



Vimba

Vimba Manual for Windows

4.2.0

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

All texts, pictures and graphics are protected by copyright and other laws protecting intellectual property.

All rights reserved.

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

1	Contacting Allied Vision	8
2	Document history and conventions	9
2.1	Document history	10
2.2	Conventions used in this manual	10
2.2.1	Styles	11
2.2.2	Symbols	11
3	Vimba SDK Overview	12
3.1	Compatibility	13
3.2	Architecture	14
3.3	API Entities Overview	16
3.4	Features in Vimba	17
3.5	Vimba's Transport Layers	18
3.6	Synchronous and asynchronous image acquisition	18
3.7	Notifications	20
3.8	Building applications	21
3.8.1	Setting up Visual Studio for C and C++ projects	21
4	Vimba Class Generator	22
4.1	Main window	23
4.2	C++ code generation	24
4.3	C# code generation	25
5	Vimba Driver Installer	26
5.1	Main window	27
5.2	Changing drivers	28
5.3	Command line options	30
6	Vimba Firmware Updater	33
6.1	Uploading firmware	34
6.2	Aborting a firmware upload	35
6.3	Troubleshooting	36
6.4	Command line Firmware Updater	36
7	Vimba Deployment	38
7.1	Modules	39
7.2	Examples	40
8	Compiling the Vimba C++ API	41

9	Vimba - Feature Overview	42
10	Vimba System	43
10.1	Info [Allied Vision]	44
10.1.1	Elapsed [Allied Vision]	44
10.1.2	GeVTlIsPresent [Allied Vision]	44
10.1.3	FiWTlIsPresent [Allied Vision]	45
10.1.4	UsbTlIsPresent [Allied Vision]	45
10.1.5	CLTlIsPresent [Allied Vision]	45
10.2	Discovery [Allied Vision]	45
10.2.1	GeVDiscoveryAlloff [Allied Vision]	46
10.2.2	GeVDiscoveryAllAuto [Allied Vision]	47
10.2.3	GeVDiscoveryAllOnce [Allied Vision]	47
10.2.4	GeVDiscoveryStatus [Allied Vision]	47
10.2.5	GeVDiscoveryAllDuration [Allied Vision]	48
10.2.6	DiscoveryCameraIdent [Allied Vision]	48
10.2.7	DiscoveryCameraEvent [Allied Vision]	48
10.2.8	DiscoveryInterfaceIdent [Allied Vision]	49
10.2.9	DiscoveryInterfaceEvent [Allied Vision]	49
10.3	ForceIP [Allied Vision]	49
10.3.1	GevDeviceForceIP [Allied Vision]	50
10.3.2	GevDeviceForceMACAddress [Allied Vision]	50
10.3.3	GevDeviceForceIPAddress [Allied Vision]	50
10.3.4	GevDeviceForceSubnetMask [Allied Vision]	51
10.3.5	GevDeviceForceGateway [Allied Vision]	51
10.4	ActionControl [Allied Vision]	51
10.4.1	ActionCommand [Allied Vision]	51
10.4.2	ActionDeviceKey [Allied Vision]	52
10.4.3	ActionGroupKey [Allied Vision]	52
10.4.4	ActionGroupMask [Allied Vision]	52
10.4.5	GevActionDestinationIPAddress [Allied Vision]	53
11	Ancillary Data Features	54
11.1	ChunkData [Allied Vision]	55
11.1.1	ChunkAcquisitionFrameCount [Allied Vision]	55
11.1.2	ChunkUserValue [Allied Vision]	55
11.1.3	ChunkExposureTime [Allied Vision]	56
11.1.4	ChunkGain [Allied Vision]	56
11.1.5	ChunkSyncInLevels [Allied Vision]	56
11.1.6	ChunkSyncOutLevels [Allied Vision]	57
12	Vimba Cognex Adapter	58

13 References**59**

List of Figures

1	Vimba Architecture	14
2	Vimba API Entities	16
3	Acquisition Models	19
4	Vimba Class Generator - Main Window	23
5	Driver Installer - Main Window	27
6	Driver Installer - Prepare actions	28
7	Driver Installer - Incompatible products detected	29
8	Driver Installer - Installation successful	29
9	Firmware Updater - Main Window	34
10	Firmware Updater - Manual Update	35
11	Firmware Updater - Abort	35

List of Tables

1	Use cases for the command line Firmware Updater	37
2	Command options for the firmware update	37

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 Gerke, Peter Tix

2 Document history and conventions



This chapter includes:

2.1	Document history	10
2.2	Conventions used in this manual	10
2.2.1	Styles	11
2.2.2	Symbols	11

2.1 Document history

Version	Date	Changes
1.0	2012-11-26	Initial version
1.1	2013-04-03	Different links, small changes
1.2	2013-06-18	Added chapter for Class Generator, small corrections, layout changes
1.3	2014-07-09	Major rework of the whole document
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
2.0	2016-02-29	Added Goldeye CL compatibility, new document layout
2.1	2017-01-27	Updated layout and formatting
2.1.3	September 2017	Integration of Vimba Features Manual, added chapter Building applications
2.1.4	May 2018	Integration of Vimba Features Manual, added chapter Building applications
3.0.0	June 2019	Bug fixes
3.1.0	October 2019	GenTL 1.5 support, bug fixes, update of supported OS
4.0.0	May 2020	New Python API, several linguistic changes
4.2.0	October 2020	Bug fixes, update of supported OS

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	<i>Title</i>
Web reference	Links to web pages	Link
Document reference	Links to other documents	Document
Output	Outputs from software GUI	Output
Input	Input commands, modes	<i>Input</i>
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 Vimba SDK Overview



This chapter includes:

3.1	Compatibility	13
3.2	Architecture	14
3.3	API Entities Overview	16
3.4	Features in Vimba	17
3.5	Vimba's Transport Layers	18
3.6	Synchronous and asynchronous image acquisition	18
3.7	Notifications	20
3.8	Building applications	21
3.8.1	Setting up Visual Studio for C and C++ projects	21

3.1 Compatibility

Vimba for Windows is an SDK for all Allied Vision cameras with GigE, USB, and 1394 interface. Moreover, the configuration of Goldeye CL cameras is supported. Vimba is designed to be compatible with future Allied Vision cameras connected to other hardware interfaces. Since Vimba is based on GenICam, common third-party software solutions can easily be supported. Vimba includes GigE, USB, and 1394 drivers not only for performance reasons: While the use of the GigE filter driver is optional, the USB and 1394 drivers are required to allow access to Allied Vision USB and 1394 cameras.

Vimba also provides a Camera Link configuration transport layer, which supports the configuration of Goldeye CL cameras. Image streaming is handled via the API provided by the frame grabber manufacturer. Vimba functions requiring image streaming are unavailable for Camera Link cameras.

Supported cameras

- Allied Vision GigE cameras
- Allied Vision USB cameras
- Allied Vision 1394 cameras
- Allied Vision Goldeye CL cameras (configuration only)

Supported operating systems

- Windows 7 (64-bit)
- Windows 10 (64-bit)

Vimba and third-party software

Vimba's transport layers (GenTL producers) are based on GenICam and therefore can be used with any third-party software that includes a GenTL consumer. For more information, see chapter Vimba's Transport Layers.

Cognex VisionPro users can access Allied Vision cameras via the Vimba Cognex Adapter. If installed, see the [Vimba Cognex Adapter Manual](#) for more details.

