

GenICam – The New Programming Interface Standard for Cameras

We had a dream: any machine vision camera from any vendor using any interface technology implementing any feature should be accessible through any image processing library using the same standardized application programming interface (API). “We” that is a group of currently 28 machine vision companies which started in September 2004 to define an API standard and provide a reference implementation. Two years later the standard and the reference implementation are now ready and products are becoming available. In particular any GigE Vision compliant Ethernet based camera has a built in support for GenICam but solutions for 1394 DCAM and Camera Link are also available. This article explains the details.

GenICam¹ aims to give customers access to a camera’s features through a standardized API no matter if the customer is talking to the camera directly or through an image processing library (see Figure 1).

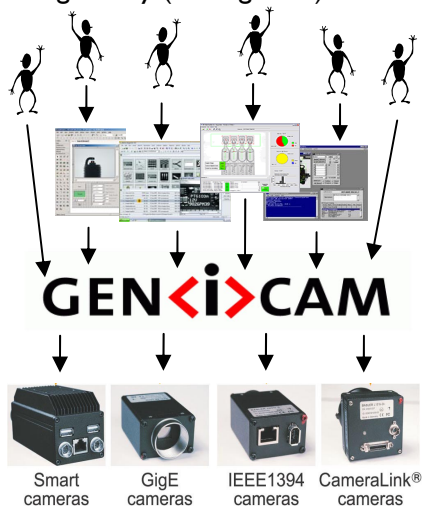


Figure 1 GenICam provides a Unified Programming Interface for machine vision cameras

Technically the camera vendor provides an XML file containing a machine readable description of the camera’s registers and how to map the registers to features. This **camera description file** is interpreted by the GenICam library either at compile time or at run-time yielding a programming interface exposing all features described in the XML file.

The camera vendor is free to choose any name he likes for a feature but he is encouraged to select one of the ~180 **standard feature names** provided for the most typical camera features.

If a camera vendor implements a new feature all he has to do in order to make it accessible through any GenICam aware image processing library is to add its description to the XML file. There is no need for direct support by the library vendor any more. This saves cost and reduces time to market. Of course direct access without a library is also possible.

GenICam is a vendor independent standard and is not bound to a specific interface technology. Currently, GenICam focuses on GigE Vision, 1394 DCAM, Camera Link, and smart cameras, but other interfaces can be supported as well.

GenICam supports three main use cases:

- Configuring the camera
- Grabbing images
- Providing a user interface

In addition GenICam can deliver asynchronous events and give access to extra data transmitted together with the image.

Configuring the Camera

Configuring a camera means, for example, setting its Gain. The respective GenICam code snippet in C++ looks like this:

```
if( IsAvailable(Camera.Gain) )
    Camera.Gain = 42;
```

¹ GenICam = **Generic** Interface for **C**ameras

Before the Gain is set to the value of 42, the code verifies that the Gain feature is really available. This makes the code **generic**, meaning that it could be used with all kinds of cameras even if they do not have a Gain feature.

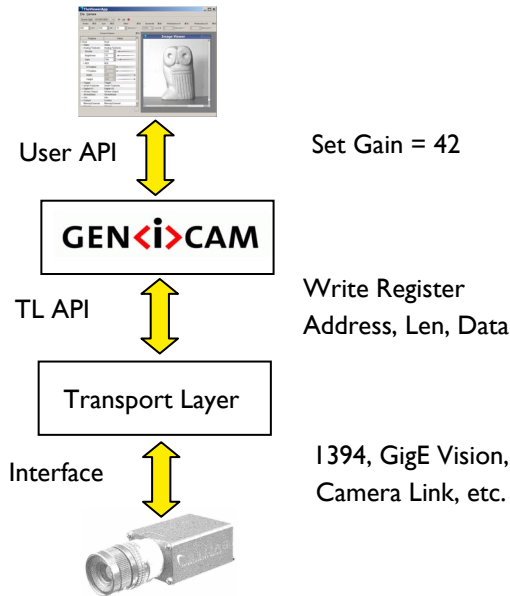


Figure 2 “Configuring a Camera” use case

The GenICam standard defines how to describe a camera’s interface in an abstract way. There is also a free **reference implementation** available that provides the actual implementation of GenICam. Currently, the reference implementation supports only C++ as programming language, but other languages can easily be added.

GenICam requires a **transport layer** as shown in Figure 2. GenICam assumes that the camera has a **register based interface**.

In the “configuring a camera” use case, the transport layer is responsible for providing access to the registers in a camera. That is, the transport layer must provide the *ReadRegister* and *WriteRegister* functions. GenICam in turn is responsible for translating the feature based *Camera.Gain = 42* call to a set of *ReadRegister / WriteRegister* calls to the camera.

The transport layer is provided by the frame grabber / driver vendors. In order to become GenICam aware, vendors must supply a small C++ adapter class that translates the standard *ReadRegister* and *WriteRegister* function calls to the driver specific methods.

Grabbing Images

It is possible to use GenICam solely for the purpose of configuring a camera and use whatever grab interface is appropriate. However, GenICam also provides a standard way to acquire image data (see Figure 3).

