AVT Vimba

User Guide for Linux

V1.2
2013-Jun-25

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-Apr-03	Initial version
1.1	2013-Jun-18	Added chapter for Class Generator, small corrections, layout changes

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	*Mode*
Parentheses and/or blue	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

The Vimba SDK is a camera-independent SDK for the 32-bit and 64-bit operating systems Windows XP, Windows 7, Windows 8, and various Linux distributions that can be applied for both AVT 1394 (Windows only) and AVT GigE Vision cameras. With the Vimba SDK, your application immediately supports AVT's 1394a / 1394b digital cameras (Windows only) as well as AVT's GigE Vision cameras.

3.1 Architecture

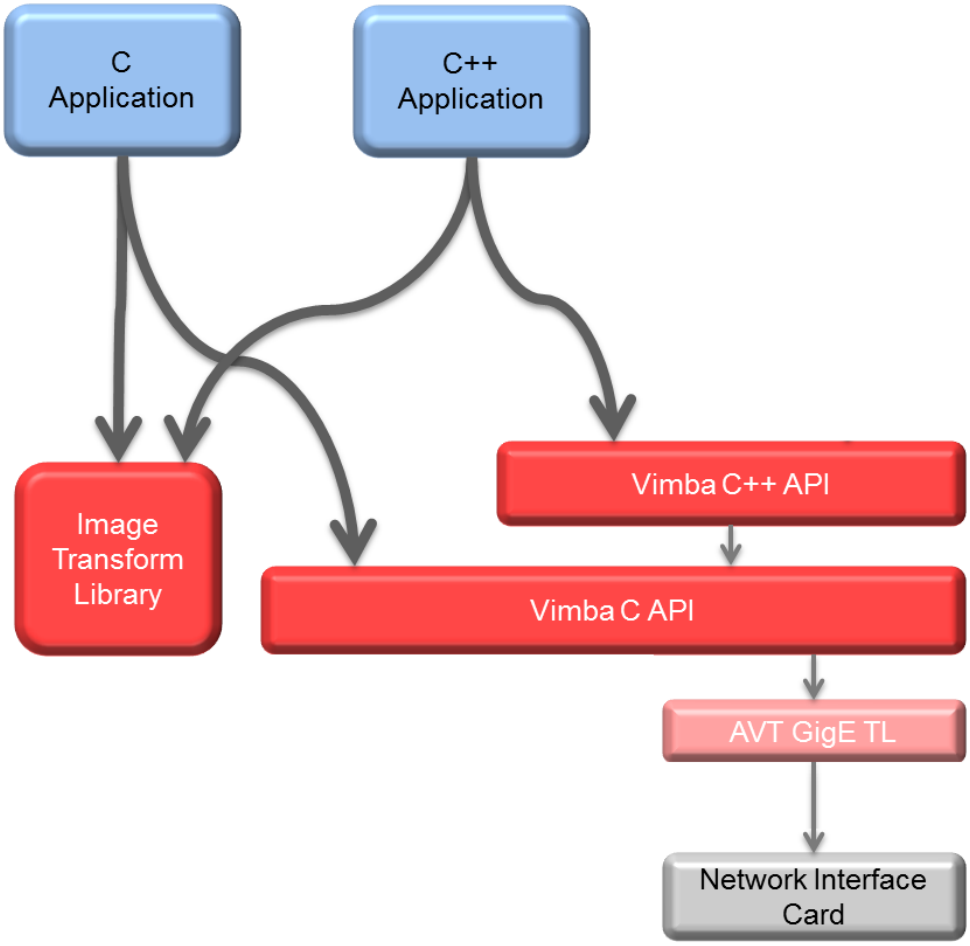

Figure 1: Vimba Architecture

The Vimba SDK provides application programming interfaces (APIs) for different programming languages: C and C++. All APIs cover the following functions:

- listing currently connected cameras
- controlling camera features
- receiving images from the camera
- notifications about cameras connecting or disconnecting

The Image Transform Library converts camera images into other pixel formats and creates color images from raw images (debayering). This is separated for the C and C++ API.

The APIs use transport layer (TL) modules to actually communicate with the cameras. These modules (currently only AVT GigE TL) are not directly accessible for the user application.

For more detailed information about the different APIs please look at the documents listed in section **??**.

3.2 API Object Model

The Vimba APIs use a defined object model for providing access to the different entities. For object oriented programming languages like C++, this object model is reflected in the API's class design, but even the C API supports this model by using handles as a representation of the different objects.