3.2 Architecture

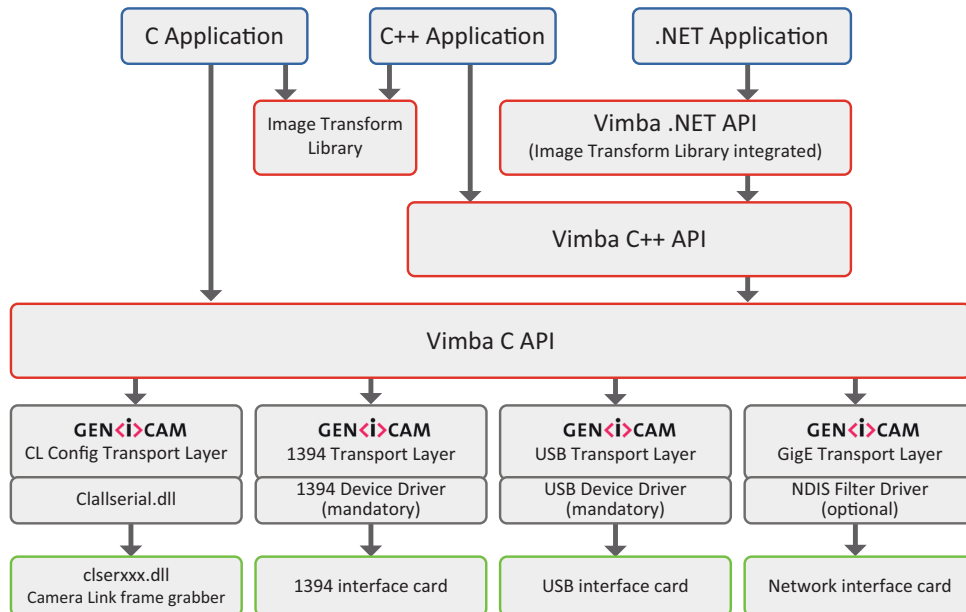


Figure 1: Vimba Architecture

Vimba provides four APIs:

- The **Python API** is ideal for quick prototyping. We also recommend this API for an easy start with machine vision or embedded vision applications. For best performance and deterministic behavior, the C and C++ APIs are a better choice.
- The **C API** is easy-to-use, but requires more lines of code than the Python API. It can also be used as API for C++ applications. The C API is also recommended to easily migrate from PvAPI to Vimba.
- The **C++ API** is designed as a highly efficient and sophisticated API for advanced object-oriented programming including shared pointers, the STL (Standard Template Library), and interface classes. If you prefer an API with less design patterns, we recommend the Vimba C API.
- The **.NET API** supports all .NET languages, for example, C#, C++/CLI, or Visual Basic .NET. Its general concept is similar to the C++ API.

All APIs cover the following functions:

- Listing currently connected cameras
- Controlling camera features
- Receiving images from the camera
- Notifications about camera connections or disconnections

The Image Transform Library converts camera images into other pixel formats and creates color images from raw images (debayering). While this is separated for the C and C++ API, the .NET API includes these functions. Therefore, a .NET application does not have to access the Image Transform Library.

The APIs use GenICam transport layer (GenTL) libraries to actually communicate with the cameras. These libraries (Vimba GigE TL, Vimba USB TL, and Vimba 1394 TL) can not be accessed directly through Vimba.

For more detailed information, read:

- [Vimba C Manual \(includes a function reference\)](#)
- [Vimba C++ Manual \(includes a function reference\)](#)
- [Vimba .NET Manual \(includes a function reference\)](#)
- [Vimba Python Manual \(for function documentation, see the docstrings\)](#)
- [Vimba Image Transform Manual](#)

3.3 API Entities Overview

This chapter provides a rough overview of Vimba's entities to explain their basic principles. The exact functionalities depend on the programming language.

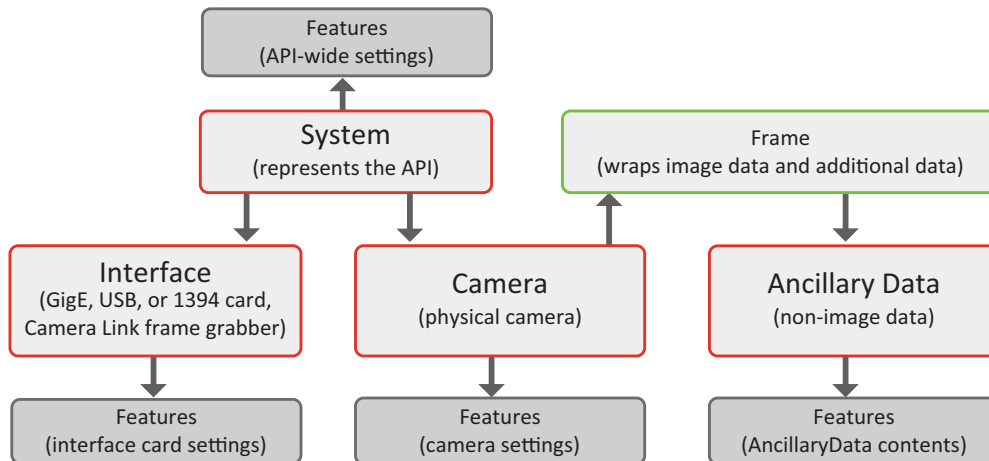


Figure 2: Vimba API Entities

All Vimba APIs use the same basic model for providing access to its entities. For object-oriented programming languages, this model is reflected in the class design, but even the C API supports this model by using handles as a representation of the different entities. The Python API uses *with* statements to access and handle the entities.

The **System** entity represents the API itself. Therefore, only one instance of it is available. The application has to initialize the System entity before any other function can be used. When the application has finished using the API, it shuts it down through the System entity. The System entity holds a list of interfaces and cameras internally and serves as the main access point to these entities.

A **Camera** entity controls a physical camera and receives images from the camera. Its functions are independent of the underlying interface technology.

An **Interface** entity represents a port on a physical interface card in the PC. Configuring the interface card is the main purpose of the Interface entity. The camera can directly be accessed via the System entity.

Frames contain image meta-data as well as references to the data that were sent by the camera (image and ancillary data). For use in Vimba, they must be created by the application and then be queued at the corresponding camera. When an image was received, the next available frame is filled and handed over to the application through a dedicated notification. After having processed the image data, the application should return the frame to the API by re-queueing it at the corresponding camera.

These Vimba entities can be controlled and set up via features:

The **System Features** contain information about API-wide settings, for example, which transport layer has been loaded.

The **Camera Features** configure camera settings such as the exposure time or the pixel format.

Interface Features represent the settings of a physical interface card in the PC, for example, the IP address of a network interface card.

Frames wrap image data and, if enabled, **AncillaryData** (e.g., camera settings at the time of acquisition), which may also be queried via feature access.

3.4 Features in Vimba

Within Vimba, settings and options are controlled by features. Many features come from the camera, which provides a self-describing XML file (for 1394 cameras, the XML file is created by the Vimba 1394 TL according to the camera register entries). Vimba can read and interpret the camera XML file. This means that Vimba is immediately ready-to-use with any Allied Vision camera. Even if the camera has a unique, vendor-specific feature, Vimba does not need to be updated to use this feature because it gets all necessary information from the XML file. Other features are part of Vimba's core and transport layers.

Vimba provides several feature types:

- Integer
- Float
- Enum
- String
- Command
- Boolean
- Raw data

Vimba's features are based on the GenICam industry standard. Therefore, Vimba enables using Allied Vision cameras with GenICam-based third-party software.

Further readings

- In-depth information about **GenICam** is available on the EMVA website:
<http://www.emva.org/standards-technology/genicam/>
- Allied Vision GigE camera features are described in the [GigE Features Reference](#).
- Allied Vision USB camera features are described in the [USB Features Reference](#).
- Allied Vision 1394 camera features are described in the [1394 Transport Layer Features Manual](#).
- Goldeye camera features are described in the Goldeye G/CL Features Reference:
<https://www.alliedvision.com/en/support/technical-documentation/goldeye-cl-documentation.html>

3.5 Vimba's Transport Layers

A transport layer (TL) transports the data from the camera to an application on the PC. Vimba contains GenICam transport layers (GenTLs) for Allied Vision GigE, USB and 1394 cameras.

The **Vimba GigE TL** optionally uses the Vimba filter driver, which provides a high performance with less CPU load. It is compatible with all Allied Vision GigE cameras.

The **Vimba USB TL** uses the (mandatory) Vimba USB device driver. It is compatible with all Allied Vision USB cameras.

The **Vimba 1394 TL** uses the (mandatory) high-performance intek driver, which replaces the low-performance Microsoft driver.

The **Vimba Camera Link Config TL** provides a Clallserial.dll, which connects with the Clserxxx.dll of GenICam-compliant frame grabbers.

Since Vimba's transport layers support GenICam, Allied Vision GigE, USB, 1394 and Camera Link cameras can easily be used with a GenICam-compliant third-party software.

For a feature list, see:

- [Vimba GigE TL Features Manual](#)
- [Vimba USB TL Features Manual](#)
- [Vimba 1394 TL Features Manual](#)
- [Vimba CL Config TL Features Manual](#)

3.6 Synchronous and asynchronous image acquisition

This chapter explains the principles of synchronous and asynchronous image acquisition. For details, please refer to the API manuals.

