AVT Vimba

User Guide for Windows

V1.3
2014-07-09

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

ALLIED Vision Technologies

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	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

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 Windows is an SDK for all AVT 1394 and 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. To ensure a high performance with less CPU load, Vimba includes 1394 and GigE drivers. The 1394 driver is required to allow access to AVT 1394 cameras, while the use of the GigE filter driver is optional.

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

Supported cameras:

- AVT GigE cameras
- AVT 1394 cameras

Supported operating systems:

- Windows XP SP3 (32-bit)
- Windows 7 (32-bit and 64-bit)
- Windows 8 (32-bit and 64-bit)

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.

Users of Cognex VisionPro software can use the AVT Cognex Adapter to access AVT cameras. If installed, see the Vimba Cognex Adapter User Guide for more details.

3.1 Architecture

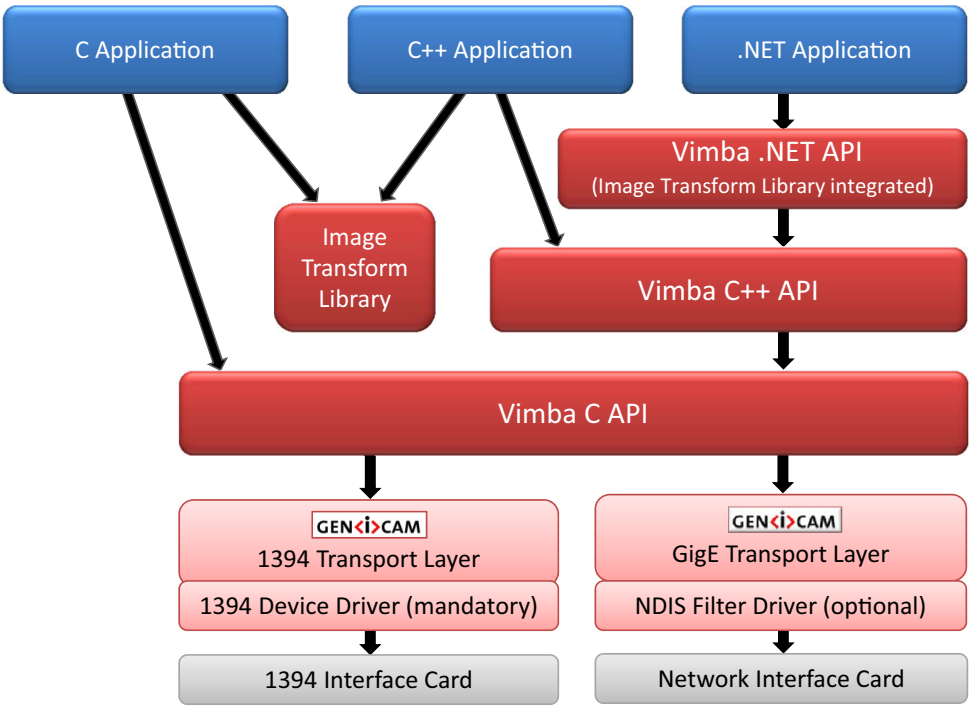

Figure 1: Vimba Architecture

Vimba provides three 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.
- 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 (AVT 1394 TL and 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 .NET 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.

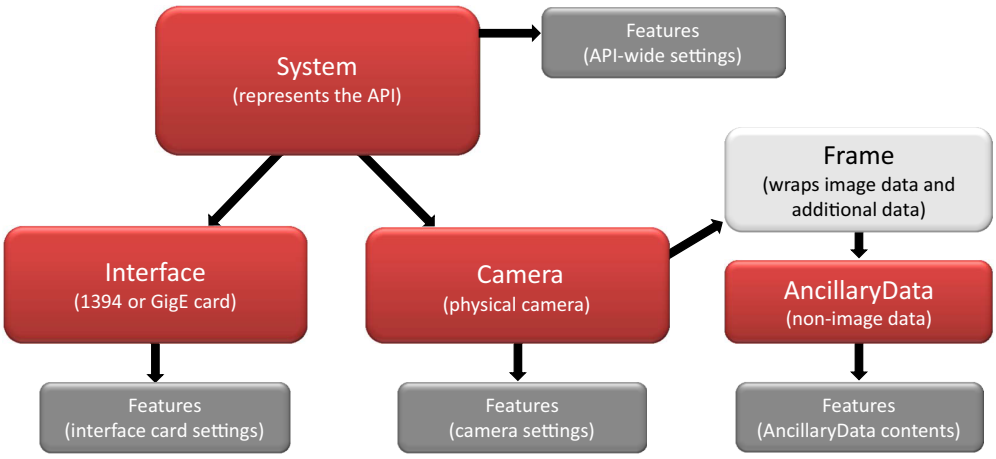

Figure 2: Vimba API Entities

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 (in the case of 1394 cameras, the XML file is created by the AVT1394TL 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 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.
- AVT 1394 camera features are described in the 1394 Transport Layer Feature 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 1394 cameras and one for AVT GigE cameras.

