AVT SOFTWARE DEVELOPMENT KIT

AVT Vimba

User Guide for Linux

V1.3
2014-07-09

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

Contents

List of Figures

1 Contacting Allied Vision Technologies

Note

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

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

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

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

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

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

2 Introduction

2.1 Document history

Version	Date	Changes
1.0	2013-04-03	Initial version
1.1	2013-05-13	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

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
Bold	Programs, inputs or highlighting important things	**bold**
Courier	Code listings etc.	`Input`
Upper case	Constants	CONSTANT
Italics	Modes, fields, features	*Mode*
Blue and/or parentheses	Links	(Link)

2.2.2 Symbols

Note

This symbol highlights important information.

Caution

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

www

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

3 Vimba SDK Overview

Vimba for Linux is an SDK for all AVT GigE cameras, which is designed to be compatible with future AVT cameras connected to other hardware interfaces. Since Vimba is based on GenICam, common third-party software solutions can easily be supported.

As of version 1.3, Vimba's image transform library runs with OpenMP.

Supported cameras:

- AVT GigE cameras

Tested PC operating systems:

- Linux Ubuntu (Tested with version 12.04 LTS "Precise Pangolin")
- Linux Debian (Tested with version 6 "Squeeze")
- Linux Fedora (Tested with version 17 "Beefy Miracle")

ARM configurations:

	Hard-float	Soft-float
Processor	ARMv7-compatible processor with VFP 3 support and Thumb extension	ARMv7-compatible processor
Tested OS	Ubuntu 13.10	Debian armel (Debian 6) Angstrom 2012.12[1]

Tested hardware/boards: PandaBoard, ODROID-XU

Vimba and third-party software

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

[1]If running soft-float applications on hard-float boards, it might become necessary to install additional runtime libraries (e.g. for soft-float support).

3.1 Architecture

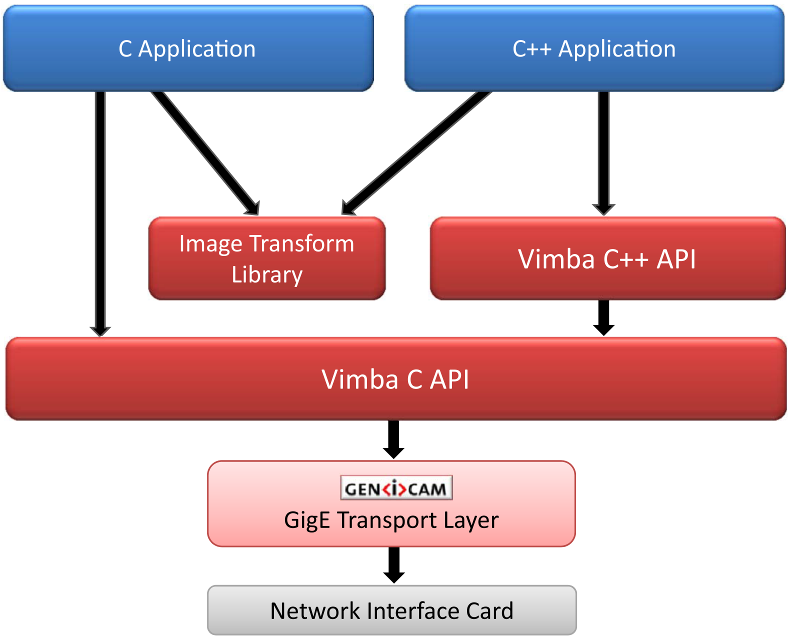

Figure 1: Vimba Architecture

Vimba provides two APIs:

- The C API is Vimba's easy-to-use basic API. It can also be used as an API for C++ applications.
- The C++ API is a sophisticated API with clear object-orientation, an elaborate class architecture, and consequent use of shared pointers.

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).

The APIs use GenICam transport layer (GenTL) libraries to actually communicate with the cameras. These libraries (currently only AVT GigE TL) can not be accessed directly through Vimba.

For more detailed information, please refer to the following documents:

- Vimba C Programmer's Manual and Function Reference
- Vimba C++ Programmer's Manual and Function Reference
- Vimba Image Transform Library Manual

3.2 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.

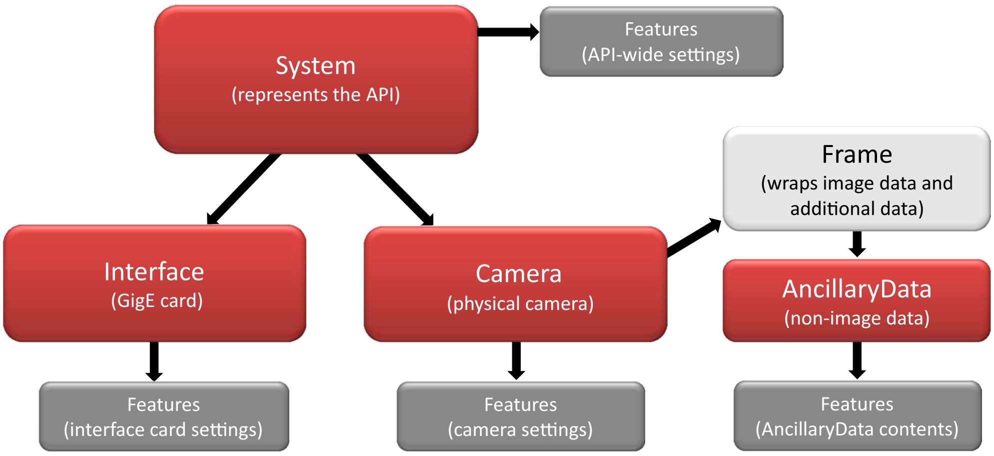

Figure 2: Vimba API Entities

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 **System** entity represents the API itself. Thus, 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 only 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-enqueuing 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.

