Allied Vision
a **TKH Vision** brand

PROGRAMMER GUIDE

eGrabber

Programming eGrabber

Document **D234EN-AV eGrabber Programmer Guide-eGrabber-26.02.1.2213** built on **2026-03-10**

Contact Us

Website, email

General

www.alliedvision.com/en/contact
info@alliedvision.com

Distribution partners

www.alliedvision.com/en/avt-locations/avt-distributors

Support

www.alliedvision.com/en/support
www.alliedvision.com/en/about-us/contact-us/technical-support-repair-/-rma

Offices

Europe, Middle East, and Africa (Headquarters)

Allied Vision Technologies GmbH
Taschenweg 2a
07646 Stadtroda, Germany
T// +49 36428 677-0 (Reception)
T// +49 36428 677-230 (Sales)
F// +49 36428 677-28

North, Central, and South America, Canada

Allied Vision Technologies Canada Inc.
300 – 4621 Canada Way
Burnaby, BC V5G 4X8, Canada
T// +1 604 875 8855

Asia-Pacific | China

Allied Vision Technologies Shanghai Co Ltd.
B-510, Venture International Business Park
2679 Hechuan Road
Minhang District, Shanghai 201103
People's Republic of China
T// +86 21 64861133

USA

Allied Vision Technologies, Inc.
102 Pickering Way - Suite 502
Exton, PA 19341, USA
Toll-free// +1-877-USA-1394
T// +1 978 225 2030

Singapore

Allied Vision Technologies Asia Pte. Ltd
82 Playfair Rd, #07-01 D'Lithium
Singapore 368001
T// +65 6634 9027

Japan

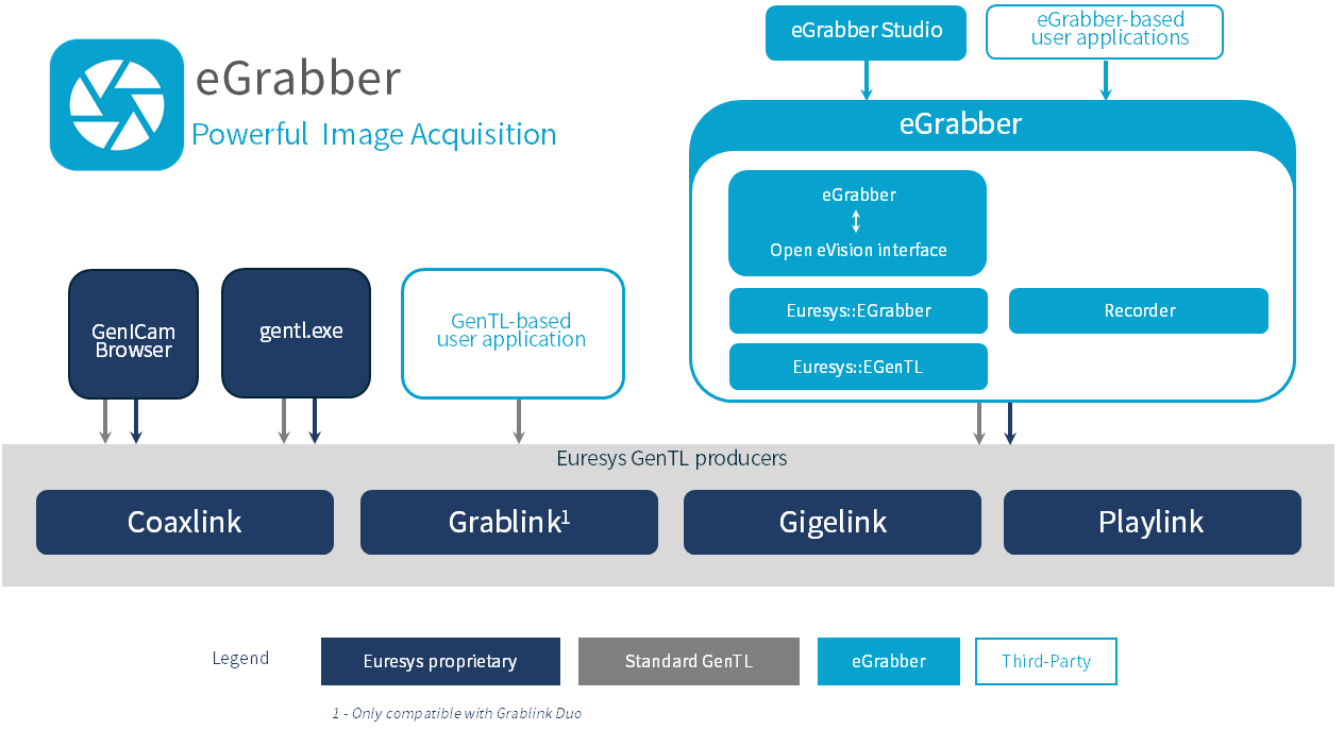
Allied Vision Technologies
Yokohama Portside Bldg. 10F
8-1 Sakae-cho, Kanagawa-ku
Yokohama-shi, Kanagawa, 221-0052
T// +81 (0) 45 577 9527

This documentation is provided with **eGrabber 26.02.1** (doc build **2213**).
https://www.alliedvision.com

Contents

1. Introduction

The *Application Programming Interface* (API) for Coaxlink cards, Grablink Duo, and Gigelink is " eGrabber" on page 12, which is based on GenICam.



The goal of GenICam is to provide a standardized, uniform programming interface for using cameras and frame grabbers based on different physical interfaces (CoaXPress, GigE Vision, etc.) or from different vendors.

GenICam is a set of EMVA standards (GenApi and GenTL), as well as related conventions for naming things (the SFNC for standard features, the PFNC for pixel formats).

- GenApi is about description. At the core of GenApi is the concept of *register description*. Register descriptions are provided in the form of XML files. They map low-level hardware registers to high-level *features*.GenApi allows applications to detect, configure and use the features of cameras and frame grabbers in a uniform and consistent way.
- GenTL is about data transport. The TL suffix stands for *Transport Layer*.The GenTL standard defines a set of C functions and data types for enumerating, configuring, and grabbing images from cameras and frame grabbers. This API is defined by a C header file.Frame grabber vendors provide libraries that implement this API (i.e., libraries that export the functions declared in the standard header file). These libraries are referred to as *GenTL producers*, or *Common Transport Interfaces* (CTI) and use the `cti` file extension. Euresys provides four GenTL producers: `coaxlink.cti` for Coaxlink cards, `grablink.cti` for the Grablink Duo, `gigelink.cti` for GigE Vision cameras, and `playlink.cti` for eGrabber Recorder containers.

This document is meant to be read from beginning to end. Each chapter and section builds upon the preceding ones. If you skip parts of the text, some of the explanations and examples may seem cryptic. If that happens, you should go back and read the parts that you've skipped over.

2. GenApi

GenApi addresses the problem of configuring cameras. The way this is achieved is generic, and applies to different kinds of devices, including frame grabbers. In this chapter, everything we say about cameras also applies to frame grabbers.

GenApi requires two things to work: a *register description*, and a *GenApi implementation*.

Register description

A *register description* is an XML file that can be thought of as a computer-readable datasheet of the camera. It defines camera settings (such as `PixelFormat` and `TriggerSource`), and instructions on how to configure them (e.g., to set `ExposureMode` to `Timed`, write value `0x12` to register `0xE0140`). It can also contain camera documentation.

GenApi implementation

A *GenApi implementation* is a software module that can read and interpret register description files.

The EMVA provides a reference implementation, but it is fairly difficult to use, and logging is very poor. Instead, we recommend using the Euresys implementation bundled with the eGrabber software package. This implementation also allows writing powerful " Euresys GenApi scripts" on page 29 and provides " Euresys GenApi Extensions" on page 41 accessible as virtual GenApi features.

Features

What the user gets from GenApi is a bunch of named *features*, organized in categories.

Set/get features

Features are simple settings of different data types such as:

- integer (e.g., Width)
- float (e.g., AcquisitionFrameRate)
- enumeration (e.g., PixelFormat)
- boolean (e.g., LUTEnable)
- string (e.g., DeviceVendorName)

The value of features can be retrieved/modified using *get/set* functions. Some features are read-only and some are write-only, but most allow read/write access.