The idea is to standardize an abstract interface and control flow for the typical grab sequence. Roughly, it looks like this:

- Get device names (from all transport layers)
- Create camera access object
- Configure camera
- Queue buffers
- Start acquisition
- Wait for buffers

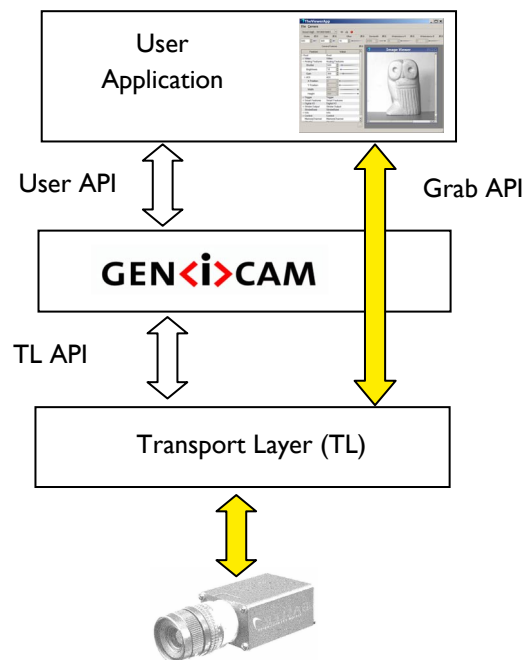


Figure 3 “Grabbing Images” use case

The actual grab interface must be implemented by the transport layer adapter object, which also provides access to the camera’s registers. The GenICam reference implementation supports this by providing an abstract C++ grab interface declaration plus certain services, for example, for registering multiple transport layers, enumerating devices across all transport layers, and instantiating the camera access objects.

Graphical User Interface

The GenICam API provides all of the means necessary to implement a sophisticated, but nevertheless generic, graphical user interface, such as:

- A list of features structured by categories
- All necessary data to feed graphical controls, for example, sliders, drop down boxes, check boxes, push buttons, etc.
- Access mode information such as whether a feature is currently read/write, read only, currently not available, or not implemented at all.
- The ability to register a callback for each feature that will fire if the feature might have changed and needs repainting. The callback capability makes building GUIs extremely easy.

Typically, vendors will implement their own GUI with a distinctive look and feel on top of the GenICam API. Figure 4 shows an example.

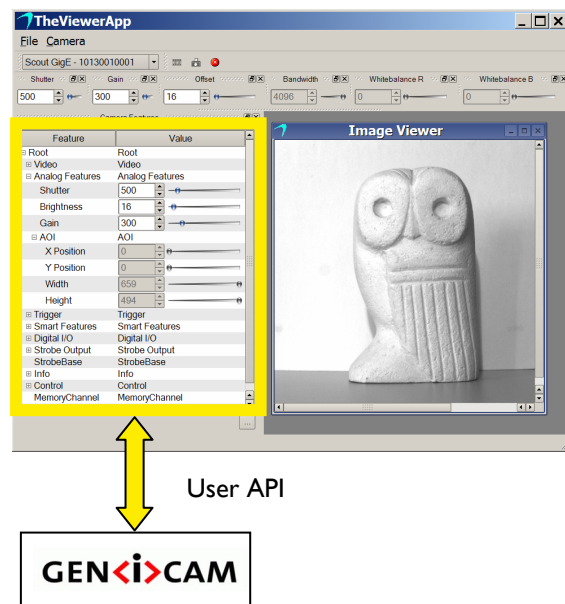


Figure 4 “Graphical User Interface” use case

Dealing with Features

As already mentioned GenICam relies on a **camera description file** that contains a kind of machine readable manual for the camera.

The camera description file is written in **XML format**, and in essence, the GenICam standard only defines the file's **syntax**. A more human readable description of the

syntax is given in the **standard’s text** and a more formal definition of the syntax is given in an **XML schema file**. The latter can be interpreted by most XML editors and provides the user with a syntax check and intellisense-like functions.

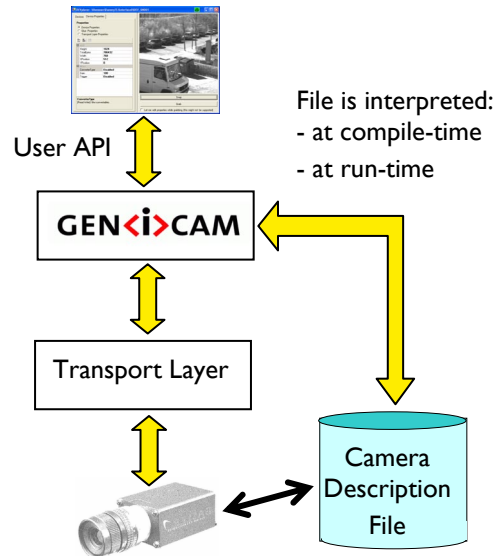


Figure 5 GenICam uses a camera description file

The camera description file is interpreted by the GenICam reference implementation, either at compile time or a run time. A **code generator** can create a C++ camera API that contains exactly the features listed and described in the camera description file. Alternatively, a camera description file can be **interpreted at runtime**. In this case, the user can enumerate the features found in the file and deal with them in a generic fashion, for example, by displaying them in a GUI. It is also possible to create a C++ API for a fixed set of features and to bind at run time to a camera description file loaded on-the-fly. In this case, only those features present in the file appear as implemented. Features in the file that are not present in the API can be also dealt with by enumerating them.