3.3 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 . Vimba can read and interpret the camera XML file. This means that Vimba is immediately ready-to-use with any AVT 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 AVT cameras with GenICam-based third-party software.

Further readings

- In-depth information about **GenICam** is available on the EMVA website: http://www.emva.org/cms/index.php?idcat=27
- AVT GigE camera features are described in the AVT GigE Camera and Driver Features Manual.
- Features for controlling Vimba are described in the Vimba Feature Manual

3.4 Vimba's Transport Layers

A transport layer (TL) transports the data from the camera to an application on the PC. Vimba contains a GenICam transport layer (GenTL) for AVT GigE cameras.

Since Vimba's transport layers support GenICam, GigE cameras can easily be used with a GenICam-compliant third-party software.

For more information, see the AVT GigETL Manual .

3.5 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 provides 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.

Buffer management

Every image acquisition requires allocating memory and handling frame buffers. Independent from the API, the following interaction between the user and the 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.

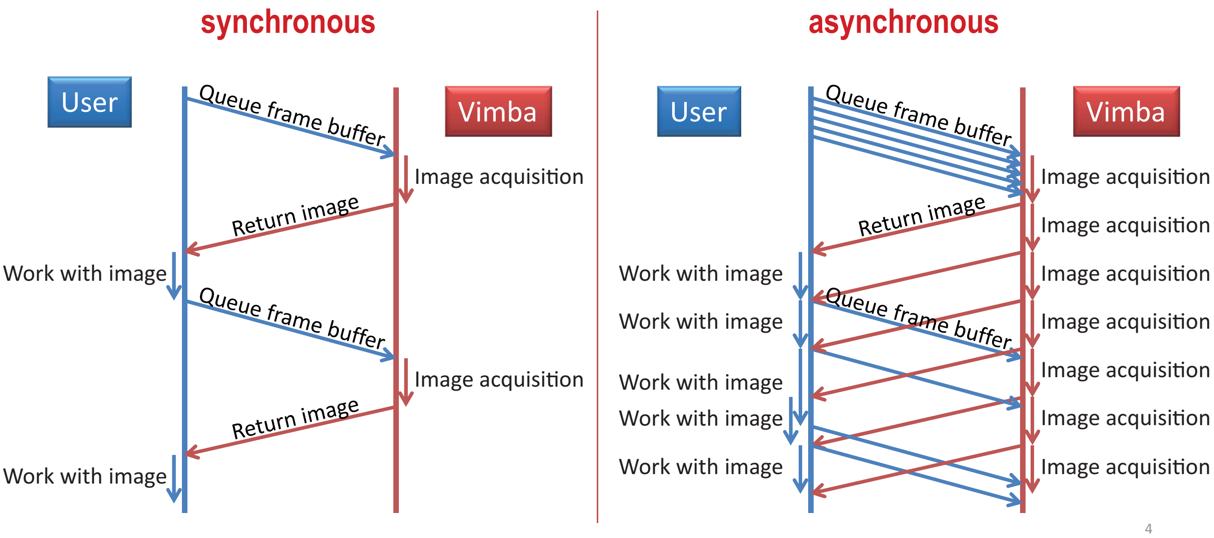

Figure 3: Acquisition Models

3.6 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 further details, refer to the example programs.

Caution

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. Thus, it is highly recommended to exit the event handler function as fast as possible.

Note

Not all API functions may be called from the event handler function. For more details, see the Programmer's Manual for the programming language of your choice: Vimba C Programmer's Manual or Vimba C++ Programmer's Manual.

4 Vimba Class Generator (not for ARM systems)

The Vimba Class Generator is a tool for easily creating classes for Vimba C++ API that are more comfortable to use than the standard API. The generated classes offer access functions for each found feature, depending on the type of the feature.