The **AVT GigETL** optionally uses the AVT filter driver, which provides a high performance with less CPU load. The AVT filter driver is available as 32-bit and 64-bit version. It is compatible with all AVT GigE cameras.

The **AVT 1394TL** uses the (mandatory) high-performance intek driver (available as 32-bit and 64-bit version), which replaces the low-performance Microsoft driver.

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

For more information, see the AVT GigETL Manual and the AVT1394TL 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 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.

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.

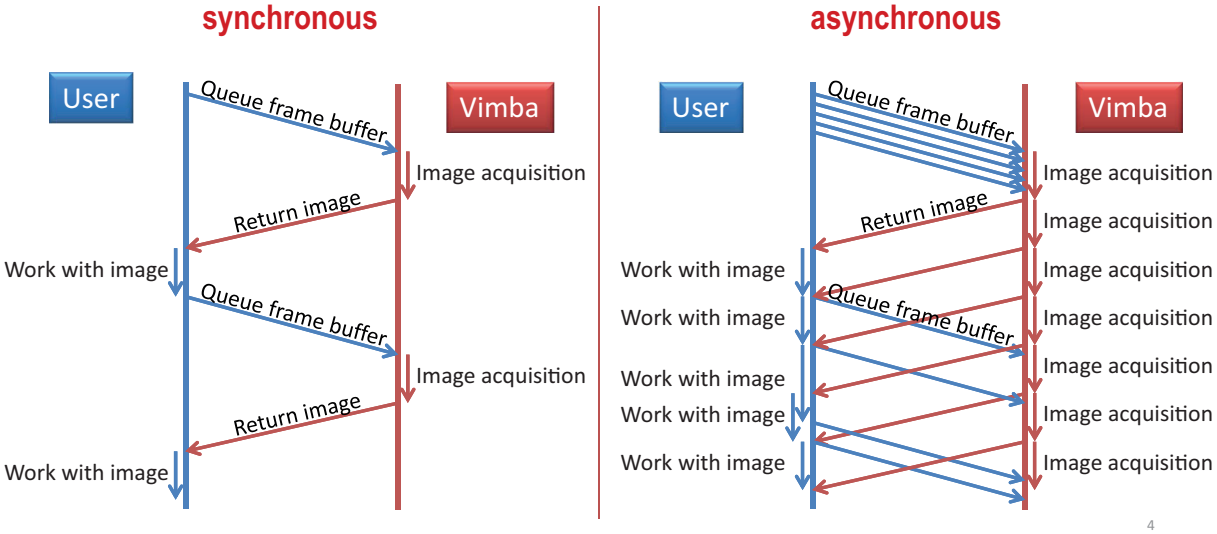

Figure 3: Acquisition Models

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 , Vimba C++ Programmer's Manual or Vimba .NET Programmer's Manual.

4 Vimba Class Generator

The Vimba Class Generator is a tool for easily creating classes for Vimba C++ and Vimba.NET 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, 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. Select the programming language
4. Customize the code generation with several visible options and template files

