba Viewer, save the settings as a file, and load them with Vimba API. To do this, use methods `LoadCameraSettings` and `SaveCameraSettings`.

For a quick start, see example LoadSaveSettings.

To control which features are saved, use the method `LoadSaveSettingsSetup` (see comments in the example LoadSaveSettings). Note that saving and loading all features including look-up tables may take several minutes. You can manually edit the XML file if you want only certain features to be restored.

4.12 Triggering cameras

Before triggering, startup Vimba and open the camera(s).

To easily configure the camera's trigger settings, use Vimba Viewer and save/load the settings.

4.12.1 External trigger

The following code snippet shows how to trigger your camera with an external device.

Listing 11: External trigger

```
switch(camera.InterfaceType)
{
    case VmbInterfaceType.VmbInterfaceEthernet:

        pFeature = camera.Features["TriggerSelector"];
        pFeature.EnumValue = "FrameStart";
        pFeature = camera.Features["TriggerSource"];
        pFeature.EnumValue = "Line1";
        pFeature = camera.Features["TriggerMode"];
        pFeature.EnumValue = "On";
        break;

    case VmbInterfaceType.VmbInterfaceUsb:

        pFeature = camera.Features["LineSelector"];
        pFeature.EnumValue = "Line0";
        pFeature = camera.Features["LineMode"];
        pFeature.EnumValue = "Input";
        pFeature = camera.Features["TriggerSelector"];
        pFeature.EnumValue = "FrameStart";
        pFeature = camera.Features["TriggerSource"];
        pFeature.EnumValue = "Line0";
        pFeature = camera.Features["TriggerMode"];
        pFeature.EnumValue = "On";
        break;

    case VmbInterfaceType.VmbInterfaceFirewire:

            pFeature = camera.Features["LineSelector"];
            pFeature.EnumValue = "Line0";
            pFeature = camera.Features["LineMode"];
            pFeature.EnumValue = "Input";
            pFeature = camera.Features["LineRouting"];
            pFeature.EnumValue = "Trigger";
            pFeature = camera.Features["TriggerSelector"];
            pFeature.EnumValue = "ExposureStart";
            pFeature = camera.Features["TriggerSource"];
            pFeature.EnumValue = "InputLines";
            pFeature = camera.Features["TriggerMode"];
            pFeature.EnumValue = "On";
            break;
}
```

4.12.2 Trigger over Ethernet – Action Commands

Triggering via the *AcquisitionStart* command (see chapter Image Acquisition) is supported by all cameras. However, it is less precise than triggering with an external device connected to the camera's I/O port.

Selected GigE cameras with the latest firmware additionally support Action Commands. With Action Commands, you can broadcast a trigger signal simultaneously to multiple GigE cameras via GigE cable. Action Commands must be set first to the camera(s) and then to the Vimba API, which sends the Action Commands to the camera(s). As trigger source, select *Action0* or *Action1*.

ActionControl parameters

The following ActionControl parameters must be configured on the camera(s) and then on the host PC.

- `ActionDeviceKey` must be equal on the camera and on the host PC. Before a camera accepts an Action Command, it verifies if the received key is identical with its configured key. Note that `ActionDeviceKey` must be set each time the camera is opened.
  Range (camera and host PC): 0 to 4294967295
- `ActionGroupKey` means that each camera can be assigned to exactly one group for Action0 and a different group for Action1. All grouped cameras perform an action at the same time. If this key is identical on the sender and the receiving camera, the camera performs the assigned action.
  Range (camera and host PC): 0 to 4294967295
- `ActionGroupMask` serves as filter that specifies which cameras within a group react on an Action Command. It can be used to create sub-groups.
  Range (camera): 0 to 4294967295
  Range (host PC): 1 to 4294967295

Executing the API feature `ActionCommand` sends the ActionControl parameters to the cameras and triggers the assigned action, for example, image acquisition. Before an Action Command is executed, each camera validates the received ActionControl parameter values against its configured values. If they are not equal, the camera ignores the command.

More information

For more information about Action Commands, see:

- The ActionCommands programming example of the Vimba SDK
- The application note Trigger over Ethernet - Action Commands
- Action Commands as camera features are described in the GigE Features Reference.
- Action Commands as Vimba features are listed in the Vimba Manual.
- Listing 12 shows how to send out an Action Command to all connected cameras via all known Gigabit Ethernet interfaces.