Note that the C++ API and the .NET API provide ready-made convenience functions for standard applications. These functions perform several procedures in just one step. However, for complex applications with special requirements, manual programming as described here is still required. The Python API handles many procedures within *with* statements.

Buffer management

Every image acquisition requires allocating memory and handling frame buffers. Except for the Python API, the following interaction between the user and the Vimba API is required:

User:

1. Allocate memory for the frame buffers on the host PC.
2. Announce the buffer (this hands the frame buffer over to the API).
3. Queue a frame (prepare buffer to be filled).

Vimba:

4. Vimba fills the buffer with an image from the camera.
5. Vimba returns the filled buffer (and hands it over to the user).

User:

6. Work with the image.
7. Requeue the frame to hand it over to the API.

Synchronous image acquisition

Synchronous image acquisition is simple, but does not allow reaching high frame rates. Its principle is to handle only one frame buffer and the corresponding image at a time, which is comparable to juggling with one ball.

Asynchronous image acquisition

Asynchronous image acquisition is comparable to juggling with several balls: While you work with an image, the next image is being acquired. Simplified said: the more images within a given time you want to work with, the more buffers you have to handle.

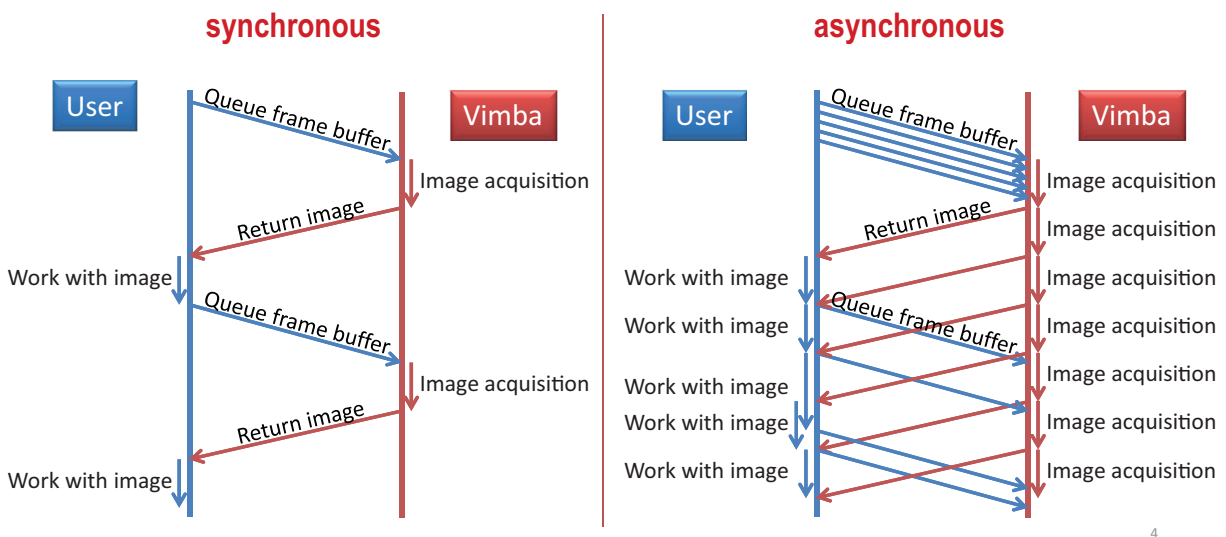


Figure 3: Acquisition Models

3.7 Notifications

In general, a vision system consisting of cameras and PCs is asynchronous, which means that certain events usually occur unexpectedly. This includes - among others - the detection of cameras connected to the PC or the reception of images. A Vimba application can react on a particular event by registering a corresponding handler function at the API, which in return will be called when the event occurs. The exact method how to register an event handler depends on the programming language. For details, use the code examples.



The registered functions are usually called from a different thread than the application. So extra care must be taken when accessing data shared between these threads (multithreading environment). Furthermore, the Vimba API might be blocked while the event handler is executed. Therefore, it is highly recommended to exit the event handler function as fast as possible.



Not all API functions may be called from the event handler function. For more details, see the API Manual for the programming language of your choice: [Vimba C Manual](#) , [Vimba C++ Manual](#), [Vimba .NET Manual](#), [Vimba Python Manual](#).

3.8 Building applications

3.8.1 Setting up Visual Studio for C and C++ projects



To ensure backward compatibility, Vimba examples are compatible with Visual Studio 15 and higher.

The easiest way to set up Visual Studio for C or C++ projects is using the property sheet Examples.props from the Vimba examples folder. The following description uses C++, but the principle can be applied to the C API as well. Users of the other APIs can use Visual Studio without any special preparations.

1. In Visual Studio, create a new project. Ignore the Application Wizard, just click **Finish**.
2. Insert this code into YourProjectName.cpp:


Listing 1: CPP code

```
#include "stdafx.h"
#include <iostream>

#include "VimbaCPP/Include/VimbaCPP.h"

using namespace AVT::VmbAPI;

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout << "Hello Vimba" << std::endl;
    VimbaSystem& sys = VimbaSystem::GetInstance();
    VmbVersionInfo_t version;
    if (VmbErrorSuccess == sys.QueryVersion(version))
    {
        std::cout << "Version:" << version.major << "." << version.minor << std::endl;
    }
    getchar();
    return 0;
}
```

3. Open the **Property Manager** window. In most Visual Studio editions, you can find it by clicking **View -> Other Windows -> Property Manager**.
4. In the Property Manager window, click the **Add Existing Property Sheet** icon . 
5. Go to the VimbaCPP_Examples folder. Unless you have changed the folder location during the Vimba installation, it is located at: C:\Users\Public\Documents\Allied Vision\Vimba_x.x. You need at least the Examples.props file, which is located in the Build\VS2010 folder. You can add the other PROPS files later as needed.

Now Visual Studio is set up and you can debug the solution.

4 Vimba Class Generator



This chapter includes:

4.1	Main window	23
4.2	C++ code generation	24
4.3	C# code generation	25

The Vimba Class Generator is a tool for easily creating classes for Vimba C++ and Vimba.NET APIs. The generated classes offer access functions for each found feature, depending on the type of the feature.



After a camera firmware update, regenerate the files and merge the access functions for new features manually into your previously generated code by copy & paste.

4.1 Main window

To generate classes, carry out the following steps (refer to the numbers in Figure 4):

1. Select the camera for which you would like to generate code
2. Choose the destination folder for the generated files
3. Select the programming language
4. Customize the code generation with several visible options and template files

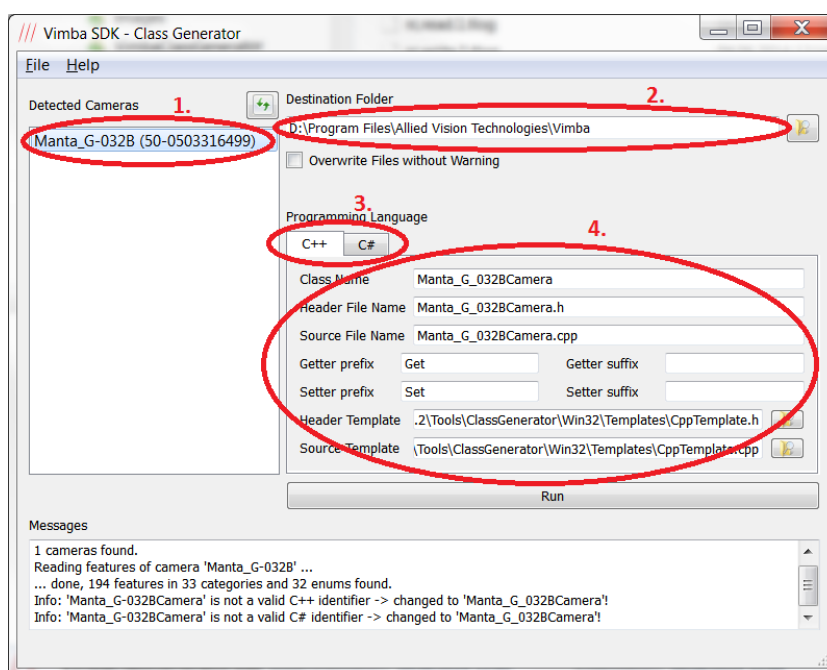


Figure 4: Vimba Class Generator - Main Window

The cameras detected during the program startup are listed in the *Detected cameras* box. Select the camera for which you want to obtain code.



