Allied Vision

vimba

Vimba

Vimba Manual for Linux

2.0

Legal Notice

Trademarks

Warranty

Copyright

Allied Vision Technologies GmbH 02/2016

All rights reserved.
Managing Director: Mr. Frank Grube
Tax ID: DE 184383113

Headquarters:

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

1 Contacting Allied Vision

Connect with Allied Vision by function

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

Find an Allied Vision office or distributor

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

Email

info@alliedvision.com
support@alliedvision.com

Telephone

EMEA: +49 36428-677-0
The Americas: +1 978-225-2030
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
President/CEO: Frank Grube
Registration Office: AG Jena HRB 208962

2 Document history and conventions

This chapter includes:

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
1.4	2015-11-09	Renamed several Vimba components and documents (”AVT” no longer in use), links to new Allied Vision website, new document layout
2.0	2016-Feb-29	Added Goldeye CL compatibility, new document layout

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 Vimba SDK Overview

This chapter includes:

Vimba for Linux is an SDK for all Allied Vision cameras with GigE and USB interface. 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.

Supported cameras:

- Allied Vision GigE cameras
- Allied Vision USB 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

You can download application notes about installing Vimba under Linux, recommended ARM boards, and cross-compiling to ARM from:
http://www.alliedvision.com/en/products/software.html

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

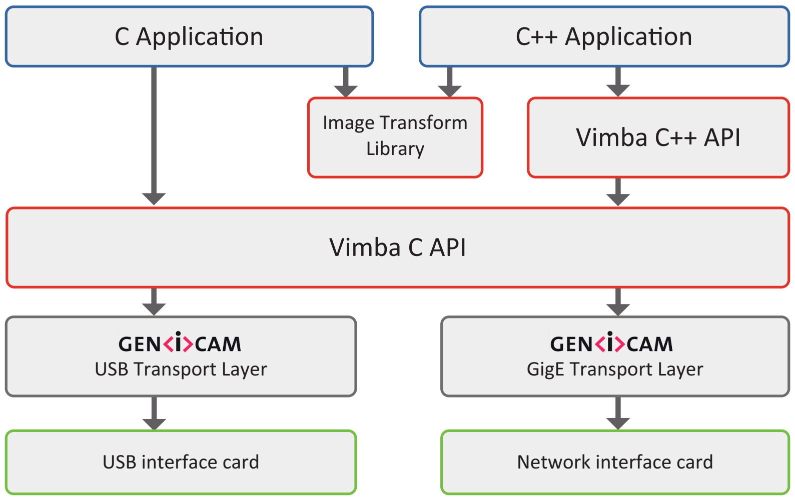


Figure 1: Vimba Architecture

Vimba provides two APIs:

- The C API is Vimba's basic API. It can also be used as an 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.

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 (Vimba GigE TL, and Vimba USB TL) can not be accessed directly through Vimba.

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

- Vimba C Manual (includes a function reference)
- Vimba C++ Manual (includes a function reference)
- Vimba Image Transform 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.

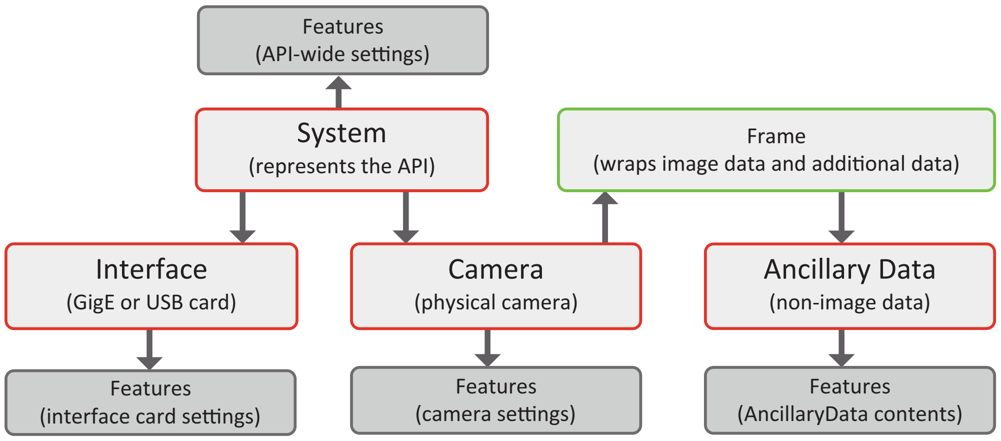


Figure 2: Vimba API Entities

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 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/cms/index.php?idcat=27
- Allied Vision GigE camera features are described in the GigE Features Reference.
- Allied Vision USB camera features are described in the USB Features Reference.
- Features for controlling Vimba are described in the Vimba Features 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 GenICam transport layers (GenTL) for Allied Vision GigE and USB cameras.