Listing 12: Action Commands

```
// Additionally to this code snippet:
// Configure the trigger settings and add image streaming

int deviceKey = 11, groupKey = 22, groupMask = 33;

// Start Vimba
Vimba system = new Vimba();
system.Startup();

// Get cameras
CameraCollection cameras = system.Cameras;

foreach( Camera camera in cameras )
{
    // Open camera
    camera.Open( VmbAccessModeType.VmbAccessModeFull );

    // Set Action Device Key
    Feature cameraFeature = camera.Features["ActionDeviceKey"];
    cameraFeature.IntValue = deviceKey;

    // Set Action Group Key
    cameraFeature = camera.Features["ActionGroupKey"];
    cameraFeature.IntValue = groupKey;

    // Set Action Group Mask
    cameraFeature = camera.Features["ActionGroupMask"];
    cameraFeature.IntValue = groupMask;
}

    // Set Device Key to API
    Feature systemFeature = system.Features["ActionDeviceKey"];
    systemFeature.IntValue = deviceKey;

    // Set Group Key to API
    systemFeature = system.Features["ActionGroupKey"];
    systemFeature.IntValue = groupKey;

    // Set Group Mask to API
    systemFeature = system.Features["ActionGroupMask"];
    systemFeature.IntValue = groupMask;

    // Send Action Command
    systemFeature = system.Features["ActionCommand"];
    systemFeature.RunCommand();

    // Close all cameras
    foreach( Camera camera in cameras )
    {
        camera.Close();
    }

    // Shutdown Vimba
    system.Shutdown();
```

4.13 Additional configuration: Listing interfaces

`Vimba.Interfaces` enumerates all interfaces (GigE, USB, or 1394 adapters) recognized by the underlying transport layers.
See Listing 13 for an example.

Listing 13: List Interfaces

```
string strName;
Vimba sys = new Vimba();
InterfaceCollection interfaces=null;

try
{
  sys.Startup();
  interfaces = sys.Interfaces;

  Console.WriteLine( "Interfaces found: " + interfaces.Count );
  Console.WriteLine();

  foreach (Interface interFace in interfaces)
  {
    try
    {
      strName = interFace.Name;
    }
    catch ( VimbaException ve )
    {
      strName = ve.Message;
    }
    Console.WriteLine( "Interface Name: " + strName );
  }
}
finally
{
    sys.Shutdown();
}
```

The `Interface` class provides the following properties to obtain information about an interface listed in Table 7.

To **get notified when an interface is detected or disconnected**, use `Vimba.OnInterfaceListChangeHandler` (see chapter Using Events). In the implementation of this delegate, it is possible to react on interfaces being plugged in or out as it will get called by Vimba .NET API on the according event.

`Vimba.Shutdown()` blocks until all events have finished execution.

Type	Name	Purpose
`string`	`Id`	The unique ID
`string`	`Name`	The name
`VmbInterfaceType`	`Type`	The camera interface type
`string`	`SerialNumber`	The serial number (not in use)
`VmbAccessModeType`	`PermittedAccess`	The mode to open the interface

Table 7: Basic properties of Interface class

Calling certain methods within the event routine would counteract the reason of the event or lead to a deadlock. Therefore these methods must **not** be called within the event routine:

- `Feature.xxxValue.set()`
- `Feature.RunCommand()`
- `Interface.Open()`
- `Interface.Close()`
- `Vimba.Startup()`
- `Vimba.Shutdown()`
- `Vimba.Interfaces.get()`
- `Vimba.OpenInterfaceByID()`

4.14 Troubleshooting

4.14.1 GigE cameras

Make sure to set the *PacketSize* feature of GigE cameras to a value supported by your network card. If you use more than one camera on one interface, the available bandwidth has to be shared between the cameras.

- *GVSPAdjustPacketSize* configures GigE cameras to use the largest possible packets.
- *StreamBytesPerSecond* enables to configure the individual bandwidth if multiple cameras are used.
- The maximum packet size might not be available on all connected cameras. Try to reduce the packet size.

Further readings:

https://www.alliedvision.com/fileadmin/content/documents/products/cameras/various/installation-manual/GigE_Installation_Manual.pdf provides detailed information on how to configure your system.

4.14.2 USB cameras

Under Windows, make sure the correct driver is applied. For more details, see Vimba Manual, chapter Vimba Driver Installer.

In order to achieve best performance, see the technical manual of your USB camera, chapter Troubleshooting:
https://www.alliedvision.com/en/support/technical-documentation.html

4.14.3 Goldeye CL cameras

- The pixel format, all features affecting the image size, and DeviceTapGeometry must be identical in Vimba and the frame grabber software.
- Make sure to select an image size supported by the frame grabber.
- The baud rate of the camera and the frame grabber must be identical.

4.15 Error Codes

All Vimba API methods may throw a `VimbaException`. Each exception contains an error `ReturnCode` of type `VmbErrorType` whose possible values are listed in table 8.