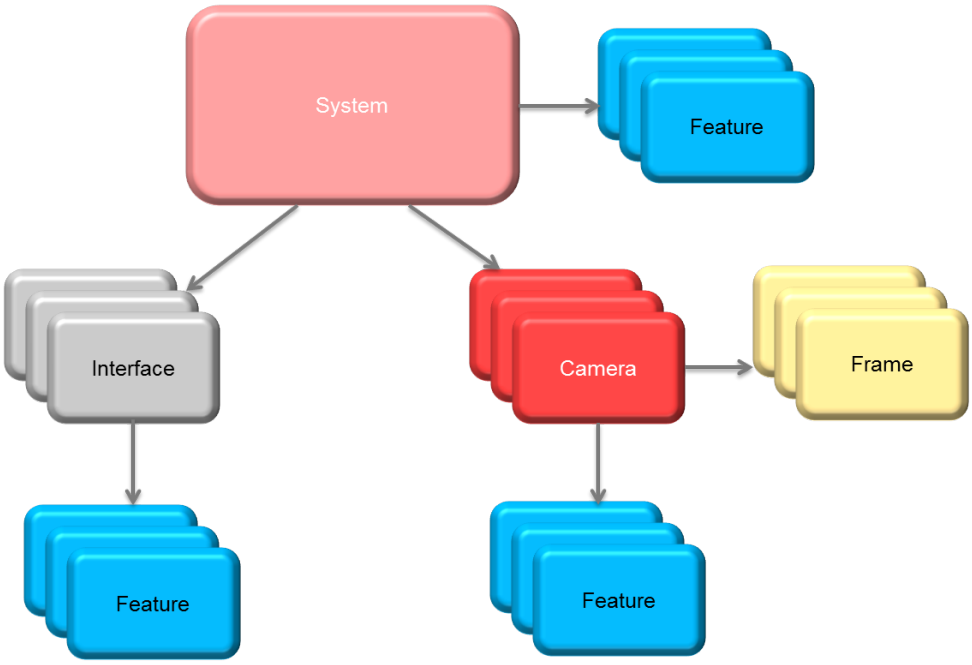

Figure 2: Vimba API Object Model

The *System* object represents the API itself. Thus only one instance of it is available. The application has to initialize the *System* object before using any other function. When the application has finished using

the API, it shuts it down through the *System* object. The *System* object holds a list of interfaces and cameras internally and serves as the main access point to these objects.

A *Camera* object controls a physical camera and receives images. It provides the same set of functions regardless of the underlying interface technology.

An *Interface* object represents a port on a physical interface card in the PC. Although a camera is connected through an interface, it is not necessary to use an *Interface* object to access a *Camera* object. This can be done directly via the *System* object. So the only purpose of the *Interface* object is to control the settings of the corresponding interface card.

All these objects - *System*, *Camera*, and *Interface* - have a list of *Feature* objects. They reflect the settings of these entities. The *System Features* contain information about API wide settings like what kind of transport layer modules are loaded. The *Camera Features* can be used to configure camera settings like exposure time or pixel format. *Interface Features* represent the settings of a physical interface card in the PC like an IP address of a network interface card. *Camera* and *Interface Features* are usually different according to the underlying interface technology.

Frame objects receive image data from the camera. They are created by the application and queued at the corresponding *Camera* object. When an image was received, the next available *Frame* is filled and handed over to the application through a dedicated notification. After the application processed the image data, it should return the *Frame* to the API by re-enqueuing it at the corresponding *Camera*.

3.3 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 - detection of cameras connected to the PC or frame reception. 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 used programming language. Have a look at the example programs for more details.

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.
Not all API functions may be called from the event handler function. For more details, see the Programmer's Reference document for the programming language of your choice.

4 Vimba Class Generator

The Vimba Class Generator is a tool for easily creating classes for Vimba C++ (Windows and Linux) and Vimba.NET (Windows only) APIs 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, re-generate the files and merge the access functions for new features manually into your previously generated code.

4.1 Main window

Upon startup the Vimba Class Generator assembles all the cameras accessible by Vimba.