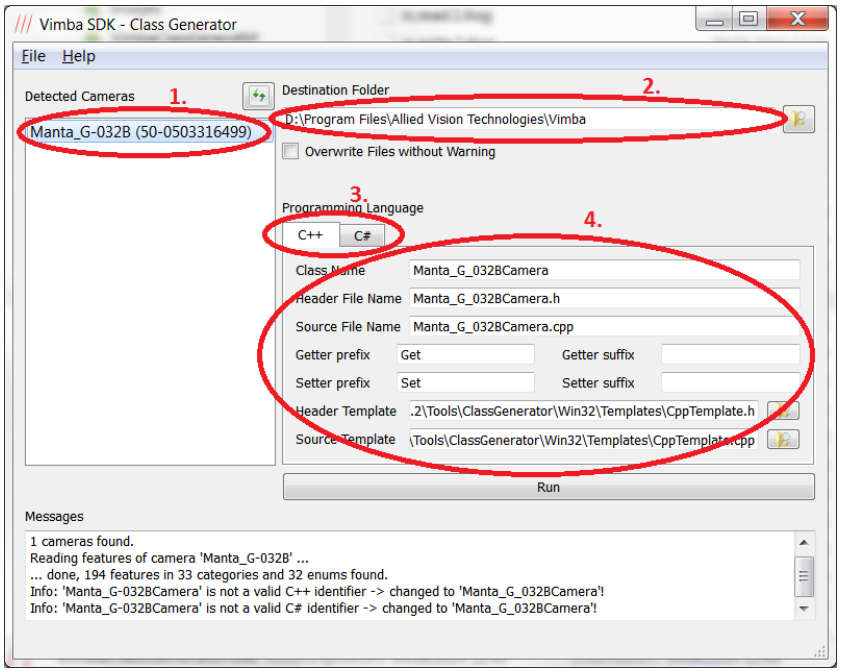

Figure 4: Vimba Class Generator - Main Window

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

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. 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 AVT Driver Installer

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

5.1 Main window

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

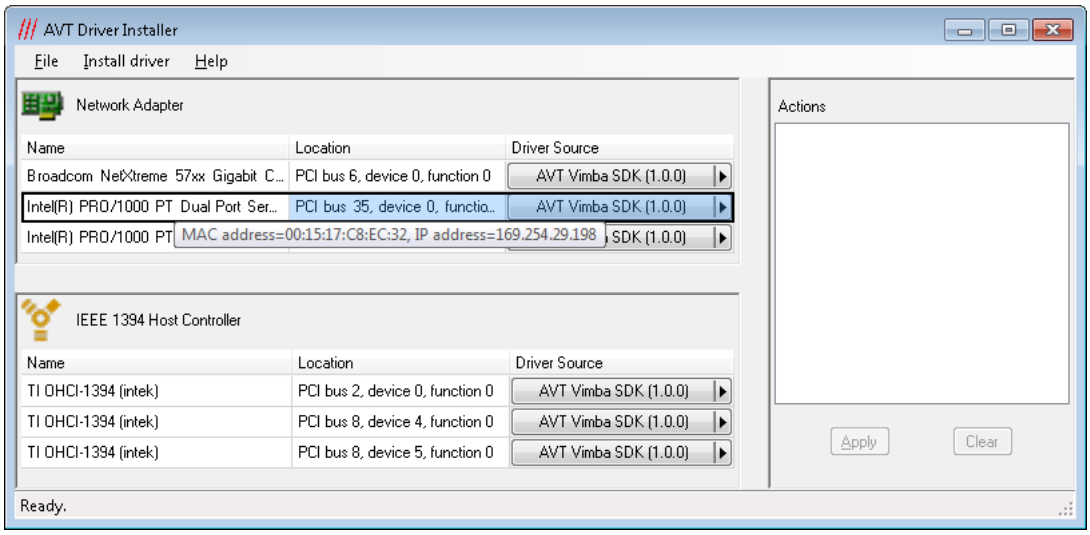

Figure 5: Driver Installer - Main Window

The window shows two panels on the left side: the top one lists the network and the bottom one the 1394 adapters. If no network or 1394 adapters are installed, the corresponding panel is hidden. If no Vimba suitable adapters are present in the system, an error message is shown and the application is closed.

All adapters are listed with their *Name* and their physical *Location* in the PC. To easily identify a network adapter, its MAC address and IP addresses are shown in a tool tip. The *Driver Source* column displays where the currently installed driver comes from (e.g. the AVT 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 if the *Apply* button is clicked.

To change the driver for adapters, select the product (e.g. AVT SDK) whose driver should be installed. This can be done by three different methods:

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

Note

The Driver Installer supports multi-selection of adapters: 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 AVT SDKs. Additionally, one item is added to each list that can be used to remove the AVT driver from the adapter: *<disable AVT filter driver>* for network adapters 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 are disabled, so you cannot add a second action for the same adapter.