Error Code	Value	Description
VmbErrorSuccess	0	No error
VmbErrorInternalFault	-1	Unexpected fault in Vimba or driver
VmbErrorApiNotStarted	-2	Startup was not called before the current comand
VmbErrorNotFound	-3	The designated instance (camera, feature etc.) cannot be found
VmbErrorBadHandle	-4	The given handle is not valid
VmbErrorDeviceNotOpen	-5	Device was not opened for usage
VmbErrorInvalidAccess	-6	Operation is invalid with the current access mode
VmbErrorBadParameter	-7	One of the parameters was invalid (usually an illegal pointer)
VmbErrorStructSize	-8	The given struct size is not valid for this version of the API
VmbErrorMoreData	-9	More data was returned in a string/list than space was provided
VmbErrorWrongType	-10	The feature type for this access method was wrong
VmbErrorInvalidValue	-11	The value was not valid; either out of bounds or not an increment of the minimum
VmbErrorTimeout	-12	Timeout during wait
VmbErrorOther	-13	Other error
VmbErrorResources	-14	Resources not available (e.g. memory)
VmbErrorInvalidCall	-15	Call is invalid in the current context (e.g. callback)
VmbErrorNoTL	-16	No transport layers were found
VmbErrorNotImplemented	-17	API feature is not implemented
VmbErrorNotSupported	-18	API feature is not supported
VmbErrorIncomplete	-19	The current operation was not completed (e.g. a multiple registers read or write)
VmbErrorIO	-20	There was an error during read or write with devices (camera or disk)
VmbErrorNETBadParameter	-100	One of the .NET parameters was invalid (usually an illegal reference or empty)
VmbErrorNETIncomplete	-101	A multiple .NET frame acquisition was incomplete
VmbErrorNETInvalidCall	-102	A .NET call is invalid in the current context (e.g. acquisition is already running)
VmbErrorNETNotSupported	-103	A .NET parameter value is not supported (e.g. unsupported pixelformat)
VmbErrorNETInternalFault	-104	Unexpected fault in VmbAPINET

Table 8: Error codes returned by Vimba exceptions

5 Function reference

This chapter includes:

This chapter lists available methods and properties.

Methods in this chapter are described in this way:

- The caption states the name of the method with empty parentheses.
- The first item is a brief description.
- The parameters of the method are listed in a table (with name and description).
- The return value is stated.
- Finally, the exceptions that this method may throw are listed.

Properties are described in this way:

- The caption states the name of the property.
- The first item it a brief description.
- The return value is stated.
- Finally, the exceptions that may be thrown are listed.

5.1 AncillaryData

AncillaryData encapsules additional data that was sent together with an image.
It is equivalent to GenICam's chunk data.

5.1.1 Buffer

The raw memory buffer of the ancillary data .

Return value: array of Bytes

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.1.2 Close()

Drops access to the ancillary data inside a frame.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.1.3 Features

A collection of Features dependent on this object.

Return value: FeatureCollection object

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.1.4 Open()

Allows access to the elements of the ancillary data, either via feature access or via the raw buffer.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

Since ancillary data is only valid for a single frame, but receives its definition from the camera the frame came from, it should be closed as soon as possible.

5.2 Camera

A Camera object represents a physical camera.
Before opening it with `Camera.Open` , only the static properties can be queried.
After opening, the camera features are accessible and you may acquire images.

5.2.1 Camera()

Protected constructor. Only for use in derived classes.

Name	Description
cameraID	The ID of the camera.
cameraName	The name of the camera.
cameraModel	The model name of the camera.
cameraSerialNumber	The serial number of the camera.
interfaceID	The ID of the interface the camera is connected to.
interfaceType	The type of the interface the camera is connected to.
interfaceName	The name of the interface the camera is connected to.
interfaceSerialNumber	The serial number of the interface the camera is connected to.
interfacePermittedAccess	The access mode of the interface the camera is connected to.

5.2.2 AcquireMultipleImages()

Gets multiple images synchronously.

Name	Description
frames	Reference to array of empty Frame objects. The size of the array determines the exact number of images to grab.
timeout	timeout how long the method should wait for the image (in milliseconds). The timeout is used for each image, not for the whole sequence.

Exceptions:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or when the passed Frame array is invalid.
- **System.TimeoutException:** Thrown if acquisition times out.
- **System.ExecutionEngineException:** Thrown on issues in the system environment.

This is a convenience method, doing all the necessary startup methods and cleanup afterwards.

If a timeout is encountered, the method is aborted.

5.2.3 AcquireMultipleImages()

Gets multiple images synchronously and returns the number of valid frames.

Name	Description
frames	Reference to array of empty Frame objects. The size of the array determines the maximum number of images to grab.
timeout	timeout how long it should wait for the image (in milliseconds). The timeout is used for each image, not for the whole sequence.
numFramesCompleted	number of frames with valid frame data.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or when the passed Frame array is invalid.

This is a convenience method, doing all the necessary startup methods and cleanup afterwards.

5.2.4 AcquireSingleImage()

Acquires one image into a `Frame` .

Name	Description
frame	Reference to an empty Frame object. It will contain the image after the call.
timeout	Timeout how long it should wait for the image (in milliseconds).

Exceptions:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the Vimba environment.
- **System.TimeoutException:** Thrown if acquisition times out.
- **System.ExecutionEngineException:** Thrown on issues in the system environment.