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

For more information, see the Vimba GigE TL Features Manual and the Vimba USB TL Features 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.

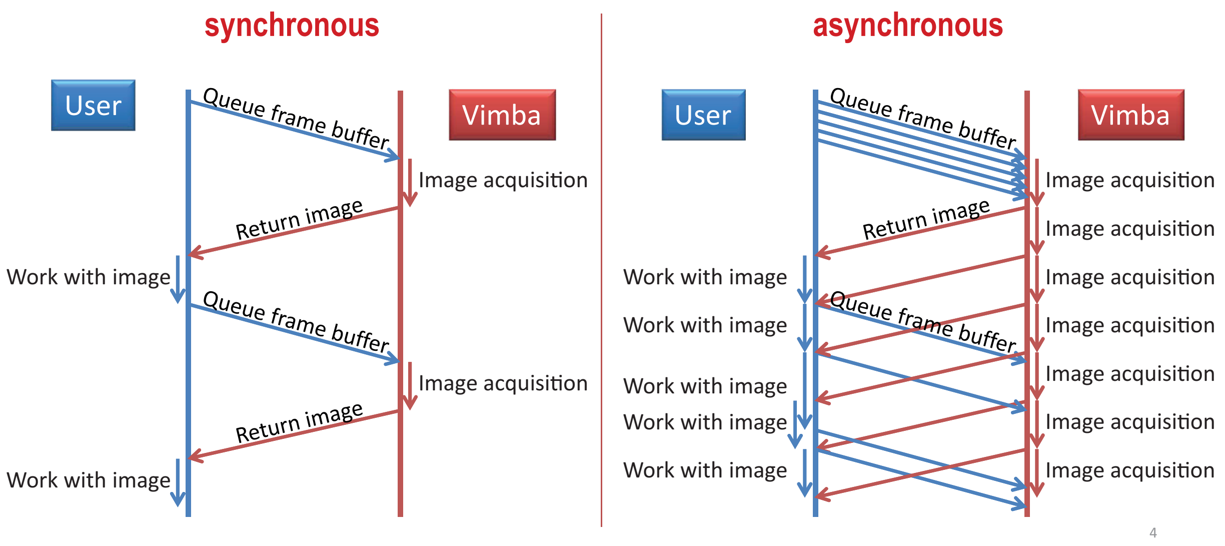


Figure 3: Acquisition Models

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.

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 API Manual for the programming language of your choice: Vimba C Manual or Vimba C++ Manual.

4 Vimba Class Generator (not for ARM systems)

This chapter includes:

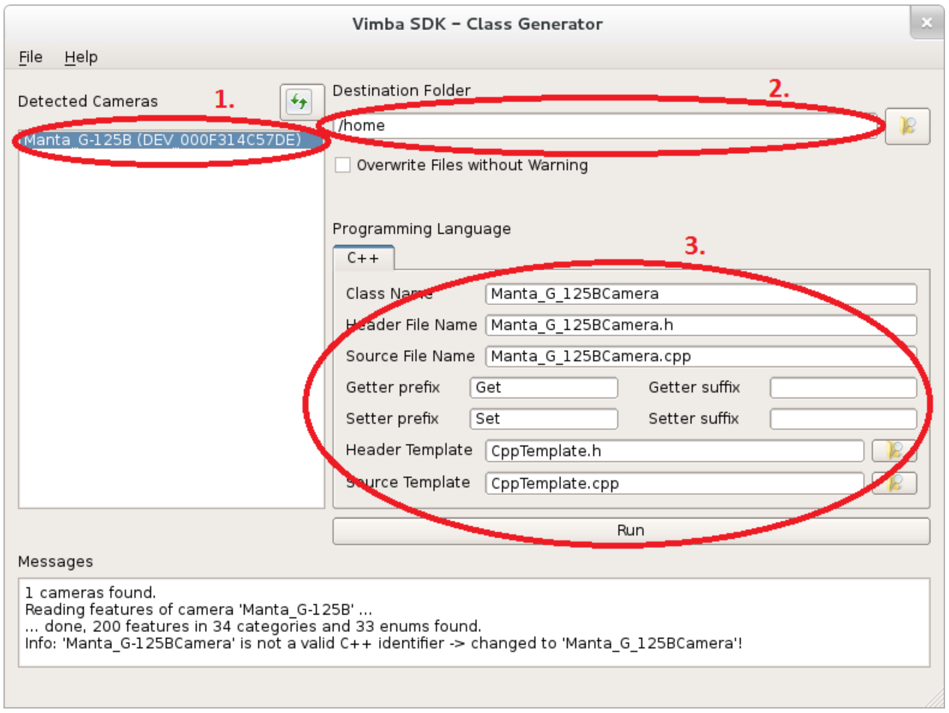
The Vimba Class Generator is a tool for easily creating classes for Vimba C++ API that are more comfortable to use than the standard API. The generated classes offer access functions for each found feature, depending on the type of the feature.