Commands

There is also another kind of features: *commands* (e.g., `AcquisitionStart`). Commands are special: they don't have any associated value; they have side effects. Command features are meant to be *executed*. When a command is executed, some action happens in the camera (e.g., a software trigger is generated). Obviously, *get/set* functions don't make sense for commands and can't be used.

3. GenTL

GenTL defines 5 types of objects, organized in a parent/child relationship:

1. the system module
2. the interface module
3. the device module
4. the data stream module
5. the buffer module

Each module:

- corresponds to a particular element of the system;
- defines relevant pieces of information (info commands) that can be queried (using get info functions);
- allows exercizing that module's functionality (using specific functions).

Additionally, all modules except the buffer module behave as *ports* that allow read/write operations. These port functions are used by " GenApi" on page 6 to load that module's description file, and to use its GenApi features.

System module

The system module (also referred to as *TLSystem*), represents the GenTL producer (e.g., the `coaxlink.cti` library). This module is at the top of the parent/child tree.

The system module provides basic information about the GenTL producer: things like the complete path to the CTI file and the vendor name (Euresys).

The real point of the system module is to list the " Interface module" on page 8 (or frame grabbers) present in the system. The most important functions of the system module are `TLGetNumInterfaces` (to retrieve the number of frame grabbers in the system) and `TLOpenInterface` (to get access to one of the frame grabbers).

Interface module

The GenTL standard calls frame grabbers *interfaces*. The system module has one child interface for each frame grabber: if there are 2 Coaxlink cards in the computer, the system module will have two child interfaces.

Each interface represents a frame grabber (or a network interface card). Global frame grabber features such as digital I/O lines belong in the interface module. This means that the " GenApi" on page 6 features controlling the I/O lines are attached to the interface.

Each interface also acts as parent to one or several " Device module" on page 9. The most important functions of the interface module are `IFGetNumDevices` (to retrieve the number of cameras that can be connected to the interface) and `IFOpenDevice` (to get access to one of the devices).

Device module

The GenTL standard uses the terms *device* and *remote device* for two related but different concepts. A *remote device* is a real camera, physically connected to a frame grabber (or network interface card). This is different from the device module we describe here.

The device module is the module that contains the frame grabber settings relating to the camera. This includes things like triggers and strobes.

The device module also acts as parent to one or several " Data stream module" on page 9, and can be viewed as the sibling of the *remote device*. The most important functions of the device module are `DevOpenDataStream` (to get access to one of the data streams) and `DevGetPort` (to get access to the remote device).

Data stream module

The data stream module handles " Buffer module" on page 9. During acquisition runs, images are sent from the camera to the frame grabber, which transfers them to memory buffers allocated on the host computer. The data stream module is where image acquisition occurs. It is where most of the functionality resides.

Buffer handling is very flexible. Any number of buffers can be used. Buffers are either in the input queue, in the output queue, or temporarily unqueued. The application decides when empty buffers are queued (to the input FIFO), and when filled buffers are popped (from the output FIFO).

Buffer module

The buffer module simply represents a memory buffer given to a parent data stream. Useful metadata is associated to buffers. This includes the image width, height, pixel format, timestamp… These are retrieved through *info commands* (see `BUFFER_INFO_CMD_LIST` in the standard GenTL header file).

The buffer module is the only module that doesn't have read/write port functions; it doesn't have GenApi features.

GenTL API

GenTL makes it possible to detect, control and use all camera and frame grabber features, but its usage is tedious:

- cti files must be dynamically loaded, and the functions they export must be accessed through pointers.
- Functions return an error code that must be checked by the application.
- Most functions read from/write to untyped buffers: the application must determine the required buffer size, allocate a temporary buffer, convert data to/from this buffer, and finally release the buffer memory.

Instead of using the GenTL API directly, we recommend using either:

- the EGenTL C++ class which deals with these complications so that the user doesn't have to;
- or the eGrabber library which provides a high-level, easy-to-use interface.

4. EGenTL

Euresys::EGenTL is a C++ class that provides the same functionality as standard GenICam GenTL, but with a more user-friendly interface. For example, it uses std::string instead of raw char pointers, and error codes are transformed into exceptions. Euresys::EGenTL also takes care of locating the GenTL producer and loading the functions it exports.

To use Euresys::EGenTL, simply include the relevant header file[1]:

```
#include <EGenTL.h>
```

Instead of the raw, low-level C functions that GenTL defines, we get a Euresys::EGenTL object that represents the GenTL producer:

- Each GenTL function is available as a member method of Euresys::EGenTL. GenTL function names start with an upper-case prefix. In Euresys::EGenTL, method names start with the same prefix, but written in lower-case. For example, the GCReadPort function is exposed as the gcReadPort method, and the TLOpenInterface function as the tlOpenInterface method.
- All GenTL functions return a GC_ERROR code indicating success or failure. When a function returns a code other than GC_ERR_SUCCESS, an exception is thrown. This removes the burden of manually checking error codes after each function call.
- Since GenTL functions return a GC_ERROR, output values are returned through pointers passed as arguments. Euresys::EGenTL methods offer a more natural interface; they return the output value directly:GC_API TLGetNumInterfaces(TL_HANDLE hTL, uint32_t *piNumIfaces);uint32_t tlGetNumInterfaces(TL_HANDLE tlh);(Note that GC_API is defined as GC_IMPORT_EXPORT GC_ERROR GC_CALLTYPE. It is simply a GC_ERROR decorated with calling convention and DLL import/export attributes.)
- For GenTL functions that deal with text, the corresponding Euresys::EGenTL methods convert from char * to std::string and vice-versa:GC_API TLGetInterfaceID(TL_HANDLE hTL, uint32_t iIndex, char *sID, size_t *piSize);std::string tlGetInterfaceID(TL_HANDLE tlh, uint32_t index);
- Some GenTL functions retrieve information about the camera or frame grabber. These functions fill a void * buffer with a value, and indicate in an INFO_DATATYPE the actual type of the value. Euresys::EGenTL uses C++ templates to make these functions easy to use:GC_API GCGetInfo(TL_INFO_CMD iInfoCmd, INFO_DATATYPE *piType, void *pBuffer, size_t *piSize);template<typename T> T gcGetInfo(TL_INFO_CMD cmd);

A first example

This program uses Euresys::EGenTL to iterate over the Coaxlink cards present in the system, and display their id:

```
#include <iostream>
#include <EGenTL.h>                                    // 1

void listCards() {
```

[1] On Windows, the application must be linked with Kernel32.lib.

On Linux, the application must be linked with -ldl and -lpthread or preferably compiled and linked with -pthread.

```
    Euresys::EGenTL gentl;                                          // 2
    GenTL::TL_HANDLE tl = gentl.tlOpen();                           // 3
    uint32_t numCards = gentl.tlGetNumInterfaces(tl);               // 4
    for (uint32_t n = 0; n < numCards; ++n) {
        std::string id = gentl.tlGetInterfaceID(tl, n);             // 5
        std::cout << "[" << n << "] " << id << std::endl;
    }
}

int main() {
    try {                                                           // 6
        listCards();
    } catch (const std::exception &e) {                             // 6
        std::cout << "error: " << e.what() << std::endl;
    }
}
```

1. Include `EGenTL.h`, which contains the definition of the `Euresys::EGenTL` class.
2. Create a `Euresys::EGenTL` object. This involves the following operations:
   - locate and dynamically load the GenTL producer (e.g., coaxlink.cti);
   - retrieve pointers to the functions exported by the GenTL producer, and make them available via Euresys::EGenTL methods;
   - initialize the GenTL producer (this is done by calling the GenTL initialization function GCInitLib).
3. Open the GenTL producer. This returns a handle of type `GenTL::TL_HANDLE`. The `GenTL` namespace is defined in the standard GenTL header file, which has been automatically included by `EGenTL.h` in step 1.
4. Find out how many cards are present in the system.
5. Retrieve the id of the n-th card.
6. `Euresys::EGenTL` uses exceptions to report errors, so we wrap our code inside a `try ... catch` block.

Example of program output:

```
[0] PC1633 - Coaxlink Quad G3 (1-camera, line-scan) - KQG00014
[1] PC1632 - Coaxlink Quad (1-camera) - KQU00031
```

Relevant files

include/EGenTL.h	Main header. Includes all the other headers. Defines `Euresys::EGenTL`.
include/GenTL_v1_5.h	Standard GenTL header. Defines standard types, functions and constants.
include/GenTL_EuresysCustom.h	Defines eGrabber-specific constants.