To detect new cameras, click the Refresh button.

To change the default destination folder, click the button beside the text field *Destination Folder*. Alternatively, enter a path manually (make sure that it is valid).

By default, the Vimba Class Generator warns you before overwriting existing files. To change this behavior, select the checkbox *Overwrite Files without Warning*.

The options below *Programming Language* allow you to configure the code generation. To switch between programming languages, click the tab. For language-dependent options, see chapters C++ code generation and C# code generation.

If everything is configured, the *Run* button is enabled. Click it to generate the code for the selected camera, programming language, and options.

The *Messages* text box informs you, e.g., about changed camera names.

4.2 C++ code generation

In the *C++* tab of the main window, you have the following options:

- Class Name: The name of the generated class.
- Header File Name: The name of the header file to create.
- Source File Name: The name of the cpp file to create.
- Getter prefix: The text that is inserted before the feature name for each getter function.
- Getter suffix: The text that is added after the feature name for each getter function.
- Setter prefix: The text that is inserted before the feature name for each setter function.
- Setter suffix: The text that is added after the feature name for each setter function.
- Header template: The file that is used as a template for generating the header file.
- Source template: The file that is used as a template for generating the cpp file.

Templates for the header file and the cpp file are available in a subfolder below the class generator program. A template file for the header file contains the following hashtags that serve as placeholders:

- `### HEADER_FILE_MACRO_NAME ###`: Generated from the *Header File Name* in the main window.
- `### CLASS_NAME ###`: Corresponds to *Class Name* in the main window.
- `### ENUM_DECLARATIONS ###`: This is where the enum declarations are inserted.
- `### METHOD_DECLARATIONS ###`: This is where the method declarations are inserted.
- `### VARIABLE_DECLARATIONS ###`: This is where the variable declarations are inserted.

A template file for the cpp file may contain the following placeholders:

- `### HEADER_FILE_NAME ###`: Corresponds to *Header File Name* in the main window.
- `### CLASS_NAME ###`: Corresponds to *Class Name* in the main window.
- `### METHOD_IMPLEMENTATIONS ###`: This is where the method implementations are inserted.

In the template file, you can change the order of the variables to generate files that better suit your requirements.

4.3 C# code generation

In the *C#* tab of the main window, you have the following options:

- Namespace: The namespace of the generated class.
- Class Name: The name of the generated class.
- Class File Name: The name of the class file to create.
- Template File: The file that is used as a template for generating the class file.

A template for the source file to be generated is available in a subfolder below the class generator program. The template file may contain the following placeholders:

- `### NAMESPACE_NAME ###`: Generated from the *Namespace* in the main window.
- `### CLASS_NAME ###`: Corresponds to *Class Name* in the main window.
- `### PUBLIC_PROPERTIES ###`: This is where the generated properties are inserted.
- `### PUBLIC_METHODS ###`: This is where the generated methods are inserted.
- `### ENUM_DECLARATIONS ###`: This is where the enum declarations are inserted.

In the template file, you can change the order of the variables to generate a file that better suits your requirements.

5 Vimba Driver Installer



This chapter includes:

5.1	Main window	27
5.2	Changing drivers	28
5.3	Command line options	30

The Vimba Driver Installer is a tool to easily install and configure hardware devices or filter drivers. Although the same functionality can be achieved with the Windows device manager or through the network settings, the Vimba Driver Installer provides a more convenient way to select the correct driver for the Allied Vision software.

5.1 Main window

Upon startup, the Driver Installer examines the current hardware configuration, which may take a while. After the necessary information is collected, the main window is shown.

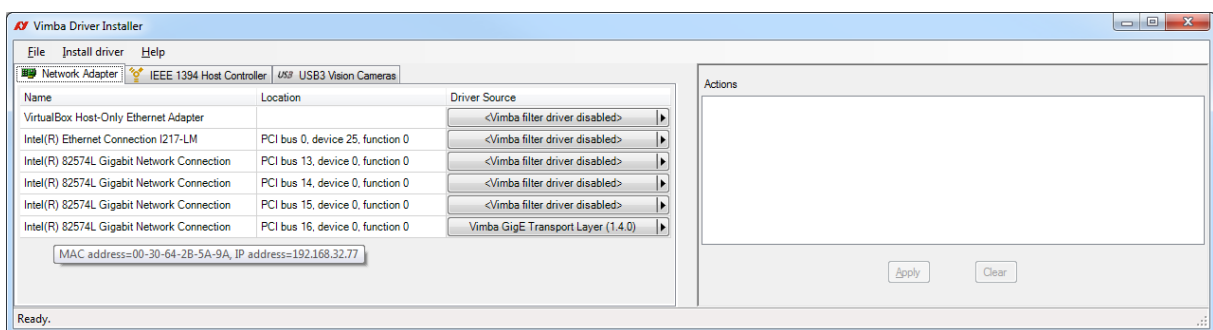


Figure 5: Driver Installer - Main Window

The main window lists:

- Installed network adapters
- Installed 1394 adapters
- Connected USB cameras

If your camera interface is not listed or the application closes with an error message, install a suitable network or 1394 adapter or connect your USB camera.

All adapters are listed with their *Name* and their physical *Location* in the PC.

- GigE: To easily identify a network adapter, its MAC address and IP address are shown in a tool tip.
- USB: The camera serial number is shown under *Location*.

The *Driver Source* column displays where the currently installed driver comes from (e.g., the Allied Vision SDK it was installed with).

The *Actions* panel shows the actions to be performed by the Driver Installer if the *Apply* button is clicked. It is initially empty.

5.2 Changing drivers

Changing individual driver settings does not take effect instantly. Instead, the changes are queued as *Actions* in the panel on the right side of the screen. They are executed when the *Apply* button is clicked.

To change the driver for host adapters or cameras, select its tab (e.g., *Network Adapter* or *USB3 Vision Cameras*) whose driver you want to install or change. This can be done by three different methods:

- Click the button in the column *Driver Source* of the corresponding host adapter or camera and select an entry from the popup menu, or
- Right-click anywhere in the row of the corresponding host adapter or camera and select an entry from the popup menu, or
- Select a host adapter or a camera by clicking in the corresponding row and choose an entry from the *Install Driver* menu.



The Driver Installer supports multi-selection of adapters or cameras: On your keyboard, press *Ctrl + A* or hold down the *Shift* or *Control* key while clicking.

The list of products to choose from is determined at startup by searching the PC for currently installed Allied Vision SDKs. Additionally, one item is added to each list that can be used to remove the Allied Vision driver from the adapter or camera: *<disable Allied Vision filter driver>* for network adapters, *<remove Allied Vision driver>* for USB cameras, and *Microsoft* for 1394 adapters.

After selecting a new driver source, one or more actions to be performed by the Driver Installer are added to the right panel. The corresponding adapters or cameras are disabled, so you can not add a second action for the same adapter or camera.

To undo your selections, click the *Clear* button. This will empty the list of actions so that a new product can be selected.