Camera description files are provided by the camera manufacturers. This removes the burden from the software vendors to adapt to each and every feature that different cameras might implement. It also provides multiple software adapters for the same feature because different vendors typically use different register layouts to implement the same feature in their cameras. On the other

hand, camera manufacturers get new features delivered to their customers quickly without needing to ask and wait for support from the library vendors.

Each feature described in the camera configuration file has a **type**. The type is defined by an **abstract interface** that describes what the user can do with a feature of that kind. An integer is for example defined in terms of a value that can be get and set, a minimum, a maximum, and an increment. In addition, the user can ask for the access mode – to check whether the feature is readable, writable and so on – and can convert the integer value from and to a string.

GenICam defines multiple types, some of which are shown in the following list along with the graphic widget they are typically mapped to:

- *Integer, IFloat* ⇔ slider
- *IString* ⇔ edit control
- *IEnumeration* ⇔ drop down box
- *IBoolean* ⇔ check box
- *ICommand* ⇔ command button

To summarize, GenICam lets the user deal with features in an abstract manner and hide all of the details of the mapping between the abstract features and the actual registers of a camera. As a corollary, GenICam can make cameras with the same abstract features, but with different register implementation details, look alike from the user's standpoint.

Feature Namespaces and Standard Feature Lists

Camera vendors are completely free to choose whatever names they like for the features of their cameras. Nevertheless, users are able to retrieve all these features through a generic interface enabling them to write generic software such as a generic GUI. Note, however, that if the names are arbitrary, the **meaning** of the features is unknown to any generic software.

To overcome this, a **list of standard features** is required that defines features in terms of their name, type, and meaning. The use cases described in such a list must cover things like acquisition control, control of the

analog image features, triggering, digital I/O, etc.

Currently, a standard features list for GigE Vision cameras is available and a list for 1394 IIDC cameras is being planned. These lists will heavily overlap, but it will not be possible to make them completely identical.

Using the Reference Implementation

The reference implementation is provided by the members of the GenICam standard group, and it is intended to be used in commercial products. The code is written in C++ and has production quality that is ensured by regression tests with a very good coverage.

GenICam currently supports Win2k/WinXP and MS VisualStudio 7.1/8.0. Linux and the GNU compiler are under preparation.

GenICam is organized in modules:

- **GenApi** : Interprets the camera description file and provides the User API
- **GenTL** : Manages multiple transport layer DLLs, enumerates cameras, and instantiates camera access objects

Each module has a maintainer who ensures code integrity and prepares the releases.

The reference implementation comes in two flavors. The **runtime version** is required for using GenICam in an application, creating camera description files, and creating transport layer adapters. The license for the runtime version is BSD-like; everyone can use it at **no cost** but not modify it. The **source code version** is available for members of the GenICam group only. Everyone can become a member of the group at **no cost**. However, the rules of the group must be obeyed, which ensures that there will be only one well-tested, official version of GenICam.

The GenICam Organization

The GenICam standard is hosted by the European Machine Vision Association (EMVA). Membership in the GenICam group is free and is also open to non-EMVA members.

Signing up as a GenICam member is easy. Just download the registration form from www.genicam.org and read the rules of the GenICam group that are described on the form. Fill out the form, sign it, and fax it to the provided EMVA fax number. You will then

receive a welcome mail that will guide you through the rest of the process.

The GenICam organization also has a strong connection to the GigE Vision standard committee hosted by the Automated Imaging Association (www.machinevisiononline.org).

The GigE Vision camera standard refers to the GenICam standard and states that (a) a GigE Vision compliant camera must provide a camera description file and (b) defines seven mandatory features in terms of GenICam types, names, and meaning. Though it is not mandatory for customers to use GenICam, it is mandatory for camera vendors to make sure that they can if they want to.

A common GenICam file is being planned for 1394 IIDC cameras. Since the IIDC standard

hosted by the 1394 Trade Association (www.1394ta.org) has a fixed register layout, a single camera description file is sufficient for all cameras on the market.

Status and Roadmap

The first version of GenICam standard and its reference implementation is released and available as part of commercial products in the market. The first release concentrates on the GenApi module. The GenTL module is planned for H1/2007.

To get an update on the current status and the roadmap, please visit www.genicam.org.



click. see. smile!

Basler Vision Technologies

Germany Headquarters

Phone +49 4102 463-500

Fax +49 4102 463-599

vc.sales.europe@baslerweb.com

USA

Phone +1 610 280 0171

Fax +1 610 280 7608

vc.sales.usa@baslerweb.com

Asia

Phone +65 6425 0472

Fax +65 6425 0473

vc.sales.asia@baslerweb.com

www.basler-vc.com