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



Manta_G-125B (DEV_000F314C57DE)

/home

Overwrite Files without Warning

Class Name: Manta_G_125BCamera
Header File Name: Manta_G_125BCamera.h
Source File Name: Manta_G_125BCamera.cpp
Getter prefix: Get
Getter suffix:
Setter prefix: Set
Setter suffix:
Header Template: CppTemplate.h
Source Template: CppTemplate.cpp

```
1 cameras found.
Reading features of camera 'Manta_G-125B' ...
... done, 200 features in 34 categories and 33 enums found.
Info: 'Manta_G-125BCamera' is not a valid C++ identifier -> changed to 'Manta_G_125BCamera'!
```

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, 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 Firmware Updater

This chapter includes:

The Vimba Firmware Updater supports firmware uploads to Allied Vision USB cameras 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. Compatibility:

- The GUI application is available for x86 host PCs systems.
- The command line application is available for all x86 systems and x64 host PCs.

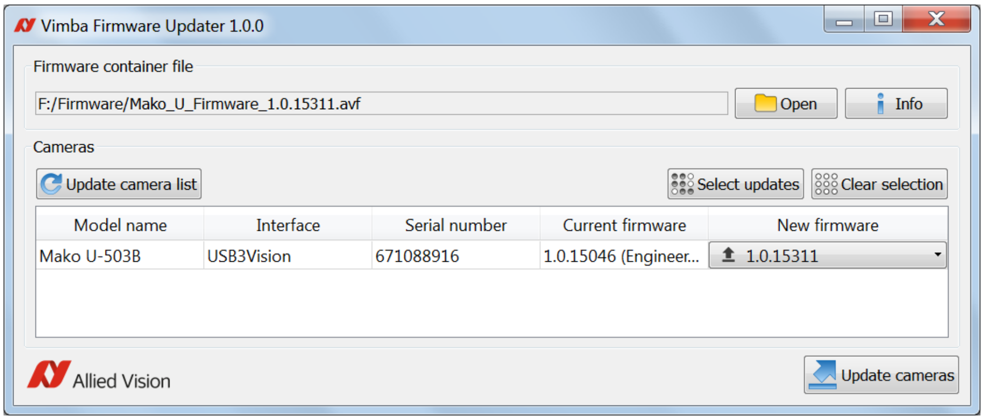
If you prefer to upload firmware via command line, see chapter Command line Firmware Updater.

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

5.1 Uploading firmware

To upload new firmware to your cameras, carry out the following steps (see Figure 5):

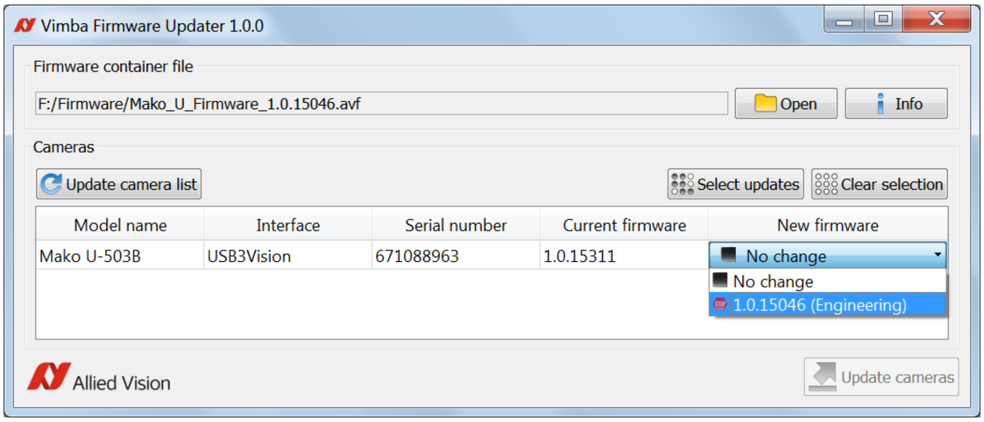
1. Connect your Allied Vision cameras and start Vimba Firmware Updater.
2. Click **Open** and select a firmware container file.
   Optional: Click **Info** to get details about the selected firmware.
3. Click **Update cameras** to upload the automatically selected firmware to your cameras.