This is a convenience method, doing all the necessary startup methods and cleanup afterwards, so using it in a loop will be relatively slow. Instead, consider using `AcquireMultipleImages` or `StartContinuousImageAcquisition`

5.2.5 AnnounceFrame()

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

Name	Description
frame	Frame to announce.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or when the passed Frame is invalid.

5.2.6 Close()

Closes a Camera. Frees the access to this camera.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.7 EndCapture()

Stops the API from being able to receive frames from this camera.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.8 Features

A feature collection of the camera.

Return value: FeatureCollection.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.9 FlushQueue()

Flushes the capture queue.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.10 Id

Gets the camera id.

Return value: System::String.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.11 InterfaceID

Gets the camera interface id.

Return value: System::String.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.12 InterfaceType

Gets the type of the interface the camera is connected to, and consequently the type of the camera itself.

Return value: VmbInterfaceType.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.13 LoadCameraSettings()

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

Name	Description
fileName	Determines name and/or path of xml file to be loaded

Camera must be opened

5.2.14 LoadSaveSettingsSetup()

Sets Load/Save settings behaviour

Name	Description
persistType	Determines which feature shall be saved/loaded
maxIterations	Determines how many iterations shall be used during loading

Camera must be opened

5.2.15 Model

Gets the camera model.

Return value: System::String.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.16 Name

Gets the camera name.

Return value: System::String.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.17 OnFrameReceived()

Event will be triggered when a Frame buffer is filled with data. See `AVT.VmbAPINET.Camera.OnFrameReceivedHandler` for the expected signature of event subscribers.

5.2.18 OnFrameReceivedHandler()

Delegate used for OnFrameReceived events.

5.2.19 Open()

Opens a Camera. Use this method to access features of a camera and - in case of full access - to acquire images.

Name	Description
accessMode	Parameter of type VmbAccessModeType to set the desired accessibility.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.20 PermittedAccess

Gets the permitted access of the camera.

Return value: VmbAccessModeType.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.21 QueueFrame()

Queues a frame that may be filled during frame capturing.

Name	Description
frame	Frame to queue.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or when the passed Frame is invalid.

5.2.22 ReadMemory()

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

Name	Description
address	start address to read.
bufferArray	Reference to array of data buffer. Memory will be saved to this array.
readLength	uiReadLength determines how many bytes to read.
completedReads	Reference to an uint. The number of read memory bytes will be saved to this variable.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or when the length is invalid.

5.2.23 ReadRegisters()

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

Name	Description
addressArray	Array of addresses. Size of address buffer determines how many registers to read.
dataArray	Reference to array of data. Registers will be saved to this array.
completedReads	Reference to an uint. The number of read registers will be saved to this variable.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or when the address array is invalid.

5.2.24 RevokeAllFrames()

Revokes all frames assigned to this certain camera.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.25 RevokeFrame()

Revokes a frame from the API.

Name	Description
frame	Frame to revoke.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or when the passed Frame is invalid.

5.2.26 SaveCameraSettings()

Saves the current camera setup to an XML file

Name	Description
fileName	Determines name and/or path of xml file to be saved

Camera must be opened

5.2.27 SerialNumber

Gets the camera serial number.

Return value: System::String.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.28 StartCapture()

Prepares the API for incoming frames from this camera.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.2.29 StartContinuousImageAcquisition()

Starts streaming and allocates the needed frames.

Name	Description
frameCount	count of Frame(s) which should be used for this method.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or when the passed frame count is invalid.

This is a convenience method.
Register a FrameObserver and queue the received Frame(s).

5.2.30 StopContinuousImageAcquisition()

Stops streaming and frees the used frames

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

This is a convenience method. Deregister the FrameObserver again.

5.2.31 ToString()

Returns the name of camera.

Return value: System::String.

5.2.32 WriteMemory()

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

Name	Description
address	start address to write.
bufferArray	array of data buffer.
completedWrites	Reference to an uint. The number of read memory bytes will be saved to this variable.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or when the length is invalid.

5.2.33 WriteRegisters()

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

Name	Description
addressArray	Array of addresses. Size of address buffer determines how many registers to write.
dataArray	Array of register data.
completedWrites	Reference to an uint. The number of written registers will be saved to this variable.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or when the address/data array is invalid.

5.3 CameraCollection

Collects functionality for a list of cameras, e.g. used in the `Vimba.Cameras` property.

5.3.1 CopyTo()

Copies the elements of the ICollection to an Array, starting at a particular Array index.

Name	Description
items	The one-dimensional Array that is the destination of the elements copied from ICollection.
index	The zero-based index in array at which copying begins.

Exceptions:

- **System.IndexOutOfRangeException:** Thrown when index exceeds range.
- **System.ArgumentNullException:** array is null.
- **System.ArgumentOutOfRangeException:** index is less than zero.
- **System.ArgumentException:** array is multidimensional.
  -or- index is equal to or greater than the length of array.
  -or- The number of elements in the source System.Collections.ICollection is greater than the available space from index to the end of the destination array.