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

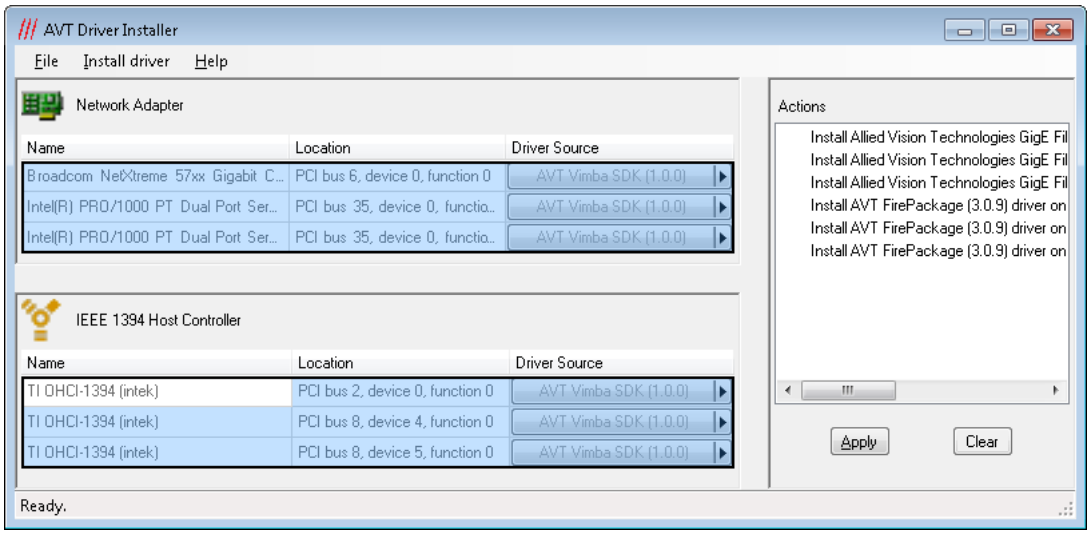

Figure 6: Driver Installer - Prepare actions

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

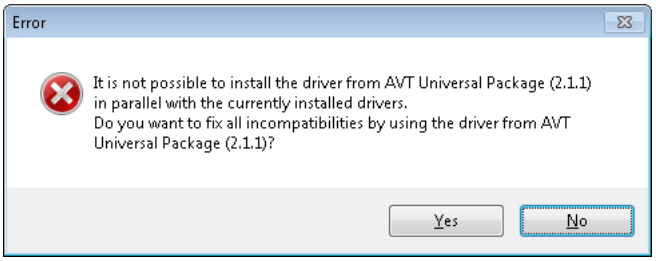

Figure 7: Driver Installer - Incompatible products detected

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

To start the actual installation, click the *Apply* button. During the installation, which can 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.

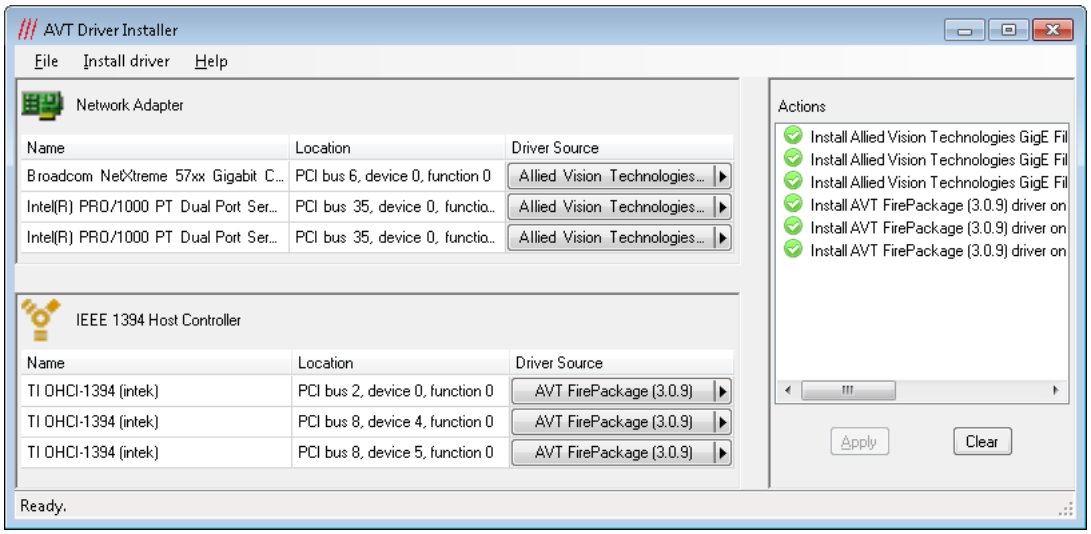

Figure 8: Driver Installer - Installation successful

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:

```
AVTDriverInstaller.exe [command] [options]
```

The following commands are available:

Command	Description	Mandatory options
`help`	Displays infos about the usage of the Driver Installer	*none*
`info`	Displays infos about installed AVT products and hardware.	*none*
`add`	Adds a driver of an AVT product to the driver store.	*/product*
`remove`	Removes a driver of an AVT product from the driver store.	*/product*
`exists`	Checks whether the given AVT product is installed.	*/product*
`install`	Installs a driver of an AVT product on the given adapters.	*/product* and */deviceClass*

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: 'AVT Vimba (1.0.0))'.
`/deviceClass deviceClassName`	Identifies the hardware devices by their device class (either 'net' for network adapter 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 AVT 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 */deviceClass* can be specified multiple times.

