What is

?

An Introduction

Table of Contents

HISTORY

Version	Date	Changed by	Change
0.0	09.03.2006	Fritz Dierks, Basler	First Draft
0.1	13.03.2006	Fritz Dierks, Basler	Included comments from Stephane Maurice, Matrox, and Eric Carey, Dalsa/Coreco
0.2	15.03.2006	Tony Piery, Basler	Polished the English
0.2a	17.03.2006	Fritz Dierks, Basler	Added Sick Logo
1.0	06.09.2006	Fritz Dierks, Basler	Updated member list and roadmap after first GenICam release

1 Scope of this Document

This document provides answers to the following questions:

- What is GenICam?
- How does GenICam work?
- How is the GenICam group organized?
- Who is driving GenICam?
- What is GenICam's status and roadmap?
- How can you become part of GenICam?
- What are your benefits from GenICam?

2 GenICam's Key Idea

As shown in Figure 1, GenICam aims to provide a unified application programming interface (API) to the users of machine vision cameras. GenICam is a vendor independent standard and is not bound to a specific interface technology. Currently, GenICam focuses on GigE Vision, 1394 IIDC, Camera Link, and smart cameras, but other interfaces can be supported as well.

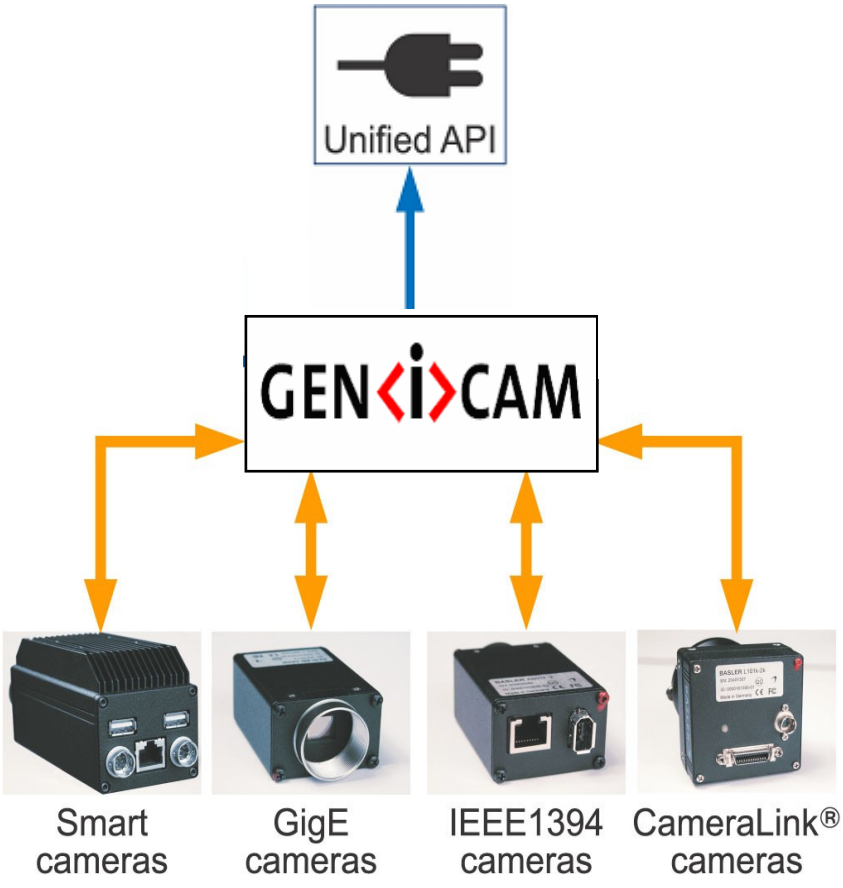

Figure 1 The basic idea of GenICam

GenICam is easy to integrate for **camera** vendors, suppliers of **image processing libraries**, and **frame grabber / driver** providers.

3 GenICam Use Cases

GenICam supports four main use cases:

- Configuring the camera
- Grabbing images
- Providing a user interface
- Transmitting extra image data

3.1 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;
```

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.

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.

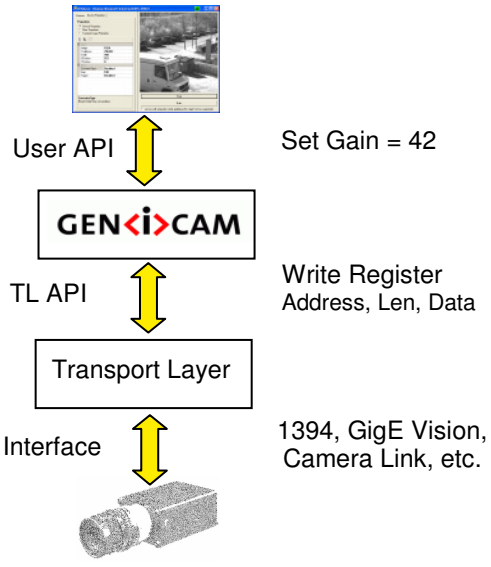

Figure 2 "Configuring a Camera" use case

GenICam requires a **transport layer** as shown in Figure 2. Because this is the native style of both the GigE and the 1394 interfaces, GenICam currently assumes that the camera has a

register based interface. Other interface styles, such as an ASCII command based interface, are being planned.

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.

3.2 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
```