5.3.2 Count

Gets the number of elements contained in the collection.

Return value: An int representing the number of elements.

5.3.3 GetEnumerator()

Returns an enumerator that iterates through the camera collection.

Return value: A System::Collections::IEnumerator for iterating through the collection

5.3.4 IsSynchronized

Gets a value indicating whether access to the ICollection is synchronized (thread safe).

Return value: A bool value

5.3.5 Item

Fetches the camera with the given `index` from the current list.

Name	Description
index	Index of the camera to get.

Return value: A Camera object.

Exception:

- **System.IndexOutOfRangeException:** Thrown when index exceeds range.

5.3.6 SyncRoot

Gets an object that can be used to synchronize access to the ICollection.

Return value: A System::Object.

5.3.7 default

Fetches the camera with the given ID from the current list.

Name	Description
cameraId	Camera id.

Return value: A Camera object.

Exceptions:

- **System.ArgumentOutOfRangeException:** Thrown when index exceeds range.
- **System.ArgumentNullException:** cameraID is null.

5.4 CameraCollectionEnumerator

Supports a simple iteration over a `CameraCollection` . See `System.Collections.IEnumerator` . See also `AVT.VmbAPINET.CameraCollection` .

5.5 EnumEntry

Provides Enumeration feature functionality.

5.5.1 Description

Gets the description of an enumeration.
Return value: String value.

5.5.2 DisplayName

Gets a more declarative name of an enumeration.
Return value: String value.

5.5.3 Name

Gets the name of an enumeration.
Return value: String value.

5.5.4 SFNCNamespace

Gets the standard feature naming convention namespace of the enumeration.
Return value: String value.

5.5.5 Tooltip

Gets a tooltip that can be used as pop up help in a GUI.
Return value: String value.

5.5.6 Value

Gets the integer value of an enumeration.
Return value: Int64 value.

5.5.7 Visibility

Gets the visibility of an enumeration.

Return value: VmbFeatureVisibilityType value.

5.6 EnumEntryCollection

Collects functionality for a list of EnumEntry items, e.g. used in the `Feature` class.

5.6.1 CopyTo()

Copies the elements of the ICollection to an Array, starting at a particular Array index.

Name	Description
items	The one-dimensional Array that is the destination of the elements copied from ICollection.
index	The zero-based index in array at which copying begins.

Exceptions:

- **System.IndexOutOfRangeException:** Thrown when index exceeds range.
- **System.ArgumentNullException:** array is null.
- **System.ArgumentOutOfRangeException:** index is less than zero.
- **System.ArgumentException:** array is multidimensional. -or- index is equal to or greater than the length of array. -or- The number of elements in the source System.Collections.ICollection is greater than the available space from index to the end of the destination array.

5.6.2 Count

Gets the number of elements contained in the ICollection.

5.6.3 GetEnumerator()

Returns an enumerator that iterates through a collection.

5.6.4 IsSynchronized

Gets a value indicating whether access to the ICollection is synchronized (thread safe).

5.6.5 Item

Gets the EnumEntry of the selected input index.

Name	Description
index	Number of EnumEntry list index to get.

Return value: EnumEntry object.

Exception:

- **System.IndexOutOfRangeException:** Thrown when index exceeds range.

5.6.6 SyncRoot

Gets an object that can be used to synchronize access to the ICollection.

Return value: Returns an object that can be used to synchronize access to the ICollection.

5.6.7 default

Gets the EnumEntry of the selected input string ID.

Name	Description
enumEntry	String of the enum entry.

Return value: EnumEntry object.

Exceptions:

- **System.ArgumentOutOfRangeException:** Thrown when index exceeds range.
- **System.ArgumentNullException:** string is null.

5.7 EnumEntryCollectionEnumerator

Supports a simple iteration over an `EnumEntryCollection` . See `System.Collections.IEnumerator` . See also `AVT.VmbAPINET.EnumEntryCollection` .

5.8 Feature

Collects all functionality for Vimba features.

5.8.1 AffectedFeatures

Gets collection of Features.

Return value: FeatureCollection object.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.2 BoolValue

Sets/Gets value of Feature.

Return value: Boolean value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.3 Category

Gets value of Feature.

Return value: String value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.4 DataType

Gets value of Feature.

Return value: VmbFeatureDataType value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.5 Description

Gets value of Feature.

Return value: String value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.6 DisplayName

Gets value of Feature.

Return value: String value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.7 EnumEntries

Gets collection of EnumEntries.

Return value: EnumEntryCollection object.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.8 EnumIntValue

Sets/Gets int value of Feature.

Return value: int value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.9 EnumIntValues

Gets all possible Int64 enums of Feature range in an array of Int64.