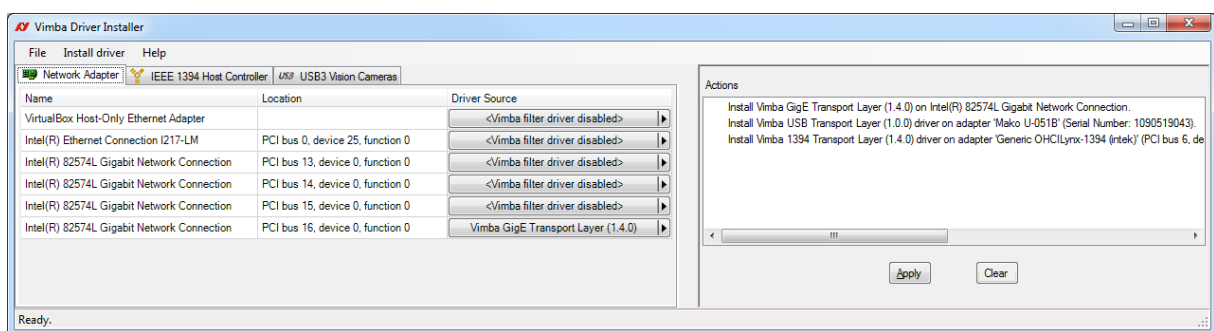


Figure 6: Driver Installer - Prepare actions

Incompatibility message Some Allied Vision drivers are not compatible with all installed Allied Vision SDKs. When this is detected, the following error message is shown:

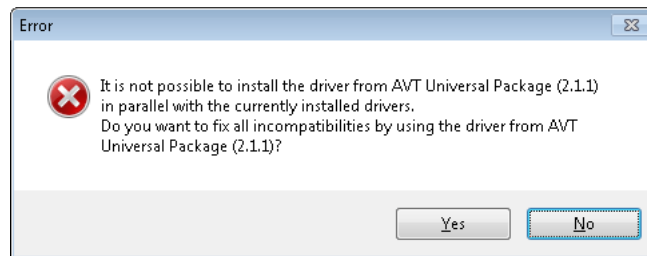


Figure 7: Driver Installer - Incompatible products detected

If you click *Yes*, additional actions will be added to install the driver of the new SDK on the other adapters, too. If you click *No*, the action will be removed.

To start the actual installation, click the *Apply* button. During the installation, which may take a while, the operating system might bring up other windows asking for permission to install the driver. The result of each action is displayed by a small icon in front of the action.

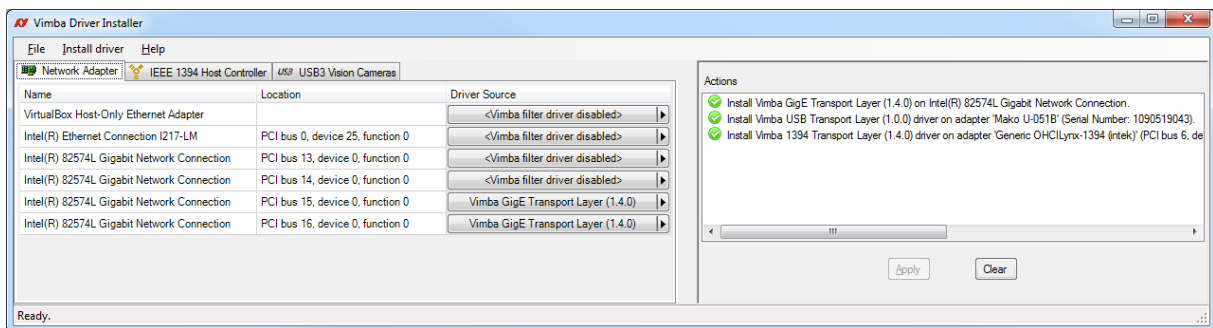


Figure 8: Driver Installer - Installation successful

5.3 Command line options

The Driver Installer can either be used in interactive mode or perform a single task without showing a graphical user interface (GUI). The syntax for starting the application is:

```
VimbaDriverInstaller.exe [command] [options]
```

The following commands are available:

Command	Description	Mandatory options
help	Displays infos about the usage of the Driver Installer	<i>none</i>
info	Displays infos about installed Allied Vision products and hardware.	<i>none</i>
add	Adds a driver of an Allied Vision product to the driver store.	<i>/product</i>
remove	Removes a driver of an Allied Vision product from the driver store.	<i>/product</i>
exists	Checks whether the given Allied Vision product is installed. The function returns 0 if product was found, otherwise see return code table.	<i>/product</i>
install	Installs a driver of an Allied Vision product on the given adapters or cameras.	<i>/product and /deviceType</i>

The following options can be used:

Option	Description
/l filename	Log trace messages into the given file.
/product upgradeCode	Identifies a product by its upgrade code (example: 'd108d42f-c1d4-4d86-aa1f-e3ec4a137aaf').
/product productNameAndVersion	Identifies a product by its name and version (example: 'Vimba (1.4.0)').
/deviceType deviceTypeName	Identifies the hardware devices by their device class (either 'net' for network adapter, 'USB' for USB cameras or '1394' for 1394 adapter).
/force	Force removal of a product even if it is in use. A reboot might be necessary when this option is used. This option is valid for the "remove" command only.
/hidden	Suppress output to console window. Normally, when the Driver Installer is run without a GUI, it creates a console window to display its output. This can be disabled by using this option.
/adjustMaxFilterDriverNums	Windows operating systems limit the number of network filter drivers that can be installed on a system at the same time. Starting with Windows Vista, this number is configurable. Use this option if you want to adjust the number automatically to ensure that the Vimba GigE Filter Driver can be installed. This option is valid only when the "install" command is used together with a network device.

The options */product* and */deviceType* can be specified multiple times.



Use the *info* command to find out the upgrade codes or exact names and versions for installed Allied Vision products.

If started without parameters, the application is displayed in interactive mode with GUI.

The following table lists the return code when started in silent mode:

Return code	Description
1	Success, reboot required.
0	Success.
-1	Unknown error.
-2	Unknown command.
-10	Given device not found.
-11	At least one device must be specified for this command.
-12	Exactly one device must be specified for this command.
-13	Given device cannot be modified, because it is in use.
-20	Given product not found.
-21	At least one product must be specified for this command.
-22	Exactly one product must be specified for this command.
-23	Given product cannot be removed from the driver store, because it is in use.
-24	Given product is not found in the driver store.
-25	Given product is not installed on the given device.
-1001	The network configuration system is locked by another application. Terminate all other applications and run the Driver Installer again.
-1002	The maximum number of installed network filter drivers is reached. Use the "adjustMaxFilterDriverNums" option or remove a network filter driver by uninstalling the corresponding application.

Examples:

- `VimbaDriverInstaller info`
Displays infos about installed Allied Vision products and hardware.
- `VimbaDriverInstaller remove /product "Vimba (1.4.0)"`
Removes all drivers of Vimba 1.4 from the driver store.
- `VimbaDriverInstaller install /product d108d42f-c1d4-4d86-aa1f-e3ec4a137aaf /deviceType net /deviceType USB /deviceType 1394`
Install drivers of Vimba 1.4 on all network, USB, and 1394 devices. Here the product is given by its upgrade code.

6 Vimba Firmware Updater



This chapter includes:

6.1	Uploading firmware	34
6.2	Aborting a firmware upload	35
6.3	Troubleshooting	36
6.4	Command line Firmware Updater	36

The Vimba Firmware Updater supports firmware uploads to Allied Vision USB cameras and Goldeye CL cameras. New firmware for each connected camera is automatically detected and selected. You can update several cameras in one step. Uploading older firmware is also possible.

If you prefer to upload firmware via command line, see chapter Command line Firmware Updater.



Download the latest USB firmware from our website:
<https://www.alliedvision.com/en/support/firmware.html>.

6.1 Uploading firmware



Before changing the firmware of your USB camera, install the Allied Vision USB 3 driver for your camera (see chapter Vimba Driver Installer).

To upload new firmware to your cameras, carry out the following steps (see Figure 9):

1. Connect your Allied Vision cameras and start Vimba Firmware Updater.
2. Click **Open** and select a firmware container file.
Optional: Click **Info** to get details about the selected firmware.
3. Click **Update cameras** to upload the automatically selected firmware to your cameras.