5. eGrabber

eGrabber is a library of C++ classes that provide a high-level interface. It is built on top of "EGenTL" on page 10, and is recommended for most users.

A .NET assembly, built on top of eGrabber, is also provided. In this document, we focus mainly on the C++ API. Minor differences between the C++ and .NET interfaces are listed in " .NET assembly" on page 55.

Python bindings are also available for eGrabber. Again, the differences between the C++ and Python interfaces are listed in " Python" on page 59.

To use the classes described here, you need to include the main eGrabber file:

```
#include <EGrabber.h>
```

eGrabber[1] comprises several classes, the most important of which is `Euresys::EGrabber`:

```
namespace Euresys {
    class EGrabber;
}
```

In this text, we'll refer to this class as a *grabber*. A grabber encapsulates a set of related GenTL modules:

- An interface: the module that represents global frame grabber settings and features. This includes digital I/O control, PCIe and firmware status…
- A device (or local device, as opposed to remote device): the module that contains the frame grabber settings and features relating to the camera. This consists mainly of camera and illumination control features: strobes, triggers…
- A data stream: the module that handles image buffers.
- A remote device: the camera.
- A number of buffers.

Go back to " GenTL" on page 8 if these concepts are not clear.

A first example

This example creates a grabber and displays basic information about the interface, device, and remote device modules it contains:

```
#include <iostream>
#include <EGrabber.h>                                                    // 1

static const uint32_t CARD_IX = 0;
static const uint32_t DEVICE_IX = 0;

void showInfo() {
```

[1] On Windows, the application must be linked with `Kernel32.lib`.

On Linux, the application must be linked with `-ldl` and `-lpthread` or preferably compiled and linked with `-pthread`.

```
    Euresys::EGenTL gentl;                                                          // 2
    Euresys::EGrabber<> grabber(gentl, CARD_IX, DEVICE_IX);                         // 3

    std::string card = grabber.getString<Euresys::InterfaceModule>("InterfaceID");  // 4
    std::string dev = grabber.getString<Euresys::DeviceModule>("DeviceID");         // 5
    int64_t width = grabber.getInteger<Euresys::RemoteModule>("Width");             // 6
    int64_t height = grabber.getInteger<Euresys::RemoteModule>("Height");           // 6

    std::cout << "Interface:    " << card << std::endl;
    std::cout << "Device:       " << dev  << std::endl;
    std::cout << "Resolution:   " << width << "x" << height << std::endl;
}

int main() {
    try {                                                                           // 7
        showInfo();
    } catch (const std::exception &e) {                                             // 7
        std::cout << "error: " << e.what() << std::endl;
    }
}
```

1. Include `EGrabber.h`, which defines the `Euresys::EGrabber` class, and includes the other header files we need (such as `EGenTL.h` and the standard GenTL header file).
2. Create a `Euresys::EGenTL` object. This involves the following operations:
   - locate and dynamically load the GenTL producer (e.g., coaxlink.cti);
   - retrieve pointers to the functions exported by the GenTL producer, and make them available via Euresys::EGenTL methods;
   - initialize the GenTL producer (this is done by calling the GenTL initialization function GCInitLib).
3. Create a `Euresys::EGrabber` object. The constructor needs the `gentl` object created in step 2. It also takes as optional arguments the indices of the interface and device to use.The purpose of the angle brackets (<>) that come after `EGrabber` will become clear later. For now, they can be safely ignored.
4. Use " GenApi" on page 6 to query the ID of the interface. `Euresys::InterfaceModule` indicates that we want an answer from the " Interface module" on page 8.
5. Similarly, query the ID of the device. This time, we use `Euresys::DeviceModule` to target the " Device module" on page 9.
6. Finally, read the camera resolution. `Euresys::RemoteModule` indicates that the value must be retrieved from the camera.
7. eGrabber uses exceptions to report errors, so we wrap our code inside a `try ... catch` block.

Example of program output:

```
Interface:    PC1633 - Coaxlink Quad G3 (2-camera) - KQG00014
Device:       Device0
Resolution:   4096x4096
```

Discovering grabbers and cameras

Using indices

In the previous example, we created a grabber object using the indices of the interface (`CARD_IX`) and device (`DEVICE_IX`) to use:

```
Euresys::EGrabber<> grabber(gentl, CARD_IX, DEVICE_IX);
```

Those indices are optional and are set to `0` by default. We have seen earlier that GenTL modules are organized as a tree structure where the root is the system module (please refer to " GenTL" on page 8 for details). In short, the system module maintains a list of interfaces present in the system, each interface maintains a list of devices associated to that interface, and each device maintains a list of available data streams. So, using 3 indices, we can identify a specific data stream within the GenTL module hierarchy. Each index represents the position of an element in 3 successive lists (interface, device and stream lists).

The GenTL standard defines two functions to update those lists:

- TLUpdateInterfaceList to update the list of available interfaces in the system;
- IFUpdateDeviceList to update the list of available devices connected to a specific interface.

Please note that the GenTL standard mandates that those lists cannot change between calls to their corresponding update functions. In other words, when `TLUpdateInterfaceList` is called, a “snapshot” of the present interfaces is taken and maintained in the GenTL library until the next call. The same applies to `IFUpdateDeviceList` for the available devices of a specific interface.

To simplify the creation of a grabber object, the `EGrabber` constructor calls the “update list” functions automatically so that the indices given as arguments refer to the current state of the system.

Those constructors are fine and easy to use when the system hierarchy is fixed, i.e. when it does not change dynamically.

Coaxlink and Grablink cards are fixed interfaces in the system and those interfaces have a fixed number of devices that depends on the selected firmware. Calling several times the “update list” functions from the same process on such fixed systems will lead to the same results; therefore an application can safely use the indices to identify specific modules and create grabbers.

However, this simple approach may lead to trouble on systems that change dynamically. The Gigelink producer is a system that uses network interfaces to find and establish connections to GigE Vision cameras. Obviously, calling `IFUpdateDeviceList` on such systems will lead to completely different results depending on what’s available on the network when the function is invoked. In such environments, referring to a specific interface, device, or data stream using indices might not be applicable.

To solve this issue, and to provide a simple way to discover possible grabber objects and connected cameras in the system, we provide the `Euresys::EGrabberDiscovery` module.

Using EGrabberDiscovery

The `Euresys::EGrabberDiscovery` module provides a function `discover` that scans the system to detect available grabbers and cameras. Here is an example:

```
#include <EGrabber.h>
#include <iostream>

static void discover() {
    Euresys::EGenTL gentl;
    Euresys::EGrabberDiscovery discovery(gentl);                // 1

    discovery.discover();                                       // 2
    for (int i = 0; i < discovery.egrabberCount(); ++i) {       // 3
        Euresys::EGrabber<> grabber(discovery.egrabbers(i));    // 4
        // ...
    }
    for (int i = 0; i < discovery.cameraCount(); ++i) {         // 5
        Euresys::EGrabber<> grabber(discovery.cameras(i));      // 6
        // ...
    }
}

int main() {
    try {
        discover();
    } catch (const std::exception &e) {
        std::cout << "error: " << e.what() << std::endl;
    }
}
```

1. Create a `Euresys::EGrabberDiscovery` object for a specific producer (`gentl` argument). The object takes exclusive control of the producer interface and device lists. As long as the `discovery` object exists, it prevents any other module from updating the lists.
2. Update the producer interface and device lists and scan the system to discover the available GenTL modules as well as the cameras that are connected at that moment.
3. Walk through the discovered grabbers.
4. For each discovered grabber, create a `Euresys::EGrabber` object; the method `discovery.egrabbers(n)` returns the information that identifies the n-th grabber (0-based index).
5. Walk through the discovered cameras.
6. For each discovered camera, create a `Euresys::EGrabber` object; the method `discovery.cameras(n)` returns the information that identifies the n-th camera (0-based index).

Remarks

- `Euresys::EGrabberDiscovery` provides two ways of exploring the system:
  - a grabber-oriented discovery, using egrabberCount and egrabbers;
  - a camera-oriented discovery, using cameraCount and cameras.