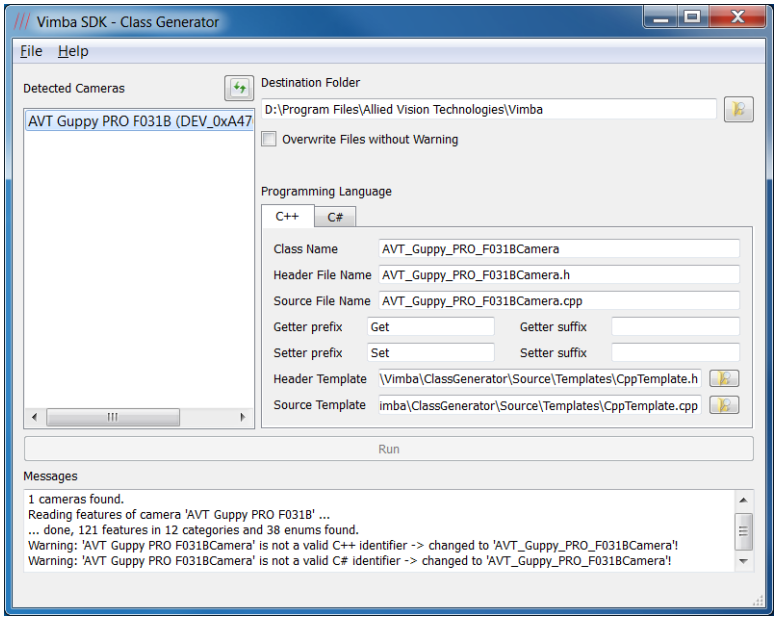

Figure 3: Vimba Class Generator - Main Window

The available cameras are listed in the box on the left side of the screen below *Detected cameras*. Select the camera that you want code generated for.

Note

You may click the Refresh button above the listed cameras at any time to re-scan for new cameras.

In the edit field below *Destination Folder*, you may enter the folder where the files will be generated. By clicking the associated button, you may select a folder from the file system.

The Vimba Class Generator will not overwrite existing files without warning. By selecting the check-box *Overwrite Files without Warning*, you may change this behavior.

The options below *Programming Language* allow you to configure the code generation. By selecting the tab, you may switch the programming language for creation. For more language-dependent options, see chapter C++ code generation.

If everything is configured, the *Run* button becomes active. Clicking it will generate the code for the selected camera, programming language and options.

Any message that is generated during selection of the options or during code generation is displayed in the text box at the bottom of the window.

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 put before the feature name for each getter function.
- Getter suffix: The text that is put after the feature name for each getter function.
- Setter prefix: The text that is put before the feature name for each setter function.
- Setter suffix: The text that is put 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.

The template file for the header file may contain the following 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 put.
- ### METHOD_DECLARATIONS ###: This is where the method declarations are put.
- ### VARIABLE_DECLARATIONS ###: This is where the variable declarations are put.

The 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 put.

You may move around these variables in the template file to generate a file that better suits your requirements.

5 Installing AVT Vimba on Linux

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 packages. You will probably find most of them being already part of your system.

- tar
- make
- pkg-config
- g++ (Version 4.4.5 or above)
- glibc6 (Version 2.11 or above)
- Qt (Version 4.8.4)
- TinyXML (Version 2.5.3 or above)

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

5.2 Setting up AVT Vimba

AVT Vimba comes as a tarball. Simply uncompress the archive with the command *tar -xf ./AVTVimba.tgz* to a directory you have writing privileges for. This will create a directoy named AVTVimba. First off navigate to AVTVimba/AVTGigETL and execute the shell script *Install.sh* with root privileges (e.g. *sudo ./Install.sh*). This will register the *GENICAM_GENTL32_PATH* and / or the *GENICAM_GENTL64_PATH* environment variable through a startup script in */etc/profile.d* so that every GenICam GentTL consumer (such as the examples that ship with AVT Vimba) can access the AVT Gigabit Ethernet Transport Layer. Please note that you have to log off once before these changes will be applied to your 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. VimbaViewer must be run with root privileges (e.g. *sudo -E ./VimbaViewer*) if you want to change the IP configuration of a camera in a foreign subnet (running it as *root* user instead of using *sudo -E* will require you to set the environment variables from above manually).
Furthermore you can run one of the 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 examples folders and type *make* in your shell.

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

- GigE Vision Transport Layer Feature Description.

AVT Image Transform Library

- Programmer's Manual.

Vimba C API

- Programmer's Manual
- Function Reference
- Vimba Features

Vimba C++ API

- Programmer's Manual
- Function Reference
- Vimba Features