Return value: Int64[] value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.10 EnumValue

Sets/Gets value of Feature.
Return value: String value.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.11 EnumValues

Gets all possible string enums of Feature range in an array of strings.
Return value: String[] value.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.12 Flags

Gets value of Feature.
Return value: VmbFeatureFlagsType value.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.13 FloatHasIncrement

Gets float increment support of the Feature.
Return value: bool support state.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.14 FloatIncrement

Gets float increment value of Feature.
Return value: double value for increment.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown if not implemented or on issues in the environment.

5.8.15 FloatRangeMax

Gets maximum value of Feature range.

Return value: Double value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.16 FloatRangeMin

Gets minimum value of Feature range.

Return value: Double value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.17 FloatValue

Sets/Gets value of Feature.

Return value: Double value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.18 IntIncrement

Gets int increment value of Feature.

Return value: long value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.19 IntRangeMax

Gets maximum value of Feature range.

Return value: long value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.20 IntRangeMin

Gets minimum value of Feature range.

Return value: long value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.21 IntValue

Sets/Gets value of Feature.

Return value: long value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.22 IsCommandDone()

Checks if command Feature is done.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.23 IsEnumValueAvailable()

Indicates whether an existing enumeration-value is currently available.

An enumeration value might not be selectable due to the camera's current configuration.

Name	Description
enumValue	int64 enum value to test.

Return value: Boolean value.

Exception:

- **AVT.VmbAPINET.VimbaException:**
  Thrown on issues in the environment. Possible ReturnCodes are:
  - VmbErrorSuccess: If no error
  - VmbErrorInvalidValue: If the given value is not a valid enumeration-value for this enum
  - VmbErrorApiNotStarted: Startup() was not called before the current command
  - VmbErrorInvalidAccess: Operation is invalid with the current access mode
  - VmbErrorWrongType: The feature is not an enumeration

5.8.24 IsEnumValueAvailable()

Indicates whether an existing enumeration-value is currently available.

An enumeration value might not be selectable due to the camera's current configuration.

Name	Description
enumValue	String enum value to test.

Return value: Boolean value.

Exception:

- **AVT.VmbAPINET.VimbaException:**
  Thrown on issues in the environment. Possible ReturnCodes are:
  - VmbErrorSuccess: If no error
  - VmbErrorInvalidValue: If the given value is not a valid enumeration-value for this enum
  - VmbErrorApiNotStarted: Startup() was not called before the current command
  - VmbErrorInvalidAccess: Operation is invalid with the current access mode
  - VmbErrorWrongType: The feature is not an enumeration

5.8.25 IsReadable()

Checks if it is readable.

Return value: Boolean value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.26 IsStreamable()

Checks if it is streamable.

Return value: Boolean value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.27 IsWritable()

Checks if it is writable.

Return value: Boolean value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.28 Name

Gets value of Feature.

Return value: String value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.29 OnFeatureChangeHandler()

Delegate used for OnFeatureChanged events

Name	Description
	Feature that was the reason of this event

5.8.30 OnFeatureChanged()

Event will be triggered when the feature changes. See `AVT.VmbAPINET.Feature.OnFeatureChangeHandler` for the expected signature of event subscribers.

5.8.31 PollingTime

Gets value of Feature.

Return value: int value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.32 RawValue

Sets/Gets value of Feature.

Return value: Byte[] value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.33 Representation

Gets value of Feature.

Return value: String value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.34 RunCommand()

Executes a command Feature.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.35 SFNCNamespace

Gets value of Feature.

Return value: String value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.36 SelectedFeatures

Gets collection of Features.

Return value: FeatureCollection object.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.37 StringValue

Sets/Gets value of Feature.

Return value: String value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.38 ToString()

Returns name of feature.

5.8.39 ToolTip

Gets value of Feature.

Return value: String value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.40 Unit

Gets value of Feature.

Return value: String value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.8.41 Visibility

Gets value of Feature.

Return value: VmbFeatureVisibilityType value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.9 FeatureCollection

Collects functionality for a list of features, e.g. used in `Camera, Interface` , or `AncillaryData` . See `AVT.VmbAPINET.Feature` . See also `AVT.VmbAPINET.Camera.Features` . See also `AVT.VmbAPINET.Interface.Features` . See also `AVT.VmbAPINET.AncillaryData.Features` .

5.9.1 ContainsName()

Checks the existence of a feature by a feature name string.

Name	Description
name	String of the feature name.

Return value: Boolean value.
Exception:

- **System.ArgumentNullException:** Thrown if name string is null.

5.9.2 CopyTo()

Copies the elements of the ICollection to an Array, starting at a particular Array index.

Name	Description
items	The one-dimensional Array that is the destination of the elements copied from ICollection.
index	The zero-based index in array at which copying begins.