Note

Use the *info* command to find out the upgrade codes or exact names and versions for installed AVT 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:

- `AVTDriverInstaller info`
  Displays infos about installed AVT products and hardware.
- `AVTDriverInstaller remove /product "AVT Vimba (1.0.0)"`
  Removes all drivers of Vimba 1.0 from the driver store.
- `AVTDriverInstaller install /product d108d42f-c1d4-4d86-aa1f-e3ec4a137aaf /deviceClass 1394 /deviceClass net`
  Install drivers of Vimba 1.0 on all 1394 and network devices. Here the product is given by its upgrade code.

6 Vimba Deployment

With Vimba, it is easy to integrate the setup into your own projects. Of course you can use the Vimba setup executable to manually install the software on the target machine, but it is also possible to run the installer silently. At first you need the Microsoft Installer files (MSI). As they are contained in the setup executable, it is not necessary to download them separately from the AVT website. Just run

```
AVTVimba.exe /extract
```

to extract the MSIs from the setup executable. 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
AVT1394TL_Win32.msi	AVT 1394 Transport Layer for 32-bit Windows operating systems.
AVT1394TL_Win64.msi	AVT 1394 Transport Layer for 64-bit Windows operating systems.
AVTGigETL_Win32.msi	AVT GigE Vision Transport Layer for 32-bit Windows operating systems.
AVTGigETL_Win64.msi	AVT GigE Vision Transport Layer for 64-bit Windows operating systems.
AVTVimba_Win32.msi	AVT Vimba for 32-bit Windows operating systems.
AVTVimba_Win64.msi	AVT 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 AVT transport layer modules. If your application uses one of the Vimba APIs, it is necessary to additionally install Vimba.

Microsoft provides a tool named *msiexec* to install an MSI file. The following command line installs the core components of the AVT 1394 Transport Layer module to the default programs folder:

```
msiexec -i AVT1394TL_Win32.msi
ADDLOCAL="VCRedist100,FireWireTL_RegisterGenICamPathVariable,FireWireTL_Core"
```

And the same for the AVT GigE Vision Transport Layer module:

```
msiexec -i AVTGigETL_Win32.msi
ADDLOCAL="VCRedist100,GigETL_RegisterGenICamPathVariable,GigETL_Core"
```

The installation of Vimba depends on what kind of Vimba API should be provided. For the Vimba C API, the command line is:

```
msiexec -i AVTVimba_Win32.msi ADDLOCAL="VCRedist90,VCRedist100,
VCRedist100_MFC,Vimba_Core,VimbaC_Core,AvtImageTransform_Core,Vimba_DrvInst"
```

For the Vimba C++ API it is:

```
msiexec -i AVTVimba_Win32.msi ADDLOCAL="VCRedist90,VCRedist100,
VCRedist100_MFC,Vimba_Core,VimbaCPP_Core,AvtImageTransform_Core,Vimba_DrvInst"
```

And the Vimba .NET API is installed with:

```
msiexec -i AVTVimba_Win32.msi ADDLOCAL="VCRedist90,VCRedist100,
VCRedist100_MFC,Vimba_Core,VimbaNET_Core,AvtImageTransform_Core,Vimba_DrvInst"
```

Other useful parameters for *msiexec* are:
INSTALLDIR="*path to desired installation directory*"
ALLUSERS="1" Performs a machine-wide installation. If omitted, a per user installation is performed

Note

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

7 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 a sub-folder `VimbaCPP_Source` to the examples installation folder.

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++ Programmer's Manual.

Note

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

8 Vimba Cognex Adapter

Since Vimba 1.2, there is an adapter for Cognex software in the Vimba package. With this adapter, all cameras that are supported in Vimba are automatically supported in Cognex software.

For a detailed description, see the Vimba Cognex Adapter User Guide.

9 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 1394 Transport Layer and Cameras

- AVT 1394 TL Feature Manual.

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.

Vimba .NET API

- Programmer's Manual.
- Vimba Features.

AVT Vimba Cognex Adapter

- Vimba Cognex Adapter User Guide.