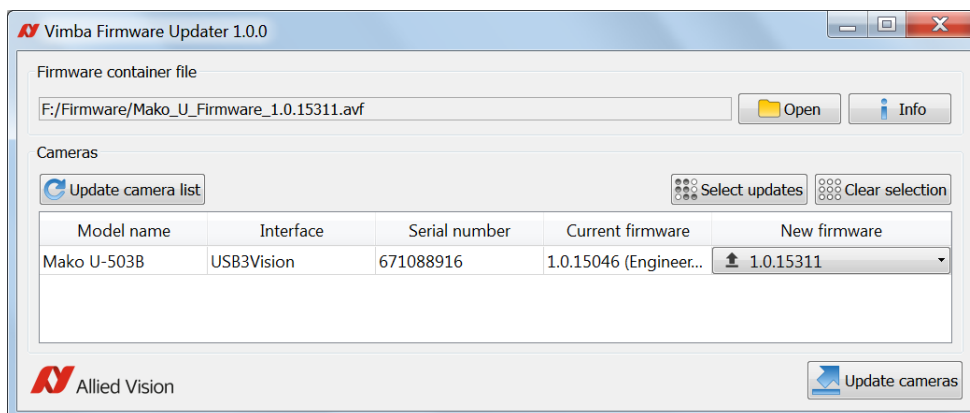


Figure 9: Firmware Updater - Main Window

To **manually select the updates**, click the drop-down field:

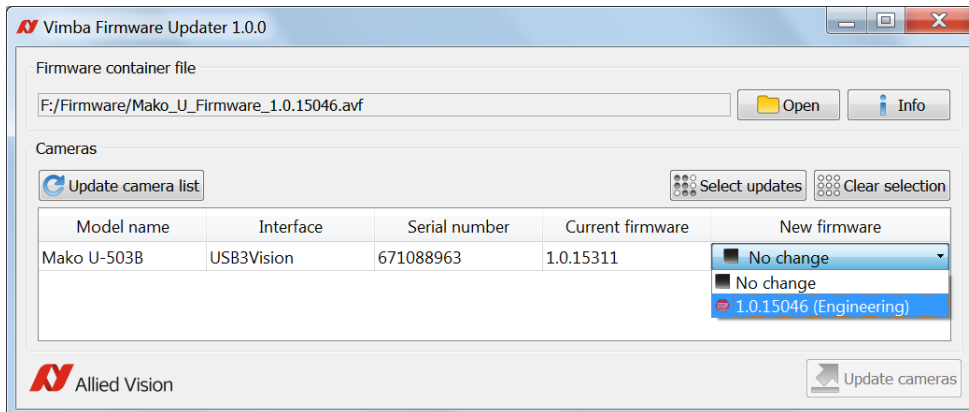


Figure 10: Firmware Updater - Manual Update

Update cameras becomes active as soon as a firmware is chosen.

Optionally, you can use the buttons **Select updates** and **Clear selection**, which switch on/off the automatic selection of firmware with higher versions than the firmware on the cameras.

6.2 Aborting a firmware upload

The firmware upload to several cameras takes some time. During the upload, the **Abort** button finishes the upload to the current camera, but does not upload firmware to the next models.

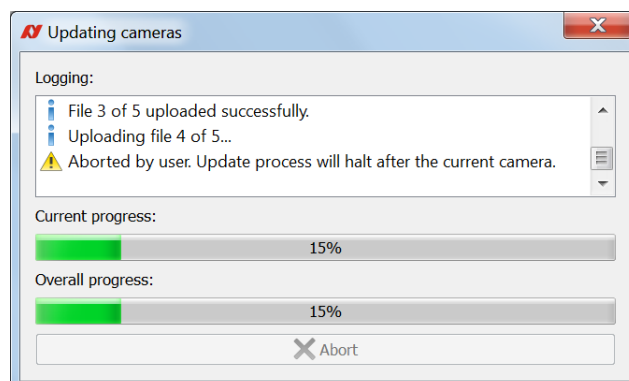


Figure 11: Firmware Updater - Abort

6.3 Troubleshooting

If your camera is not detected or the firmware cannot be updated:

- Make sure no other application uses the camera.
- Restart the PC.
- Start the firmware upload again. Check if the camera works with Vimba Viewer. If not, start the command line Firmware Updater and use repair mode or contact our support team:
<https://www.alliedvision.com/support>
- With USB cameras, start Vimba Driver Installer and make sure the Vimba USB driver is in use.
- If you connected your USB camera to a hub, unplug the hub from the PC and disconnect its power supply. Reconnect it and try again.
- Connect your USB camera to a different USB 3.0 input or a different hub.

6.4 Command line Firmware Updater

To update firmware via command line, use FWUpdaterConsole.exe. This tool provides two main functionalities:

- "--show" or "-s" displays information about camera firmware or a firmware file.
- "--write" or "-w" performs the actual update.

See the following list of use cases:

Use case	Parameters
Show device info for list of cameras	<code>--show --device "list of ids"</code>
Show a list of matching firmware sets for all cameras	<code>--show --container "file" --device all</code>
Show detailed info about matching firmware sets for one camera	<code>--show --container "file" --device "id"</code>
Show firmware set info for one set	<code>--show --container "file" --index "index"</code>
Show firmware set info for list of sets	<code>--show --container "file" --index "list of indices"</code>
Show firmware set info for whole container	<code>--show --container "file"</code>
Write one firmware set to one camera	<code>--write --container "file" --device "id" --index "index"</code>
Write 'best' (latest) firmware set to list of cameras	<code>--write --container "file" --device "list of ids"</code>
Write 'best' (latest) firmware set to all cameras	<code>--write --container "file" --device all</code>
Write one firmware set to one camera in repair mode	<code>--write --repair --container "file" --device "id" --index "index"</code>

Table 1: Use cases for the command line Firmware Updater

The following options may be added to the "show" or "the "write" functionality:

Option	Parameters
Show full information	<code>--verbose, -v</code>
Force writing	<code>--force, -f</code>
Repair device firmware during write	<code>--repair, -r</code>

Table 2: Command options for the firmware update



By calling `FWUpdaterConsole --help [command/option]`, you get more details about the tool or its parameters.

7 Vimba Deployment



This chapter includes:

7.1	Modules	39
7.2	Examples	40

7.1 Modules

You can use the Vimba setup executable to manually install the software on the target machine or run the installer silently. The required Microsoft Installer (MSI) files are contained in the setup executable, it is not necessary to download them separately. To extract the MSIs from the setup executable, run:

```
Vimba.exe /extract.
```

Please note that this command will neither install anything nor will it affect a current installation on the machine. After selecting the destination folder, the following MSI files are created:

Name	Description
VimbaGigETL_Win32.msi	Vimba GigE Vision Transport Layer for 32-bit Windows operating systems.
VimbaGigETL_Win64.msi	Vimba GigE Vision Transport Layer for 64-bit Windows operating systems.
VimbaUSBTL_Win32.msi	Vimba USB3 Vision Transport Layer for 32-bit Windows operating systems.
VimbaUSBTL_Win64.msi	Vimba USB3 Vision Transport Layer for 64-bit Windows operating systems.
Vimba1394TL_Win32.msi	Vimba 1394 Transport Layer for 32-bit Windows operating systems.
Vimba1394TL_Win64.msi	Vimba 1394 Transport Layer for 64-bit Windows operating systems.
VimbaCLConfigTL_Win32.msi	Vimba CL Config Transport Layer for 32-bit Windows operating systems.
VimbaCLConfigTL_Win64.msi	Vimba CL Config Transport Layer for 64-bit Windows operating systems.
Vimba_Win32.msi	Vimba for 32-bit Windows operating systems.
Vimba_Win64.msi	Vimba for 64-bit Windows operating systems.

If you want to install Vimba for an application using the GenICam Transport Layer interface, it is sufficient to install one or more Vimba transport layer modules. If your application uses one of the Vimba APIs, it is necessary to additionally install Vimba.

7.2 Examples

Microsoft provides a tool named *msiexec* to install an MSI file. To install Vimba's core components into the default programs folder, apply the following command lines:

Install the Vimba GigE Transport Layer module:

```
msiexec -i VimbaGigETL_Win32.msi  
ADDLOCAL="VCRedist100,GigETL_RegisterGenICamPathVariable,GigETL_Core"
```

Install the Vimba USB Transport Layer module:

```
msiexec -i VimbaUSBTL_Win32.msi  
ADDLOCAL="VCRedist100,USBTL_RegisterGenICamPathVariable,USBTL_Core"
```

Install the Vimba 1394 Transport Layer module:

```
msiexec -i Vimba1394TL_Win32.msi  
ADDLOCAL="VCRedist100,FireWireTL_RegisterGenICamPathVariable,FireWireTL_Core"
```

Install the Vimba CL Config Transport Layer module:

```
msiexec -i VimbaCLConfigTL_Win32.msi  
ADDLOCAL="VCRedist100,CLConfigTL_RegisterGenICamPathVariable,CLConfigTL_Core"
```

The installation of Vimba depends on the API. Install the Vimba C API:

```
msiexec -i Vimba_Win32.msi ADDLOCAL="VCRedist90,VCRedist100,  
VCRedist100_MFC,Vimba_Core,VimbaC_Core,VimbaImageTransform_Core,Vimba_DrvInst"
```

Install the Vimba C++ API:

```
msiexec -i Vimba_Win32.msi ADDLOCAL="VCRedist90,VCRedist100,  
VCRedist100_MFC,Vimba_Core,VimbaCPP_Core,VimbaImageTransform_Core,Vimba_DrvInst"
```

Install the Vimba .NET API:

```
msiexec -i Vimba_Win32.msi ADDLOCAL="VCRedist90,VCRedist100,  
VCRedist100_MFC,Vimba_Core,VimbaNET_Core,VimbaImageTransform_Core,Vimba_DrvInst"
```

Other useful parameters for *msiexec*:

INSTALLDIR="*path to desired installation directory*"

ALLUSERS="1" Performs a machine-wide installation. If omitted, a per user installation is performed.