Exceptions:

- **System.IndexOutOfRangeException:** Thrown when index exceeds range.
- **System.ArgumentNullException:** array is null.
- **System.ArgumentOutOfRangeException:** index is less than zero.
- **System.ArgumentException:** array is multidimensional. -or- index is equal to or greater than the length of array. -or- The number of elements in the source System.Collections.ICollection is greater than the available space from index to the end of the destination array.

5.9.3 Count

Gets the number of elements contained in the ICollection.

5.9.4 GetEnumerator()

Returns an enumerator that iterates through a collection.

5.9.5 IsSynchronized

Gets a value indicating whether access to the ICollection is synchronized (thread safe).

5.9.6 Item

Gets the Feature of the selected input index.

Name	Description
index	Number of feature list index to get.

Return value: Feature object.
Exception:

- **System.IndexOutOfRangeException:** Thrown when index exceeds range.

5.9.7 SyncRoot

Gets an object that can be used to synchronize access to the ICollection.
Return value: Returns an object that can be used to synchronize access to the ICollection.

5.9.8 default

Gets the Feature of the selected input string name.

Name	Description
name	String of the feature name.

Return value: Feature object.
Exceptions:

- **System.ArgumentOutOfRangeException:** Thrown when index exceeds range.
- **System.ArgumentNullException:** featurename is null.

5.10 FeatureCollectionEnumerator

Supports a simple iteration over a `FeatureCollection` . See `System.Collections.IEnumerator` . See also `AVT.VmbAPINET.FeatureCollection` .

5.11 Frame

Collects functionality for handling image data sent by a camera.

5.11.1 Frame()

Creates an instance of class Frame.

Name	Description
buffer	Buffer array to use.

5.11.2 Frame()

Creates an instance of class Frame.

Name	Description
bufferSize	Size of internal buffer to be created.

5.11.3 AncillaryData

Returns the part of a frame that describes the chunk data as an object. See `AVT.VmbAPINET.AncillaryData`

Return value: AncillaryData object.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.4 AncillarySize

Returns the memory size of the chunk data.

Return value: System::UInt32.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.5 Buffer

Returns the complete buffer including image and chunk data.

Return value: Byte[] array.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.6 BufferSize

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

Return value: uint value.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.7 Fill()

Fills the given bitmap with current buffer data in an appropriate pixel format.
If `bitmap` is null, an appropriate object will be created and filled with frame data.
If `bitmap` is not null, the existing object will be filled with frame data.
For more details, see chapter "Converting Frames into Bitmaps" in the Vimba .NET manual.

Name	Description
bitmap	Reference to a bitmap.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment or bitmap failure.

5.11.8 FrameID

Returns the frame ID.

Return value: System::UInt64.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.9 Height

Returns the height of the image.
Return value: System::UInt32.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.10 ImageSize

Returns the memory size of the image.
Return value: System::UInt32 value.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.11 OffsetX

Returns the x offset of the image.
Return value: System::UInt32.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.12 OffsetY

Returns the y offset of the image.
Return value: System::UInt32.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.13 PixelFormat

Returns the GeV compliant pixel format.
Return value: VmbPixelFormatType.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.14 ReceiveStatus

Returns the receive status of a frame.

Return value: VmbFrameStatusType.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.15 Timestamp

Returns the time stamp.

Return value: System::UInt64.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.11.16 Width

Returns the width of the image.

Return value: System::UInt32.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.12 IFrame

Interface to cover important properties of the `Frame` class.

5.12.1 Buffer

Gets the buffer of a Frame.

Return value: An array of unsigned char.

5.12.2 Height

Gets the height of a Frame.

Return value: System::UInt32.

5.12.3 PixelFormat

Gets the pixel format of a Frame.

Return value: VmbPixelFormatType.

5.12.4 Width

Gets the width of a Frame.

Return value: System::UInt32.

5.13 Interface

An Interface object represents a logical interface as provided by the underlying transport layers.
Before opening it with `Interface.Open`, only static properties can be queried.
After opening, the interface features are accessible.

5.13.1 Close()

Closes an interface. Will free the access to this interface.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.13.2 Features

Returns a feature collection of the interface.

Return value: FeatureCollection object.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.13.3 Id

Gets the ID of an interface.

Return value: System::String.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.13.4 Name

Gets the name of an interface.

Return value: System::String.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.13.5 Open()

Opens an Interface. Will provide access to an interface.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.13.6 PermittedAccess

Gets the access mode of an interface.

Return value: VmbAccessModeType.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.13.7 SerialNumber

Gets the serial number of an interface.

Return value: System::String.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.13.8 ToString()

Returns name of interface.

Return value: System::String.

5.13.9 Type

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

Return value: VmbInterfaceType.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.14 InterfaceCollection

Collects functionality for a list of `Interface` elements.

5.14.1 CopyTo()

Copies the elements of the ICollection to an Array, starting at a particular Array index.

Name	Description
items	The one-dimensional Array that is the destination of the elements copied from ICollection.
index	The zero-based index in array at which copying begins.