- The camera-oriented discovery is strongly recommended in the following cases:
  - When an application needs to configure and/or use the cameras connected to the system and can simply ignore grabbers without cameras.
  - When a multi-bank camera is connected to the system. A multi-bank camera is a camera composed of several sub-Devices (which act as independent CoaXPress Devices). In this case, each camera bank is connected to a specific GenTL device and exposes a data stream. The camera-oriented discovery is able3 to:
    - automatically detect the N banks that make up the multi-bank camera, and determine the connection topology;
    - configure DMA transfers so that the data coming from the N banks are directly transferred where they belong in the destination buffer in order to reassemble the full image with no extra copy;
    - expose the N grabbers as a unified camera that can be easily instantiated as a Euresys::EGrabber object (itself composed of N sub-grabbers).

    The resulting Euresys::EGrabber object hides the complexity of the banks and behaves as a usual EGrabber object as far as acquisition is concerned.
- The number of grabbers discovered can be greater than the number of cameras discovered, for example because no cameras are connected to a frame grabber, or because a multi-bank camera is connected to several frame grabbers.
- As long as a `Euresys::EGrabberDiscovery` object exists, any attempt to update interface or device lists will trigger a `not_allowed` exception; this will happen if a `Euresys::EGrabber` object is created using the index-based constructor instead of the “discovery” variants because the index-based constructor updates the interface and device lists before opening the requested GenTL modules.

Please refer to the `Euresys::EGrabberDiscovery` module documentation for a detailed description of the API.

Acquiring images

This program uses eGrabber to acquire images from a camera connected to a Coaxlink card:

```
#include <iostream>
#include <EGrabber.h>

void grab() {
    Euresys::EGenTL gentl;
    Euresys::EGrabber<> grabber(gentl);                                    // 1

    grabber.reallocBuffers(3);                                             // 2
    grabber.start(10);                                                     // 3
    for (size_t i = 0; i < 10; ++i) {
        Euresys::ScopedBuffer buf(grabber);                                // 4
        void *ptr = buf.getInfo<void *>(GenTL::BUFFER_INFO_BASE);          // 5
        uint64_t ts = buf.getInfo<uint64_t>(GenTL::BUFFER_INFO_TIMESTAMP); // 6
        std::cout << "buffer address: " << ptr << ", timestamp: "
                  << ts << " us" << std::endl;
    }                                                                      // 7
}

int main() {
    try {
        grab();
    } catch (const std::exception &e) {
```

```
        std::cout << "error: " << e.what() << std::endl;
    }
}
```

1. Create a `Euresys::EGrabber` object. The second and third arguments of the constructor are omitted here. The grabber will use the first device of the first interface present in the system.
2. Allocate 3 buffers. The grabber automatically determines the required buffer size.
3. Start the grabber. Here, we ask the grabber to fill 10 buffers. If we don't want the grabber to stop after a specific number of buffers, we can do `grabber.start(GenTL::GENTL_INFINITE)`, or simply `grabber.start()`.Starting the grabber involves the following operations:
   - the DSStartAcquisition function is called to start the data stream.
   - the AcquisitionStart command is executed on the camera;

   In this example, we assume that the camera and frame grabber are properly configured. For a real application, it would be safer to run a " Euresys GenApi scripts" on page 29 before starting acquisitions (and before allocating buffers for that matter). This will be shown in another example.
4. Wait for a buffer filled by the grabber. The result is a `Euresys::ScopedBuffer`. The term *scoped* is used to indicate that the lifetime of the buffer is the current scope (i.e., the current block).
5. Retrieve the buffer address. This is done by calling the `getInfo` method of the buffer. This method takes as argument a `BUFFER_INFO_CMD`. In this case, we request the `BUFFER_INFO_BASE`, which is defined in the standard GenTL header file:enum BUFFER_INFO_CMD_LIST{ BUFFER_INFO_BASE = 0, /* PTR Base address of the buffer memory. */ BUFFER_INFO_SIZE = 1, /* SIZET Size of the buffer in bytes. */ BUFFER_INFO_USER_PTR = 2, /* PTR Private data pointer of the GenTL Consumer. */ BUFFER_INFO_TIMESTAMP = 3, /* UINT64 Timestamp the buffer was acquired. */ // ... // other BUFFER_INFO definitions omitted // ... BUFFER_INFO_CUSTOM_ID = 1000 /* Starting value for GenTL Producer custom IDs. */};typedef int32_t BUFFER_INFO_CMD;Notice that `getInfo` is a template method, and when we call it we must specify the type of value we expect. `BUFFER_INFO_BASE` returns a pointer; this is why we use `getInfo<void *>`.
6. Do the same to retrieve the timestamp of the buffer. This time, we use the `uint64_t` version of `getInfo` to match the type of `BUFFER_INFO_TIMESTAMP`.Note that timestamps are always 64-bit and expressed as the number of microseconds that have elapsed since the computer was started.
7. We reach the end of the `for` block. The local variable `buf` gets out of scope and is destroyed: the `ScopedBuffer` destructor is called. This causes the GenTL buffer contained in `buf` to be re-queued (given back) to the data stream of the grabber.

Example of program output:

```
buffer address: 0x7f3c32c54010, timestamp: 11247531003686 us
buffer address: 0x7f3c2c4bf010, timestamp: 11247531058080 us
buffer address: 0x7f3c2c37e010, timestamp: 11247531085003 us
buffer address: 0x7f3c32c54010, timestamp: 11247531111944 us
buffer address: 0x7f3c2c4bf010, timestamp: 11247531137956 us
buffer address: 0x7f3c2c37e010, timestamp: 11247531163306 us
buffer address: 0x7f3c32c54010, timestamp: 11247531188600 us
buffer address: 0x7f3c2c4bf010, timestamp: 11247531213807 us
buffer address: 0x7f3c2c37e010, timestamp: 11247531239158 us
buffer address: 0x7f3c32c54010, timestamp: 11247531265053 us
```

We can see that the three buffers that were allocated (let's call them A at `0x7f3c32c54010`, B at `0x7f3c2c4bf010`, and C at `0x7f3c2c37e010`) are used in a round-robin fashion: A → B → C → A → B → C → ⋯ This is the result of:

- the FIFO nature of input and output buffer queues:
  - the data stream pops a buffer from the front of the input queue and transfers data to that buffer;
  - when the transfer is complete, the buffer is pushed to the back of the output queue;
- the use of ScopedBuffer:
  - the ScopedBuffer constructor pops a buffer from the front of the output queue (i.e., it takes the oldest buffer);
  - the ScopedBuffer destructor pushes that buffer to the back of the input queue (hence, this buffer will be used for a new transfer after all buffers already in the input queue).

Configuring the grabber

Configuration is a very important aspect of any image acquisition program.

- The camera and the frame grabber both have to be configured according to the application requirements.
- The camera configuration must be compatible with the frame grabber configuration, and vice versa.

Configuration basically boils down to a series of " GenApi" on page 6 and/or " GenApi" on page 6 operations performed on the grabber modules: the remote device (i.e., the camera), the " Interface module" on page 8, the " Device module" on page 9, or the " Data stream module" on page 9 modules.

This program configures the grabber for the so-called *RG* control mode (asynchronous reset camera control, frame grabber-controlled exposure).

```
#include <iostream>
#include <EGrabber.h>

const double FPS = 150;

void configure() {
    Euresys::EGenTL gentl;
    Euresys::EGrabber<> grabber(gentl);
    // camera configuration
    grabber.setString<Euresys::RemoteModule>("TriggerMode", "On");                      // 1
    grabber.setString<Euresys::RemoteModule>("TriggerSource", "CXPin");                 // 2
    grabber.setString<Euresys::RemoteModule>("ExposureMode", "TriggerWidth");           // 3
    // frame grabber configuration
    grabber.execute<Euresys::DeviceModule>("DeviceReset");                              // 4
    grabber.setString<Euresys::DeviceModule>("CameraControlMethod", "RG");              // 5
    grabber.setString<Euresys::DeviceModule>("CycleTriggerSource", "Immediate");        // 6
    grabber.setFloat<Euresys::DeviceModule>("CycleMinimumPeriod", 1e6 / FPS);           // 7
}

int main() {
    try {
        configure();
    } catch (const std::exception &e) {
        std::cout << "error: " << e.what() << std::endl;
    }
}
```

1. Enable triggers on the camera.
2. Tell the camera to look for triggers on the CoaXPress link.

3. Configure the camera to use the `TriggerWidth` exposure mode.
4. Execute the `DeviceReset` command to restore default settings on the device module.
5. Set the frame grabber's camera control method to `RG`. In this mode, camera cycles are initiated by the frame grabber, and the exposure duration is also controlled by the frame grabber.
6. Tell the frame grabber to initiate camera cycles itself (at a rate defined by `CycleMinimumPeriod`), without waiting for hardware or software triggers.
7. Configure the frame rate.