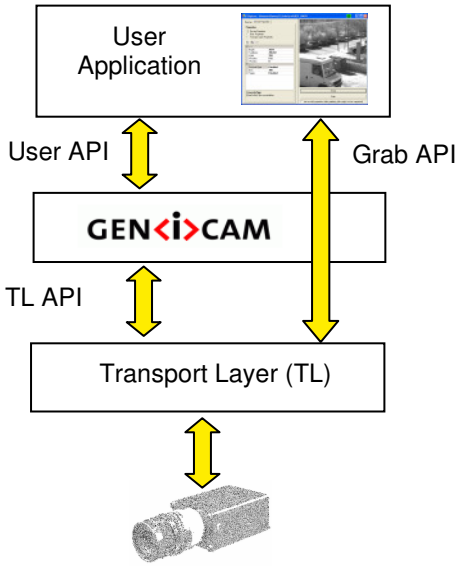

Figure 3 "Grabbing Images" use case

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.

3.3 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. A simple examples shows Figure 4.

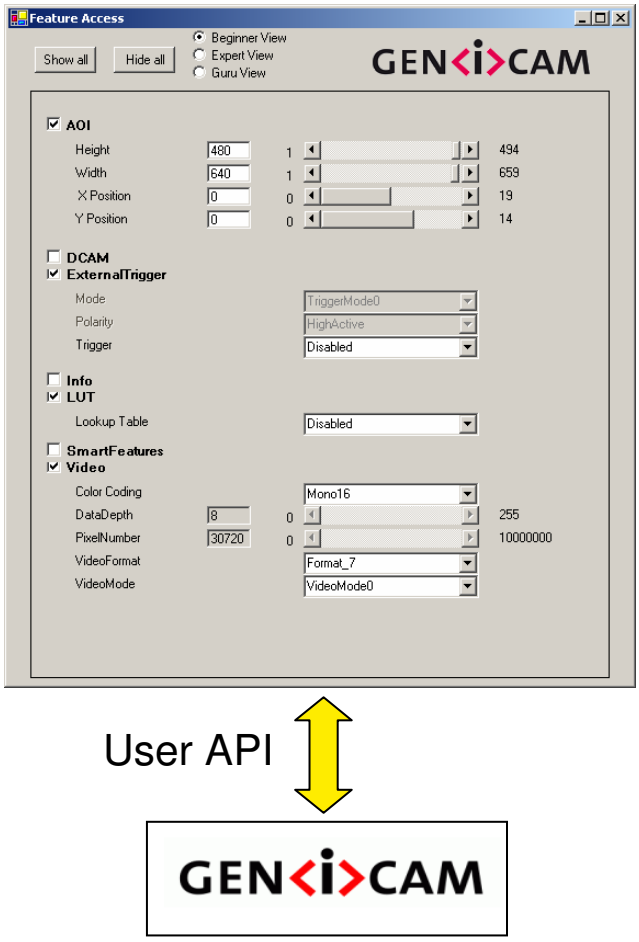

Figure 4 “Graphical User Interface” use case

3.4 Transmitting Extra Image Data

Cameras may send not only image data, but also have additional information attached to the image. Examples would be a frame counter, a trigger time stamp, the current AOI, or even a histogram of the image. In order to pack this additional data along with the image information, the GigE Vision standard defines a **chunk stream format.** This kind of stream type is also widely used with other interfaces such as 1394 IIDC and Camera Link, even though it has not yet been standardized for these technologies.

In chunk stream format, each data item is followed by a trailer that contains the length of the data item and a ChunkID that describes the content of the chunk (see Figure 5). A trailer is used instead of a header because the length of the data item might not be known in advance, for example, for run length coded data.

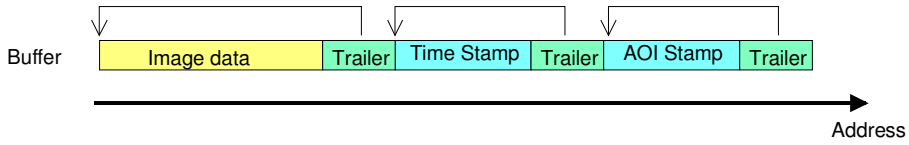

Figure 5 Chunk stream format

The software can walk the chunks, check the ChunkIDs, and process the data. If the software does not know a ChunkID, it simply ignores the corresponding data item. Note that in chunk stream mode, the image data itself also includes a trailer and might even be omitted completely in the buffer.

With GenICam, a user can check whether a certain chunk of data is present in a buffer and read its contents as shown in the following example:

```
if( IsReadable(Camera.TimeStampChunk) )
     cout << Camera.TimeStampChunk();
```

This feature is available for all interface types.

4 Making GenICam Compliant Products

This section gives you an idea of how GenICam works technically and introduces the concept of standard and custom features.