Exceptions:

- **System.IndexOutOfRangeException:** Thrown when index exceeds range.
- **System.ArgumentNullException:** array is null.
- **System.ArgumentOutOfRangeException:** index is less than zero.
- **System.ArgumentException:** array is multidimensional. -or- index is equal to or greater than the length of array. -or- The number of elements in the source System.Collections.ICollection is greater than the available space from index to the end of the destination array.

5.14.2 Count

Gets the number of elements contained in the ICollection.

5.14.3 GetEnumerator()

Returns an enumerator that iterates through a collection.

5.14.4 IsSynchronized

Gets a value indicating whether access to the ICollection is synchronized (thread safe).

5.14.5 Item

Gets the Interface of the selected input index.

Name	Description
index	Number of the interface list index to get.

Return value: Interface object.
Exception:

- **System.IndexOutOfRangeException:** Thrown when index exceeds range.

5.14.6 SyncRoot

Gets an object that can be used to synchronize access to the ICollection.
Return value: Returns an object that can be used to synchronize access to the ICollection.

5.14.7 default

Gets the Interface of the selected input string ID.

Name	Description
interfaceId	String of the interface id.

Return value: Interface object.
Exceptions:

- **System.ArgumentOutOfRangeException:** Thrown when index exceeds range.
- **System.ArgumentNullException:** interfaceId is null.

5.15 InterfaceCollectionEnumerator

Supports a simple iteration over an `InterfaceCollection` . See `System.Collections.IEnumerator` . See also `AVT.VmbAPINET.InterfaceCollection` .

5.16 Vimba

The Vimba singleton class is the entry point to the Vimba API.
To start working with Vimba, call `Startup()` first.
After finishing work, call `Shutdown()` .

5.16.1 Cameras

Retrieves a list of all cameras.

Return value: CameraCollection.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.16.2 CreateCamera()

See `AVT.VmbAPINET.Vimba.CreateCameraHandler` for the expected signature of event subscribers.

5.16.3 CreateCameraHandler()

Delegate used for CreateCamera events.

Name	Description
cameraID	Camera ID.
cameraName	Camera name.
cameraModel	Camera model.
cameraSerialNumber	Serial number of camera.
interfaceID	Interface ID.
interfaceType	Interface type.
interfaceName	Interface name.
interfaceSerialNumber	Serial number of interface.
interfacePermittedAccess	Permitted access of interface.

5.16.4 Features

Returns a feature collection of the system module.
Return value: FeatureCollection object.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.16.5 GetCameraByID()

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

Name	Description
cameraID	ID string used to select the camera.

Return value: Camera object.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown when ID not found or method parameter invalid.

5.16.6 GetInterfaceByID()

Gets a specific interface identified by an ID.

Name	Description
interfaceID	ID string used to select the interface.

Return value: Interface object.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown when ID not found or method parameter invalid.

5.16.7 Interfaces

Lists all the interfaces currently visible to Vimba API.
Return value: InterfaceCollection.
Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.16.8 OnCameraListChangeHandler()

Delegate used for OnCameraListChanged events.

Name	Description
reason	VmbUpdateTriggerType enum.

5.16.9 OnCameraListChanged()

Event that will be triggered when the camera list changes. See `AVT.VmbAPINET.Vimba.OnCameraListChangeHandler` for the expected signature of event subscribers.

5.16.10 OnInterfaceListChangeHandler()

Delegate used for OnInterfaceListChanged events.

Name	Description
reason	VmbUpdateTriggerType enum.

5.16.11 OnInterfaceListChanged()

Event that will be triggered when the interface list changes. See `AVT.VmbAPINET.Vimba.OnInterfaceListChangeHandler` for the expected signature of event subscribers.

5.16.12 OpenCameraByID()

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

Name	Description
cameraID	ID string used to select the camera.
accessMode	Parameter of type VmbAccessModeType to set the desired accessibility.

Return value: Camera object.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown when ID not found or method parameter invalid.

5.16.13 OpenInterfaceByID()

Open an interface for feature access.

Name	Description
interfaceID	ID string used to select the interface.

Return value: Interface object.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown when ID not found or method parameter invalid.

5.16.14 Shutdown()

Performs a shutdown on the API module.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.16.15 Startup()

Initializes the Vimba API module.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.16.16 Version

Retrieves the version number of Vimba API.

Return value: VmbVersionInfo-t structure.

Exception:

- **AVT.VmbAPINET.VimbaException:** Thrown on issues in the environment.

5.17 VimbaException

The exception that is thrown when there are problems in the execution of VimbaNET methods.

5.17.1 VimbaException()

Initializes a new instance of the AVT::VmbAPINET::VimbaException class.

Name	Description
returnCode	exception/result value
message	exception/result text

5.17.2 MapReturnCodeToString()

Maps current return code to a message.

Return value: System::String.

5.17.3 ReturnCode

Holds the return code of the underlying methods which triggered the exception.

Return value: System::Int32.