The installation of the Vimba Transport Layer modules does not automatically install the corresponding hardware drivers. This has to be done afterwards by calling the *VimbaDriverInstaller* (see chapter Command line options for further reference).



The installation of the Camera Link Config Transport Layer module automatically installs the Clallserial.dll.

8 Compiling the Vimba C++ API

If *C++ API development components* are installed, the Vimba C++ API source code along with Microsoft Visual Studio solution files for version 2005, 2008, and 2010 can be found in the examples folder, sub-folder:

VimbaCPP_Source

With the installed components,

- You can find an example of using the C API
- You may build your own C++ library with a different compiler
- You can build a customized C++ API with your own shared pointer implementation. For more information on how to use shared pointers in the API, refer to the corresponding chapter in the [Vimba C++ Manual](#).



Please note that Allied Vision can offer only limited support if an application uses a modified version of the Vimba C++ API.

9 Vimba - Feature Overview

Vimba provides additional functionality that is not directly covered by API functions with GenICam Features. These Features can only be accessed via certain entities within Vimba. According to the API Entity Model described in chapter API Entities Overview, the entities providing Feature access are:

- The **Vimba System**, which includes functionality for managing interfaces and cameras.
- The **Interface**, which allows configuration of hardware interfaces (e.g. a GigE port).
- The **Camera**, which allows access to all features provided by camera device, data transport features, and some driver features.
- The **AncillaryData** for each Frame.

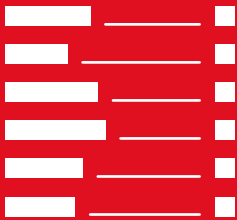
Features are described in the following documents:

- Vimba System features are described in chapter ?? in this document.
- GigE, USB, 1394, or Camera Link Interface features are handled by the Transport Layer, see chapter "Interface Features" in the [Vimba GigE TL Features Manual](#), the [Vimba USB TL Features Manual](#), the [Vimba 1394 TL Features Manual](#) and the [Vimba CL Config TL Features Manual](#).
- Camera features for **GigE or USB cameras** are listed in the [GigE Features Reference](#) or [USB Features Reference](#). **1394 camera** features are listed in the [Vimba 1394 TL Manual](#). See chapters "Camera Features", "Device Features" and "DataStream Features".
- Ancillary Data features are described in chapter Ancillary Data Features in this document.



For the latest version of GigE or USB camera features, download the corresponding Features Reference manual:
<https://www.alliedvision.com/en/support/technical-documentation.html>.

10 Vimba System



This chapter includes:

10.1	Info [Allied Vision]	44
10.1.1	Elapsed [Allied Vision]	44
10.1.2	GeVTLLsPresent [Allied Vision]	44
10.1.3	FiWTLLsPresent [Allied Vision]	45
10.1.4	UsbTLLsPresent [Allied Vision]	45
10.1.5	CLTLLsPresent [Allied Vision]	45
10.2	Discovery [Allied Vision]	45
10.2.1	GeVDiscoveryAllOff [Allied Vision]	46
10.2.2	GeVDiscoveryAllAuto [Allied Vision]	47
10.2.3	GeVDiscoveryAllOnce [Allied Vision]	47
10.2.4	GeVDiscoveryStatus [Allied Vision]	47
10.2.5	GeVDiscoveryAllDuration [Allied Vision]	48
10.2.6	DiscoveryCameraIdent [Allied Vision]	48
10.2.7	DiscoveryCameraEvent [Allied Vision]	48
10.2.8	DiscoveryInterfaceIdent [Allied Vision]	49
10.2.9	DiscoveryInterfaceEvent [Allied Vision]	49
10.3	ForceIP [Allied Vision]	49
10.3.1	GevDeviceForceIP [Allied Vision]	50
10.3.2	GevDeviceForceMACAddress [Allied Vision]	50
10.3.3	GevDeviceForceIPAddress [Allied Vision]	50
10.3.4	GevDeviceForceSubnetMask [Allied Vision]	51
10.3.5	GevDeviceForceGateway [Allied Vision]	51
10.4	ActionControl [Allied Vision]	51
10.4.1	ActionCommand [Allied Vision]	51
10.4.2	ActionDeviceKey [Allied Vision]	52
10.4.3	ActionGroupKey [Allied Vision]	52
10.4.4	ActionGroupMask [Allied Vision]	52
10.4.5	GevActionDestinationIPAddress [Allied Vision]	53

This chapter lists features that are potentially available in this module. Some features are only available under certain circumstances.

The following categories can be found below the Root category:

- Info
- Discovery
- ForcelP
- ActionControl

10.1 Info [Allied Vision]

10.1.1 Elapsed [Allied Vision]

Name	Elapsed
Interface	IFloat
Access	Read
Visibility	Beginner
Values	0.0..

Elapsed time since the API was initialized.

10.1.2 GeVTLIsPresent [Allied Vision]

Name	GeV TL Is Present
Interface	IBoolean
Access	Read
Visibility	Beginner

The GigE Vision Transport Layer is present and working.

10.1.3 FiWTLIsPresent [Allied Vision]

Name	FiW TL Is Present
Interface	IBoolean
Access	Read
Visibility	Beginner

The FireWire Transport Layer is present and working.

10.1.4 UsbTLIsPresent [Allied Vision]

Name	Usb TL Is Present
Interface	IBoolean
Access	Read
Visibility	Beginner

The USB Transport Layer is present and working.

10.1.5 CLTLIsPresent [Allied Vision]

Name	CL TL is present
Interface	IBoolean
Access	Read
Visibility	Beginner

The Camera Link Transport Layer is present and working.

10.2 Discovery [Allied Vision]

This category contains **features for camera and interface discovery** with Vimba, for example:

- Camera availability

- Notifications about camera availability
- Discovery process for GigE devices



The description below applies to the C API. For more information, see [Vimba C Manual](#), [Vimba CPP Manual](#), or [Vimba .NET Manual](#).

Discovery of GigE cameras

The discovery process of GigE cameras usually takes some time, especially if multiple cameras are connected. Many applications open only one camera directly By its ID, IP address or MAC address. Consequently, Vimba initially does not discover devices automatically.

- *GeVDiscoveryAllOnce* starts the discovery once to get a complete camera list.
- *GeVDiscoveryAllAuto* detects GigE cameras permanently, which consumes a considerable amount of bandwidth.
- Both commands wait for *GeVDiscoveryDuration* milliseconds before returning. This allows you to directly get the list of cameras afterwards.
- *GeVDiscoveryAllOff* stops automatic discovery.

Notifications

Notifications about camera discovery and interface discovery work with the same mechanism:

- *DiscoveryCameraEvent* notifies about changes to the overall camera list and changes of the accessibility status of the cameras. During a notification, querying *DiscoveryCameraIdent* returns the camera change that caused the notification.
- *DiscoveryInterfaceEvent* notifies about interface-related changes, and querying *DiscoveryInterfaceIdent* returns the interface identifier.



For more information, see chapter Using Event in the API manuals.

10.2.1 GeVDiscoveryAllOff [Allied Vision]

Name	GeV Discovery All Off
Interface	ICommand
Access	Read/Write
Visibility	Beginner

Turns devices discovery OFF for all GigE interfaces.

10.2.2 GeVDiscoveryAllAuto [Allied Vision]

Name	GeV Discovery All Auto
Interface	ICommand
Access	Read/Write
Visibility	Beginner

Turns devices discovery ON for all GigE interfaces.