4.1 Dealing with Features

In the “Configuring a camera” use case, GenICam maps high level features like “Gain” to low level registers given in terms of address and length. This is done by using 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.

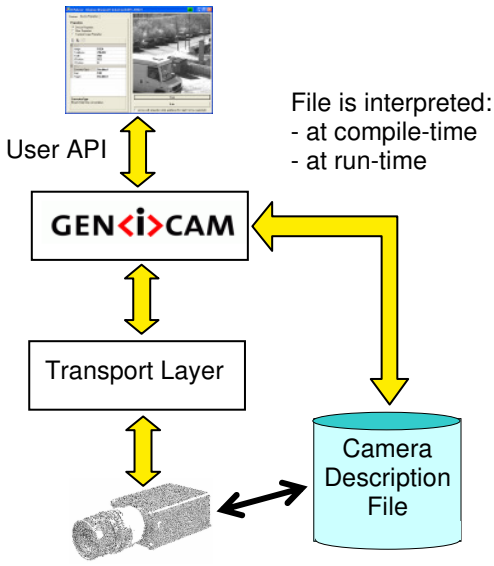

Figure 6 GenICam uses a camera description file

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. For example, Figure 7 shows the *IInteger* interface, which might be used for the *Gain* feature. As you can see, an integer is 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.

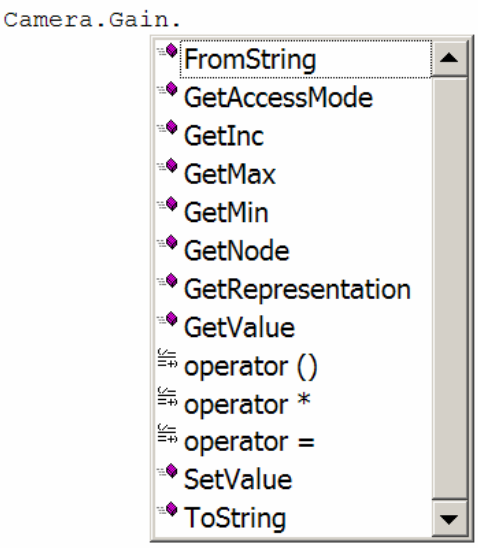

Figure 7 The gain's IInteger interface

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

- *IInteger*, *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.

4.2 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.

To support standard feature lists, GenICam provides the concept of **name spaces,** and there are **standard** name spaces for GigE Vision, 1394 IIDC, and Camera Link cameras. In

addition, there is a **custom** name space where camera manufacturers can choose their own naming conventions. If users check for the presence of a feature, they can use a full qualified name such as “Cust::Gain” or just the name “Gain”. In the former case, the reference implementation will first check the custom name space and then check the selected standard name space. This allows the addition of new features to the standard name spaces while preserving backward compatibility.

4.3 Maintaining Camera Description Files for Multiple Camera Models

Camera manufacturers typically have many camera models and constantly create new features. The question then arises whether the camera vendor must maintain one camera description file per camera model. If some care is taken to make the camera’s interface self describing, the answer is no.

This approach is heavily supported by the 1394 IIDC standard where for each feature there is an inquiry flag that indicates whether the feature is implemented or not. GenICam fully supports these mechanisms, and it is, for example, possible to create a single file for all 1394 IIDC cameras no matter which subset of features a certain camera supports. Of course, the client software must check for each feature before using it to determine whether it is available or not.

For new camera designs, the support of the feature inquiry mechanism is strongly suggested.

5 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.

6 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.

Each GenICam member can choose to be an **associated** or a **contributing member**. All members are placed on the GenICam mailing list, get full access to the source code, and can join GenICam group meetings. The sole difference between contributing and associated members is that contributing members can **vote** and associated members cannot.

Contributing members are working on the standard and the reference implementations. A company becomes contributing by taking "homework" between two meetings. At the beginning of each meeting, the homework is reviewed and accepted. Those companies that have contributed can vote during this meeting. The right to vote must be earned for each meeting. This ensures that enough contributions will be collected to maintain the reference implementation.

Non-members can download the runtime version of GenICam anonymously and can get access to the released standard text. They do not, however, get access to the source code and the mailing list.

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.

Currently, (September 2006) the GenICam group has 28 members, 8 of which are contributing to the collection of the software tests required to ensure the production quality of the code:

Figure 8 28 Members of GenICam as of September 2006

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.

7 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 teh 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.

8 Your Benefits

To summarize, why should you as a vendor of cameras, frame grabbers, drivers, or machine vision software make your products GenICam aware?

- Your maintenance costs will decrease dramatically because integrating libraries, drivers, and cameras becomes easy
- You will reach more customers because new camera features will be accessible with any combination of library and frame grabber / driver

Don't be afraid of competition. You have a good product and GenICam will increase the size of the market for your products.

And why should you as a customer and as a user of machine vision products ask for GenICam aware products?

- You can easily integrate new cameras and access new camera features with your favorite image processing library
- You can use multiple interface technologies and cameras from different vendors in parallel