But there is a better way to configure the grabber. Using a " Euresys GenApi scripts" on page 29, the program becomes:

```
#include <iostream>
#include <EGrabber.h>

void configure() {
    Euresys::EGenTL gentl;
    Euresys::EGrabber<> grabber(gentl);
    grabber.runScript("config.js");
}

int main() {
    try {
        configure();
    } catch (const std::exception &e) {
        std::cout << "error: " << e.what() << std::endl;
    }
}
```

and the configuration script is:

```
var grabber = grabbers[0];
var FPS = 150;
// camera configuration
grabber.RemotePort.set("TriggerMode", "On");
grabber.RemotePort.set("TriggerSource", "CXPin");
grabber.RemotePort.set("ExposureMode", "TriggerWidth");
// frame grabber configuration
grabber.DevicePort.execute("DeviceReset");
grabber.DevicePort.set("CameraControlMethod", "RG");
grabber.DevicePort.set("CycleTriggerSource", "Immediate");
grabber.DevicePort.set("CycleMinimumPeriod", 1e6 / FPS);
```

Using a script file has several advantages:

- The configuration can be changed without recompiling the application. This allows shorter development cycles, and makes it possible to update the configuration in the lab or in the field.
- The configuration script can be loaded by eGrabber Studio and the command-line `gentl` tool. This makes is possible to validate the configuration outside of the user application.
- The configuration script can easily be shared by several applications written in different programming languages: C++, C#, Python…
- The full power of " Euresys GenApi scripts" on page 29 is available.

Events

Background

Euresys GenTL producers can generate different kinds of events.

GenTL standard events

- New buffer events: events indicating that a buffer has been filled by a data stream.
- Remote device events: events sent by the remote device; e.g., GenApi-compatible events.

Custom events

- Data stream events: events related to a data stream and its frame store; e.g., start of camera readout, end of camera readout…
- Camera and illumination controller events: events related to the real-time control (performed by a device) of a camera and its illumination devices; e.g., start of camera trigger, end of camera trigger, start of light strobe…
- CoaXPress device events: events (coming from the device) related to a CoaXPress camera; e.g., reception of a link trigger from a CoaXPress camera.
- Device error events: events related to errors in the device.
- I/O toolbox events: events (coming from the interface) related to digital I/O lines and other I/O tools; e.g., line input tools, quadrature decoder tools, divider tools…
- CoaXPress interface events: events (also coming from the interface) related to the CoaXPress interface; e.g., CoaXPress link configuration, CRC errors…

Counters

Coaxlink and Grablink firmware count each occurrence of each custom event and make these counters available in a GenApi feature named `EventCount`. Each event has its own counter, and the value of `EventCount` depends on the selected event:

```
// select the CameraTriggerRisingEdge event
grabber.setString<DeviceModule>("EventSelector", "CameraTriggerRisingEdge");
// read the value of the counter
int64_t counter = grabber.getInteger<DeviceModule>("EventCount");
```

or, using the *selected feature* notation:

```
// read the value of the CameraTriggerRisingEdge counter
int64_t counter = grabber.getInteger<DeviceModule>("EventCount[CameraTriggerRisingEdge]");
```

Notifications

As we've just seen, when an event occurs, a dedicated counter is incremented. The GenTL producer can also notify the application of this event by having `Euresys::EGrabber` execute a user-defined " eGrabber" on page 12. But first, it is required to enable notifications of one or more events:

```
grabber.setString<DeviceModule>("EventSelector", "CameraTriggerRisingEdge");
grabber.setInteger<DeviceModule>("EventNotification", true);
grabber.setString<DeviceModule>("EventSelector", "CameraTriggerFallingEdge");
grabber.setInteger<DeviceModule>("EventNotification", true);
...
```

or:

```
grabber.setInteger<DeviceModule>("EventNotification[CameraTriggerRisingEdge]", true);
grabber.setInteger<DeviceModule>("EventNotification[CameraTriggerFallingEdge]", true);
...
```

Using a " Euresys GenApi scripts" on page 29, it is easy to enable notifications for all events:

```
function enableAllEvents(p) {                          // 1
    var events = p.$ee('EventSelector');               // 2
    for (var e of events) {
        p.set('EventNotification[' + e + ']', true);   // 3
    }
}

var grabber = grabbers[0];
enableAllEvents(grabber.InterfacePort);                // 4
enableAllEvents(grabber.DevicePort);                   // 5
enableAllEvents(grabber.StreamPort);                   // 6
```

1. Define a helper function named `enableAllEvents` and taking as argument a module (or port) `p`.
2. Use the `$ee` function to retrieve the list of values `EventSelector` can take. This is the list of events generated by module `p`. (ee stands for *enum entry*.)
3. For each event, enable notifications. (The `+` operator concatenates strings, so if `e` is `'LIN1'`, the expression `'EventNotification[' + e + ']'` evaluates to `'EventNotification[LIN1]'`.)
4. Call the `enableAllEvents` function defined in step 1 for the interface module. This will enable notifications for all events in the *I/O toolbox* and *CoaXPress interface* categories.
5. Likewise, enable notifications for all events coming from the device module (*CIC* events, *CoaXPress device events*, *device error* events).
6. Finally, enable notifications for all *data stream* events.

Callback functions

When an event occurs, and event notification is enabled for that event, `Euresys::EGrabber` executes one of several callback functions.

These callback functions are defined in overridden virtual methods:

```
class MyGrabber : public Euresys::EGrabber<>
{
    public:
        ...

    private:
        // callback function for new buffer events
        virtual void onNewBufferEvent(const NewBufferData& data) {
            ...
        }

        // callback function for remote device events
```

```
        virtual void onRemoteDeviceEvent(const RemoteDeviceData &data) {
            ...
        }

        // callback function for data stream events
        virtual void onDataStreamEvent(const DataStreamData &data) {
            ...
        }

        // callback function for Camera and Illumination controller events
        virtual void onCicEvent(const CicData &data) {
            ...
        }

        // callback function for CoaXPress device events
        virtual void onCxpDeviceEvent(const CxpDeviceData &data) {
            ...
        }

        // callback function for device error events
        virtual void onDeviceErrorEvent(const DeviceErrorData &data) {
            ...
        }

        // callback function for I/O toolbox events
        virtual void onIoToolboxEvent(const IoToolboxData &data) {
            ...
        }

        // callback function for CoaXPress interface events
        virtual void onCxpInterfaceEvent(const CxpInterfaceData &data) {
            ...
        }
};
```

As you can see, a different callback function can be defined for each category of events.

In .NET, callback functions are defined by creating delegates rather than overriding virtual methods. An example will be given in " .NET assembly" on page 55.