10.2.3 GeVDiscoveryAllOnce [Allied Vision]

Name	GeV Discovery All Once
Interface	ICommand
Access	Read/Write
Visibility	Beginner

Turns devices discovery temporary ON for all GigE interfaces.

10.2.4 GeVDiscoveryStatus [Allied Vision]

Name	GeV Discovery Status
Interface	IEnumeration
Access	Read
Visibility	Beginner
Values	AllOff, AllAuto, AllOnce

Provides state of discovery for GigE interfaces.

Possible values:

- AllOff: Discovery is OFF for all GigE interfaces.
- AllAuto: Discovery is ON for all GigE interfaces.
- AllOnce: Discovery is temporary ON for all GigE interfaces.

10.2.5 GeVDiscoveryAllDuration [Allied Vision]

Name	GeV Discovery Duration
Interface	Integer
Access	Read/Write
Visibility	Beginner

The time in ms to wait for response from any device after device discovery was started in mode "Once" or "Auto".

Defaults to 150 ms.

10.2.6 DiscoveryCameraIdent [Allied Vision]

Name	Discovery Camera Ident
Interface	String
Access	Read/Write
Visibility	Beginner

Identifier of the camera that triggered the last camera discovery event.

10.2.7 DiscoveryCameraEvent [Allied Vision]

Name	Discovery Camera Event
Interface	Enumeration
Access	Read/Write
Visibility	Beginner
Values	Missing, Detected, Reachable, Unreachable

Indicates the last camera discovery event.

Possible values:

- Missing: The camera is missing.
- Detected: The camera was detected.
- Reachable: The camera is reachable (can be talked to).
- Unreachable: The camera is unreachable (cannot be talked to).

10.2.8 DiscoveryInterfaceIdent [Allied Vision]

Name	Discovery Interface Ident
Interface	IString
Access	Read/Write
Visibility	Beginner

Identifier of the interface that triggered the last interface discovery event.

10.2.9 DiscoveryInterfaceEvent [Allied Vision]

Name	Discovery Interface Event
Interface	IEnumeration
Access	Read/Write
Visibility	Beginner
Values	Unavailable, Available

Indicates the last interface discovery event.

10.3 ForceIP [Allied Vision]

This category contains features to force port features of a camera that would otherwise be inaccessible via Vimba.

1. Set the MAC address of the used camera in feature *GeVForceIPAddressMAC*
2. Set the required values of *GeVForceIPAddressIP*, *GeVForceIPAddressSubnetMask*, or *GeVForceIPAddressGateway*
3. To send these values to the camera, run *GeVForceIPAddressSend*.

10.3.1 GevDeviceForceIP [Allied Vision]

Name	Send camera force address
Interface	ICommand
Access	Read/Write
Visibility	Expert

Send the force address command on all interfaces

10.3.2 GevDeviceForceMACAddress [Allied Vision]

Name	Camera MAC Address
Interface	Integer
Access	Read/Write
Visibility	Expert

48-bit MAC address of the camera to force IP setup

10.3.3 GevDeviceForceIPAddress [Allied Vision]

Name	Camera's desired IP Address
Interface	Integer
Access	Read/Write
Visibility	Expert

IP address of the camera to be forced to

10.3.4 GevDeviceForceSubnetMask [Allied Vision]

Name	Camera's desired subnet mask
Interface	Integer
Access	Read/Write
Visibility	Expert

Subnet mask of the camera to be forced to

10.3.5 GevDeviceForceGateway [Allied Vision]

Name	Camera's desired gateway
Interface	Integer
Access	Read/Write
Visibility	Expert

Gateway of the camera to be forced to

10.4 ActionControl [Allied Vision]

10.4.1 ActionCommand [Allied Vision]

Name	Action Command
Interface	ICommand
Access	Read/Write
Visibility	Expert

Send created Action Command.

10.4.2 ActionDeviceKey [Allied Vision]

Name	Action Device Key
Interface	Integer
Access	Read/Write
Visibility	Expert

The Device Key for the Action Command to be created.
This Key has to match Action Device Key within desired device(s).

10.4.3 ActionGroupKey [Allied Vision]

Name	Action Group Key
Interface	Integer
Access	Read/Write
Visibility	Expert

The Group Key for the Action Command to be created.
This Key has to match Action Group Key within desired device(s).

10.4.4 ActionGroupMask [Allied Vision]

Name	Action Group Mask
Interface	Integer
Access	Read/Write
Visibility	Expert

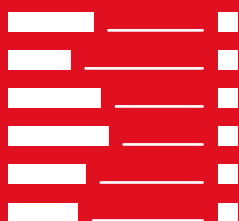
The Group Mask Key for the Action Command to be created.
This Key has to match Action Group Mask Key within desired device(s).

10.4.5 GevActionDestinationIPAddress [Allied Vision]

Name	Gev Action Destination IP Address
Interface	Integer
Access	Read/Write
Visibility	Expert

Specifies the destination IP address for the Action Command.

11 Ancillary Data Features



This chapter includes:

11.1	ChunkData [Allied Vision]	55
11.1.1	ChunkAcquisitionFrameCount [Allied Vision]	55
11.1.2	ChunkUserValue [Allied Vision]	55
11.1.3	ChunkExposureTime [Allied Vision]	56
11.1.4	ChunkGain [Allied Vision]	56
11.1.5	ChunkSyncInLevels [Allied Vision]	56
11.1.6	ChunkSyncOutLevels [Allied Vision]	57

This chapter lists the available features for Ancillary Data.

The following categories can be found below the Root category:

- ChunkData

11.1 ChunkData [Allied Vision]

Ancillary Data are non-image data that are part of the camera transfers. It relates to GenICam's Chunk Data.

Allied Vision GigE cameras usually don't expose the layout of their Ancillary Data via camera features, but the layout is the same for all cameras. Instead, they only provide feature *ChunkModeActive*, which is disabled by default. To enable transfer of Ancillary Data, set *ChunkModeActive* to "True".

11.1.1 ChunkAcquisitionFrameCount [Allied Vision]

Name	Chunk Acquisition Frame Count
Interface	Integer
Access	Read
Visibility	Beginner

This is the number of the frame during the current acquisition.

11.1.2 ChunkUserValue [Allied Vision]

Name	Chunk User Value
Interface	Integer
Access	Read
Visibility	Beginner

User value

11.1.3 ChunkExposureTime [Allied Vision]

Name	Chunk Exposure Time
Interface	IFloat
Access	Read
Visibility	Beginner

Exposure duration, in microseconds.

11.1.4 ChunkGain [Allied Vision]

Name	Chunk Gain
Interface	IFloat
Access	Read/Write
Visibility	Beginner

Gain value of analog A/D stage.

Units are usually in dB.

11.1.5 ChunkSyncInLevels [Allied Vision]

Name	Chunk Sync In Levels
Interface	Integer
Access	Read/Write
Visibility	Beginner

Momentary logic levels of the hardware line inputs.

11.1.6 ChunkSyncOutLevels [Allied Vision]

Name	Chunk Sync Out Levels
Interface	Integer
Access	Read/Write
Visibility	Beginner

Output levels of hardware sync outputs, for output(s) in GPO mode.

12 Vimba Cognex Adapter

Since Vimba 1.2, Vimba Cognex Adapter supports Allied Vision GigE, USB, and 1394 cameras. For a detailed description, see the [Vimba Cognex Adapter Manual](#).

13 References

This section lists documents with more detailed information about the components of Vimba. Please note that the links are valid only if the corresponding component has been installed.

Allied Vision Vimba GigE Transport Layer and cameras:

- [Vimba GigE TL Features Manual](#)
- [GigE Features Reference](#)

Allied Vision Vimba USB Transport Layer and cameras:

- [Vimba USB TL Features Manual](#)
- [USB Features Reference](#)

Allied Vision Vimba 1394 Transport Layer and cameras:

- [Vimba 1394 TL Features Manual](#)

Allied Vision Vimba CL Config Transport Layer and cameras:

- [Vimba CL Config TL Features Manual](#)
- [Goldeye Features Reference](#)

Vimba Image Transform Library:

- [Vimba Image Transform Manual](#)

Vimba C API:

- [Vimba C Manual](#)

Vimba C++ API:

- [Vimba C++ Manual](#)

Vimba .NET API:

- [Vimba .NET Manual](#)

Vimba Cognex Adapter:

- [Vimba Cognex Adapter Manual](#)