Model name	Interface	Serial number	Current firmware	New firmware
Mako U-503B	USB3Vision	671088916	1.0.15046 (Engineer...	1.0.15311

Figure 5: Firmware Updater - Main Window

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



Model name	Interface	Serial number	Current firmware	New firmware
Mako U-503B	USB3Vision	671088963	1.0.15311	No change

No change
1.0.15046 (Engineering)

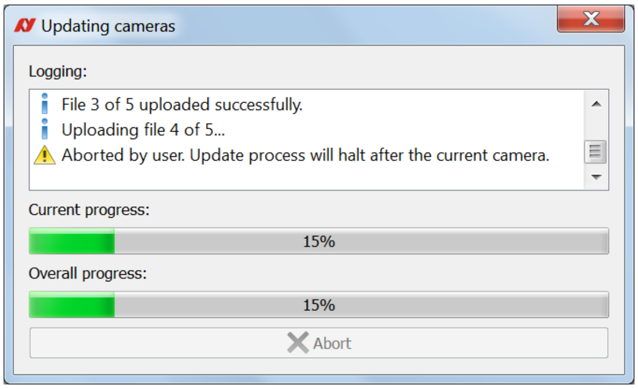
Figure 6: Firmware Updater - Manual Update

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.

5.2 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.



```
File 3 of 5 uploaded successfully.
Uploading file 4 of 5...
Aborted by user. Update process will halt after the current camera.
```

Figure 7: Firmware Updater - Abort

5.3 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: http://www.alliedvision.com/support
- Make sure the USB transport layer is available (run its shell script with root privileges).
- 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.

5.4 Command line Firmware Updater

There is also a separate command line tool for updating the firmware of cameras: FWUpdaterConsole. This tool provides two main functionalities via command line options:

- ”–show” or ”-s” allows to display information about either camera firmware or a firmware file
- ”–write” or ”-w” performs the actual update

See the following list of use cases for an explanation:

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

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	`--verbose, -v`
Force writing	`--force, -f`
Repair device firmware during write	`--repair, -r`

Table 2: Command options for the firmware update

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

6 Vimba Setup

This chapter includes:

6.1 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)
- libqt4 (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 Vimba. Furthermore the Vimba C++ example AsynchronousOpenCVRecorder requires OpenCV 3.0 and comes with a script for compilation and installation of OpenCV. OpenCV can be downloaded from

http://opencv.org

6.2 Installing Vimba

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

1. Uncompress the archive with the command tar -xf ./Vimba.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]/GigETL and execute the shell script Install.sh with super user privileges (e.g., sudo ./Install.sh or su -c ./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 Vimba) can access the Vimba GigE Transport Layer. For USB support, navigate to [InstallDir]/USBTL and proceed accordingly. Please note that these are a per-user settings.
3. Log off once. When you log on again, these changes will have been applied to the system. If you encounter problems detecting USB cameras, please reboot your machine.

Now you are ready to run the VimbaViewer that can be found in Vimba/Viewer/Bin. This program allows you to configure your Allied Vision 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 or su -m -c ./VimbaViewer). Note that running it as root user instead of using sudo -E (or su -m) 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 Vimba/VimbaC/Examples/Bin and Vimba/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.

6.3 Uninstalling Vimba

Remove the startup scripts by running the shell scripts [InstallDir]/GigETL/Uninstall.sh and [InstallDir]/USBTL/Uninstall.sh as super user. This prevents any GenTL consumer from loading the Vimba GigE and USB Transport Layers. Then simply remove the installation directory.

7 References

The following table lists some 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

Vimba Image Transform Library:

- Vimba Image Transform Manual

Vimba C API:

- Vimba C Manual
- Vimba Features Manual

Vimba C++ API:

- Vimba C++ Manual
- Vimba Features Manual