Event identification

When an event is notified to the application, the callback function that is executed indicates the category of that event. For custom events, the actual event that occurred is identified by a numerical ID, called `numid`, and defined in `include/GenTL_EuresysCustom.h`:

```
enum EVENT_DATA_NUMID_CUSTOM_LIST
{
    // EVENT_CUSTOM_IO_TOOLBOX
    EVENT_DATA_NUMID_IO_TOOLBOX_LIN1                    = ...  /* Line Input Tool 1 */
    EVENT_DATA_NUMID_IO_TOOLBOX_LIN2                    = ...  /* Line Input Tool 2 */
    EVENT_DATA_NUMID_IO_TOOLBOX_MDV1                    = ...  /* Multiplier/Divider Tool 1 */
    ...
    // EVENT_CUSTOM_CXP_INTERFACE
    EVENT_DATA_NUMID_CXP_INTERFACE_CRC_ERROR_CXP_A      = ...  /* Detected CRC error on CXP connector A */
    ...
    // EVENT_CUSTOM_CIC
    EVENT_DATA_NUMID_CIC_CAMERA_TRIGGER_RISING_EDGE     = ...  /* Start of camera trigger */
    EVENT_DATA_NUMID_CIC_CAMERA_TRIGGER_FALLING_EDGE    = ...  /* End of camera trigger */
    EVENT_DATA_NUMID_CIC_STROBE_RISING_EDGE             = ...  /* Start of light strobe */
    EVENT_DATA_NUMID_CIC_STROBE_FALLING_EDGE            = ...  /* End of light strobe */
    ...
```

```
    // EVENT_CUSTOM_CXP_DEVICE
    EVENT_DATA_NUMID_CXP_DEVICE_LINK_TRIGGER           = ...  /* LinkTrigger<N> received from CoaXPress device
*/
    ...
    // EVENT_CUSTOM_DATASTREAM
    EVENT_DATA_NUMID_DATASTREAM_START_OF_CAMERA_READOUT = ...  /* Start of camera readout */
    EVENT_DATA_NUMID_DATASTREAM_END_OF_CAMERA_READOUT   = ...  /* End of camera readout */
    ...
    // EVENT_CUSTOM_DEVICE_ERROR
    ...
};
```

For reference, the following table lists the relationships between:

- the module generating events
- the category of events
- the name of the callback function
- the data type passed to the callback function
- the common numid prefix

Module	Category	Callback function	Data type	numid prefix
Data stream	New Buffer	onNewBufferEvent	NewBufferData	*not defined for GenTL standard events*
Device	Remote Device	onRemoteDeviceEvent	RemoteDeviceData	*not defined for GenTL standard events*
Data stream	Data Stream	onDataStreamEvent	DataStreamData	EVENT_DATA_NUMID_DATASTREAM_
Device	CIC	onCicEvent	CicData	EVENT_DATA_NUMID_CIC_
Device	CXP Device	onCxpDeviceEvent	CxpDeviceData	EVENT_DATA_NUMID_CXP_DEVICE_
Device	Device Error	onDeviceErrorEvent	DeviceErrorData	EVENT_DATA_NUMID_DEVICE_ERROR_
Interface	I/O Toolbox	onIoToolboxEvent	IoToolboxData	EVENT_DATA_NUMID_IO_TOOLBOX_
Interface	CXP Interface	onCxpInterfaceEvent	CxpInterfaceData	EVENT_DATA_NUMID_CXP_INTERFACE_

Note that a simple naming scheme is followed: a category of events named *“some category”* has a callback function named on***SomeCategory***Event which takes as argument a ***SomeCategory***Data structure, and uses EVENT_DATA_NUMID_***SOME_CATEGORY***_ as common numid prefix.

Examples

We’ll soon show a few complete " Events and callbacks examples" on page 25 illustrating events and callbacks, but there is one more thing we need to explain before we can do that: the context in which callback functions are executed. This is the subject of the "EGrabber flavors" on page 24.

EGrabber flavors

When should the callback functions be called? From which context (i.e., which thread)? Thinking about these questions leads to the definition of several *callback models*:

- The application asks the grabber: *"Do you have any buffer or event for me? If yes, execute my callback function now."* This is a polling, synchronous mode of operation, where callbacks are executed when the application demands it, in the application thread.We'll refer to this callback model as *on demand*.
- The application asks the grabber to create a single, dedicated thread, and to wait for events in this thread. When an event occurs, the grabber executes the corresponding callback function, also in this single callback thread.We'll refer to this callback model as *single thread*.
- The application asks the grabber to create a dedicated thread for each of its callback functions. In each of these threads, the grabber waits for a particular category of events. When an event occurs, the corresponding callback function is executed in that thread.We'll refer to this callback model as *multi thread*.

These three callback models all make sense, and each one is best suited for some applications.

- The *on demand* model is the simplest. Although its implementation may use worker threads, from the point of view of the user it doesn't add any thread. This means that the application doesn't need to worry about things such as thread synchronization, mutexes, etc.
- If the user wants a dedicated callback thread, the *single thread* model creates it for him.When we have only one thread, things are simple. In this model, the grabber is used in (at least) two threads, so we need to start worrying about synchronization and shared data.
- In the *multi thread* model, each category of event gets its own thread. The benefit of this is that events of one type (and the execution of their callback functions) don't delay notifications of events of other types. For example, a thread doing heavy image processing in `onNewBufferEvent` will not delay the notification of CIC events by `onCicEvent` (e.g., events indicating that the exposure is complete and that the object or camera can be moved).In this model, the grabber is used in several thread, so the need for synchronization is present as it is in the *single thread* model.Of course, the application must also be aware that it might receive notifications for events of type *X* that are older than notifications of events of type *Y* that have already been received and processed. After all, this is the differentiating factor between the *single thread* and *multi thread* models.

To give the user maximum flexibility, we support all three callback models. This is why `Euresys::EGrabber` exists in different flavors. So far, we have eluded the meaning of the angle brackets in `EGrabber<>`. The `EGrabber` class is actually a *template class*, i.e., a class that is parameterized by another type:

- In this case, the template parameter is the callback model to use: one of `CallbackOnDemand`, `CallbackSingleThread` or `CallbackMultiThread`.
- The empty <> are used to select the default template parameter, which is `CallbackOnDemand`.
- The types of grabber that can be instantiated are:
  - EGrabber<CallbackOnDemand>
  - EGrabber<CallbackSingleThread>
  - EGrabber<CallbackMultiThread>
  - EGrabber<> which is equivalent to EGrabber<CallbackOnDemand>

- The eGrabber header file also defines the following type aliases (synonyms):typedef EGrabber<CallbackOnDemand> EGrabberCallbackOnDemand;typedef EGrabber<CallbackSingleThread> EGrabberCallbackSingleThread;typedef EGrabber<CallbackMultiThread> EGrabberCallbackMultiThread;
- In other programming languages, *single thread* and *multi thread* models are best implemented with native language features (e.g., *tasks* in .NET). This is why the eGrabber " .NET assembly" on page 55 and " Python" on page 59 only implement the *on demand* model.

Events and callbacks examples

On demand callbacks

This program displays basic information about CIC events generated by a grabber:

```
#include <iostream>
#include <EGrabber.h>

using namespace Euresys;                                                    // 1

class MyGrabber : public EGrabber<CallbackOnDemand> {                       // 2
    public:
        MyGrabber(EGenTL &gentl) : EGrabber<CallbackOnDemand>(gentl) {      // 3
            runScript("config.js");                                         // 4
            enableEvent<CicData>();                                         // 5
            reallocBuffers(3);
            start();
        }

    private:
        virtual void onCicEvent(const CicData &data) {
            std::cout << "timestamp: " << std::dec << data.timestamp << " us, " // 6
                      << "numid: 0x" << std::hex << data.numid              // 6
                      << " (" << getEventDescription(data.numid) << ")"
                      << std::endl;
        }
};

int main() {
    try {
        EGenTL gentl;
        MyGrabber grabber(gentl);
        while (true) {
            grabber.processEvent<CicData>(1000);                            // 7
        }
    } catch (const std::exception &e) {
        std::cout << "error: " << e.what() << std::endl;
    }
}
```

1. This *using* directive allows writing `Xyz` instead of `Euresys::Xyz`. This helps keep lines relatively short.
2. Define a new class `MyGrabber` which is derived from `EGrabber<CallbackOnDemand>`.
3. `MyGrabber`'s constructor initializes its base class by calling `EGrabber<CallbackOnDemand>`'s constructor.

4. Run a `config.js` script which should:
   - properly configure the camera and frame grabber;
   - enable notifications for CIC events.
5. Enable `onCicEvent` callbacks.
6. The `onCicEvent` callback function receives a `const CicData &`. This structure is defined in `include/EGrabberTypes.h`. It contains a few pieces of information about the event that occurred. Here, we display the `timestamp` and `numid` of each event. The " eGrabber" on page 12 indicates which CIC event occurred.
7. Call `processEvent<CicData>(1000)`:
   - the grabber starts waiting for a CIC event;
   - if an event occurs within 1000 ms, the grabber executes the onCicEvent callback function;
   - otherwise, a timeout exception will be thrown.