Note

After a 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'd like to generate code
2. Choose the destination folder for the generated files
3. Customize the code generation with several visible options and template files

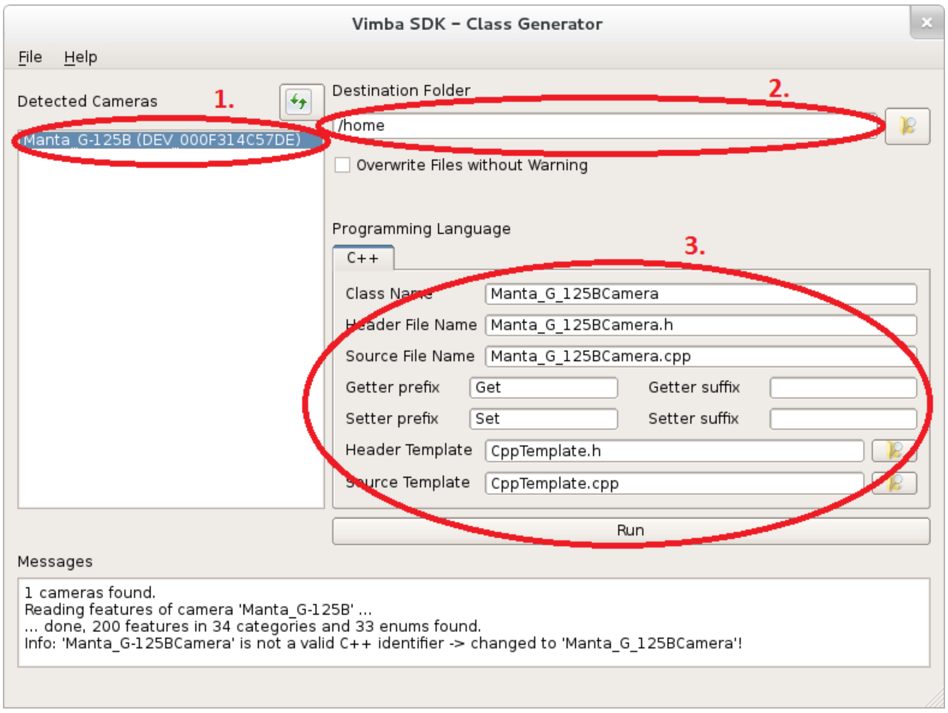

Figure 4: Vimba Class Generator - Main Window

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

Note

To detect new cameras, click the Refresh button.

To change the default destination folder, click the button beside the text field *Destination Folder*. Alternatively, you may enter a path manually, but in this case you have to 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, see chapter 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.

5 Vimba Setup

5.1 Prerequisites

If you wish to compile the examples that come with Vimba and the open source Vimba C++ API, you need to make sure you have installed the following library packages. You will probably find most of them being already part of your system.

- tar
- make
- pkg-config
- g++ (PC: Version 4.4.5 or above / ARM: Version 4.7.3 or above)
- glibc6 (PC: Version 2.11 or above / ARM: Version 2.15 or above)
- Qt (PC: Version 4.8.4 / ARM: n.a.)
- TinyXML (Version 2.5.3 or above)

Except for *tar* and the C runtime library *glibc6*, you will need these libraries (and the according development packages) only if you intend to compile the Vimba examples or the Vimba C++ API. Use the provided Makefiles to compile the examples. The remaining necessary runtime libraries for executing the examples including the VimbaViewer are provided with AVT Vimba.

5.2 Installing AVT Vimba

AVT Vimba comes as a tarball. In order to set up Vimba, follow these steps:

1. Uncompress the archive with the command tar -xf ./AVTVimba.tgz to a directory you have writing privileges for, e.g. /opt. Under this directory, Vimba will be installed in its own folder. In the following, we will refer to this path as [InstallDir].
2. Go to [InstallDir]/AVTGigETL and execute the shell script Install.sh with super user privileges (e.g., sudo./Install.sh). This registers the GENICAM_GENTL32_PATH and / or the GENICAM_GENTL64_PATH environment variable through a startup script in /etc/profile.d so that every GenICam GenTL consumer (such as the examples that ship with AVT Vimba) can access the AVT Gigabit Ethernet Transport Layer. Please note that this is a per-user setting.
3. Log off once. When you log on again, these changes will have been applied to the system.

Now you are ready to run the VimbaViewer that can be found in Vimba/Viewer/Bin. This program allows you to configure your AVT cameras and capture images.

In order to change the IP configuration of a camera in a foreign subnet, VimbaViewer must be run with super user privileges (e.g., sudo -E ./VimbaViewer). Note that running it as root user instead of using sudo -E requires the GENICAM_GENTL32_PATH and / or GENICAM_GENTL64_PATH being set for the root user as well.

Running and compiling the examples

Vimba includes many precompiled examples that can be found in AVTVimba/VimbaC/Examples/Bin and AVTVimba/VimbaCPP/Examples/Bin.

If you want to compile the examples yourself, navigate to Build/Make in the VimbaC and VimbaCPP example folders and type make in your shell.

5.3 Uninstalling AVT Vimba

Remove the startup scripts by running the shell script `Uninstall.sh` as super user. This prevents any GenTL consumer from loading the AVT Gigabit Ethernet Transport Layer. Then simply remove the installation directory.

6 References

The following table lists some documents with more detailed information about the components of AVT Vimba. Please note that the links are valid only if the corresponding component has been installed.

AVT GigE Vision Transport Layer and Cameras

- AVT GigE TL Feature Manual.
- AVT GigE Camera and Driver Features Manual.

AVT Image Transform Library

- Programmer's Manual.

Vimba C API

- Programmer's Manual.
- Vimba Features.

Vimba C++ API

- Programmer's Manual.
- Vimba Features.