Example of program output:

```
timestamp: 1502091779 us, numid: 0x8041 (Start of camera trigger)
timestamp: 1502091784 us, numid: 0x8048 (Received acknowledgement for previous CXP trigger message)
timestamp: 1502091879 us, numid: 0x8043 (Start of light strobe)
timestamp: 1502092879 us, numid: 0x8044 (End of light strobe)
timestamp: 1502097279 us, numid: 0x8042 (End of camera trigger)
timestamp: 1502097284 us, numid: 0x8048 (Received acknowledgement for previous CXP trigger message)
timestamp: 1502191783 us, numid: 0x8041 (Start of camera trigger)
timestamp: 1502191783 us, numid: 0x8045 (CIC is ready for next camera cycle)
timestamp: 1502191788 us, numid: 0x8048 (Received acknowledgement for previous CXP trigger message)
timestamp: 1502191883 us, numid: 0x8043 (Start of light strobe)
timestamp: 1502192883 us, numid: 0x8044 (End of light strobe)
timestamp: 1502197283 us, numid: 0x8042 (End of camera trigger)
timestamp: 1502197288 us, numid: 0x8048 (Received acknowledgement for previous CXP trigger message)
timestamp: 1502291788 us, numid: 0x8041 (Start of camera trigger)
timestamp: 1502291788 us, numid: 0x8045 (CIC is ready for next camera cycle)
...
```

Single thread and multi thread callbacks

This program displays basic information about CIC events generated by a grabber, this time using the `CallbackSingleThread` model.

```
#include <iostream>
#include <EGrabber.h>

using namespace Euresys;

class MyGrabber : public EGrabber<CallbackSingleThread> {                       // 1
    public:
        MyGrabber(EGenTL &gentl) : EGrabber<CallbackSingleThread>(gentl) {      // 2
            runScript("config.js");
            enableEvent<CicData>();
            reallocBuffers(3);
            start();
        }

    private:
        virtual void onCicEvent(const CicData &data) {
            std::cout << "timestamp: " << std::dec << data.timestamp << " us, "
                      << "numid: 0x" << std::hex << data.numid
```

```
                    << " (" << getEventDescription(data.numid) << ")"
                    << std::endl;
        }
};

int main() {
    try {
        EGenTL gentl;
        MyGrabber grabber(gentl);
        while (true) {                                                  // 3
        }
    } catch (const std::exception &e) {
        std::cout << "error: " << e.what() << std::endl;
    }
}
```

There are very few differences between this program and the " eGrabber" on page 12 version:

1. `MyGrabber` is derived from `EGrabber<CallbackSingleThread>` instead of `EGrabber<CallbackOnDemand>`.
2. Consequently, `MyGrabber`'s constructor initializes its base class by calling `EGrabber<CallbackSingleThread>`'s constructor.
3. `EGrabber` creates a callback thread in which it calls `processEvent`, so we don't have to. Here, we simply enter an infinite loop.

As you can see, moving from `CallbackOnDemand` to `CallbackSingleThread` is very simple. If instead you want the `CallbackMultiThread` variant, simply change the base class of `MyGrabber` to `EGrabber<CallbackMultiThread>` (and call the appropriate constructor).

New buffer callbacks

This program shows how to access information related to *new buffer* events. It uses `CallbackMultiThread`, but it could use another callback method just as well.

```
#include <iostream>
#include <EGrabber.h>

using namespace Euresys;

class MyGrabber : public EGrabber<CallbackMultiThread> {
    public:
        MyGrabber(EGenTL &gentl) : EGrabber<CallbackMultiThread>(gentl) {
            runScript("config.js");
            enableEvent<NewBufferData>();
            reallocBuffers(3);
            start();
        }

    private:
        virtual void onNewBufferEvent(const NewBufferData &data) {
            ScopedBuffer buf(*this, data);                              // 1
            uint64_t ts = buf.getInfo<uint64_t>(gc::BUFFER_INFO_TIMESTAMP);     // 2
            std::cout << "event timestamp: " << data.timestamp << " us, "       // 3
                      << "buffer timestamp: " << ts << " us" << std::endl;
        }                                                               // 4
};

int main() {
    try {
        EGenTL gentl;
```

```
        MyGrabber grabber(gentl);
        while (true) {
        }
    } catch (const std::exception &e) {
        std::cout << "error: " << e.what() << std::endl;
    }
}
```

1. In onNewBufferEvent, create a temporary ScopedBuffer object buf. The ScopedBuffer constructor takes two arguments:
   - the grabber owning the buffer: since we are in a class derived from EGrabber, we simply pass *this;
   - information about the buffer: this is provided in data.
2. Retrieve the timestamp of the buffer, which is defined as the time at which *the camera started to send data to the frame grabber*. Note that Euresys::gc is an alias for the standard GenTL C++ namespace.
3. As explained in the section about " eGrabber" on page 12, *new buffer* events are slightly different from the other kinds of events: they are standard (as per GenTL), and don't have an associated numid.As a consequence, the NewBufferData structure passed to onNewBufferEvent doesn't have a numid field. It does, however, have a timestamp field indicating the time at which *the driver was notified that data transfer to the buffer was complete*. This *event timestamp* is inevitably greater than the *buffer timestamp* retrieved in step 2.
4. We reach the end of the block where the local variable buf has been created. It gets out of scope and is destroyed: the ScopedBuffer destructor is called. This causes the GenTL buffer contained in buf to be re-queued (given back) to the data stream of the grabber.

Example of program output:

```
event timestamp: 77185931621 us, buffer timestamp: 77185919807 us
event timestamp: 77185951618 us, buffer timestamp: 77185939809 us
event timestamp: 77185971625 us, buffer timestamp: 77185959810 us
event timestamp: 77185991611 us, buffer timestamp: 77185979812 us
event timestamp: 77186011605 us, buffer timestamp: 77185999808 us
event timestamp: 77186031622 us, buffer timestamp: 77186019809 us
event timestamp: 77186051614 us, buffer timestamp: 77186039810 us
event timestamp: 77186071611 us, buffer timestamp: 77186059811 us
event timestamp: 77186091602 us, buffer timestamp: 77186079812 us
event timestamp: 77186111607 us, buffer timestamp: 77186099814 us
```

Relevant files

include/EGrabber.h	Main header. Includes all the other headers except include/FormatConverter.h. Defines Euresys::EGrabber, Euresys::Buffer, Euresys::ScopedBuffer.
include/EGrabberDiscovery.h	Defines Euresys::EGrabberDiscovery.
include/EGrabberTypes.h	Defines data types related to Euresys::EGrabber.
include/EGenTL.h	Defines Euresys::EGenTL.
include/GenTL_v1_5.h	Standard GenTL header. Defines standard types, functions and constants.
include/GenTL_EuresysCustom.h	Defines eGrabber-specific constants.
include/FormatConverter.h	Defines Euresys::FormatConverter helper class.

6. Euresys GenApi scripts

The Euresys GenApi Script language is documented in a few GenApi scripts. For convenience, they are also included here.

doc/basics.js

```
// Euresys GenApi Script uses a syntax inspired by JavaScript, but is not
// exactly JavaScript. Using the extension .js for scripts is just a way to get
// proper syntax highlighting in text editors.
//
// This file describes the basics of Euresys GenApi Script. It can be executed
// by running 'gentl script <path-to-coaxlink-scripts-dir>/doc/basics.js', or
// more simply 'gentl script egrabber://doc/basics.js'. It can also be executed
// by entering 'g.runScript egrabber://doc/basics.js' in eGrabber Studio's
// Terminal.


// Euresys GenApi Script is case-sensitive.

// Statements are always separated by semicolons (JavaScript is more
// permissive).

// Single-line comment

/* Multi-line comment
   (cannot be nested)
*/

// Function declaration
function multiply(a, b) {
    return a * b;
}

// Functions can be nested
function sumOfSquares(a, b) {
    function square(x) {
        return x * x;
    }
    return square(a) + square(b);
}

// Variable declaration
function Variables() {
    var x = 1;      // 1
    var y = 2 * x;  // 2
    var z;          // undefined
}

// Data types
function DataTypes() {
    // Primitive types: Boolean, Number, String, undefined
    function Booleans() {
        var x = true;
        var y = false;
    }
    function Numbers() {
```

```
        var x = 3.14159;
        var y = -1;
        var z = 6.022e23;
    }
    function Strings() {
        var empty = "";
        var x = "euresys";
        var y = 'coaxlink';
        var z = x + ' ' + y; // euresys coaxlink
        assertEqual(x, z.slice(0, 7));
        assertEqual(y, z.slice(8));
    }
    function Undefined() {
        // undefined is the type of variables without a value
        // undefined is also a special value
        var x; // undefined
        x = 1; // x has a value
        x = undefined; // x is now again undefined
    }
    // Objects: Object (unordered set of key/value pairs), Array (ordered list
    // of values), RegExp (regular expression)
    function Objects() {
        // Construction
        var empty = {};
        var x = { a: 1, b: 2, c: 3 };
        var y = { other: x };
        // Access to object properties
        var sum1 = x.a + x.b + x.c; // dot notation
        var sum2 = x['a'] + x['b'] + x['c']; // bracket notation
        // Adding properties
        x.d = 4;    // x: { a: 1, b: 2, c: 3, d: 4 }
        x["e"] = 5; // x: { a: 1, b: 2, c: 3, d: 4, e: 5 }
    }
    function Arrays() {
        // Construction
        var empty = [];
        var x = [3.14159, 2.71828];
        var mix = [1, false, "abc", {}];
        // Access to array elements
        var sum = x[0] + x[1]; // bracket notation
        // Adding elements
        x[2] = 1.61803; // x: [3.14159, 2.71828, 1.61803]
        x[4] = 1.41421; // x: [3.14159, 2.71828, 1.61803, undefined, 1.41421];
    }
    function RegularExpressions() {
        var x = /CXP[36]_X[124]/;
    }
    Booleans();
    Numbers();
    Strings();
    Undefined();
    Objects();
    Arrays();
}

// Like JavaScript, Euresys GenApi Script is a dynamically typed language. The
// type of a variable is defined by the value it holds, which can change.
function DynamicVariables() {
    var x = 1; // Number
    x = "x is now a string";
}

// Object types are accessed by reference.
function References() {
    var x = [3.14159, 2.71828]; // x is a reference to an array
```

```
    var y = x;                  // y is a reference to the same array
    assertEqual(x.length, y.length);
    assertEqual(x[0], y[0]);
    assertEqual(x[1], y[1]);
    y[2] = 1.61803;             // the array can be modified via any reference
    assertEqual(x[2], y[2]);

    function update(obj) {
        // objects (including arrays) are passed by reference
        obj.updated = true;
        obj.added = true;
    }
    var z = { initialized: true, updated: false };
    assertEqual(true, z.initialized);
    assertEqual(false, z.updated);
    assertEqual(undefined, z.added);
    update(z);
    assertEqual(true, z.initialized);
    assertEqual(true, z.updated);
    assertEqual(true, z.added);
}

// Supported operators
function Operators() {
    // From lowest to highest precedence:
    // Assignment operators: = += -= *= /=
    var x = 3;
    x += 2;
    x -= 1;
    x *= 3;
    x /= 5;
    assertEqual(2.4, x);
    // Logical OR: ||
    assertEqual(true, false || true);
    assertEqual('ok', false || 'ok');
    assertEqual('ok', 'ok' || 'ignored');
    // Logical AND: &&
    assertEqual(false, true && false);
    assertEqual(true, true && true);
    assertEqual('ok', true && 'ok');
    // Identity (strict equality) and non-identity (strict inequality): === !==
    assertEqual(true, 1 === 2 / 2);
    assertEqual(true, 1 !== 2);
    // Equality (==) and inequality (!=) JavaScript operators lead to confusing
    // and inconsistent conversions of their operands. They are not implemented
    // in Euresys GenApi Script.
    // Relational operators: < <= > >=
    assert(1 < 2);
    assert(1 <= 1);
    assert(2 > 1);
    assert(2 >= 2);
    // Addition and subtraction: + -
    assertEqual(1, 3 - 2);
    assertEqual(5, 2 + 3);
    assertEqual("abcdef", "abc" + "def"); // if one of the operands is of type
    assertEqual("abc123", "abc" + 123);   // string, all operands are converted
    assertEqual("123456", 123 + "456");   // to string, and concatenated
    // Multiplication and division: * /
    assertEqual(4.5, 3 * 3 / 2);
    // Prefix operators: ++ -- ! typeof
    var x = 0;
    assertEqual(1, ++x);
    assertEqual(1, x);
    assertEqual(0, --x);
    assertEqual(0, x);
```

```
    assertEqual(true, !false);
    assertEqual('boolean', typeof false);
    assertEqual('number', typeof 0);
    assertEqual('string', typeof '');
    assertEqual('undefined', typeof undefined);
    assertEqual('function', typeof function () {});
    assertEqual('object', typeof {});
    assertEqual('object', typeof []);
    assertEqual('object', typeof /re/);
    assertEqual('object', typeof null);
    // Postfix operators: ++ --
    var x = 0;
    assertEqual(0, x++);
    assertEqual(1, x);
    assertEqual(1, x--);
    assertEqual(0, x);
    // Function call: ()
    assertEqual(6, multiply(3, 2));
    assertEqual(13, sumOfSquares(3, 2));
    // Member access: . []
    var obj = { a: 1 };
    assertEqual(1, obj.a);
    obj['4'] = 'four';
    assertEqual('four', obj[2*2]);
}

// Scope of variables
function OuterFunction() {
    var x = 'outer x';
    function Shadowing() {
        assertEqual(undefined, x);
        var x = 'inner x';
        assertEqual('inner x', x);
    }
    function Nested() {
        assertEqual('outer x', x);
        var y = 'not accessible outside Nested';
        x += ' changed in Nested';
    }
    function NoBlockScope() {
        var x = 1;
        assertEqual(1, x);
        if (true) {
            // The scope of variables is the function.
            // This variable x is the same as the one outside the if block.
            var x = 2;
        }
        assertEqual(2, x);
    }
    assertEqual('outer x', x);
    Shadowing();
    assertEqual('outer x', x);
    Nested();
    assertEqual('outer x changed in Nested', x);
    NoBlockScope();
}

// Loops
function Loops() {
    // for loops
    function ForLoops() {
        var i;
        var sum = 0;
        for (i = 0; i < 6; ++i) {
            sum += i;
```

```
        }
        assertEqual(15, sum);
    }
    // for..in loops: iterating over indices
    function ForInLoops() {
        var xs = [1, 10, 100, 1000];
        var sum = 0;
        for (var i in xs) {
            sum += xs[i];
        }
        assertEqual(1111, sum);
        var obj = { one: 1, two: 2 };
        var sum = 0;
        for (var p in obj) {
            sum += obj[p];
        }
        assertEqual(3, sum);
        var str = "Coaxlink";
        var sum = "";
        for (var i in str) {
            sum += str[i];
        }
        assertEqual("Coaxlink", sum);
    }
    // for..of loops: iterating over values
    function ForOfLoops() {
        var xs = [1, 10, 100, 1000];
        var sum = 0;
        for (var x of xs) {
            sum += x;
        }
        assertEqual(1111, sum);
        var obj = { one: 1, two: 2 };
        var sum = 0;
        for (var x of obj) {
            sum += x;
        }
        assertEqual(3, sum);
        var str = "Coaxlink";
        var sum = "";
        for (var c of str) {
            sum += c;
        }
        assertEqual("Coaxlink", sum);
    }
    function ContinueAndBreak() {
        var i;
        var sum = 0;
        for (i = 0; i < 100; ++i) {
            if (i === 3) {
                continue;
            } else if (i === 6) {
                break;
            } else {
                sum += i;
            }
        }
        assertEqual(0 + 1 + 2 + 4 + 5, sum);
    }
    ForLoops();
    ForInLoops();
    ForOfLoops();
    ContinueAndBreak();
}
```

```
function Exceptions() {
    var x;
    var caught;
    var finallyDone;
    function f(action) {
        x = 0;
        caught = undefined;
        finallyDone = false;
        try {
            x = 1;
            if (action === 'fail') {
                throw action;
            } else if (action === 'return') {
                return;
            }
            x = 2;
        } catch (e) {
            // Executed if a throw statement is executed.
            assertEqual(1, x);
            caught = e;
        } finally {
            // Executed regardless of whether or not a throw statement is
            // executed. Also executed if a return statement causes the
            // function to exit before the end of the try block.
            finallyDone = true;
        }
    }
    f('fail');
    assertEqual(1, x);
    assertEqual('fail', caught);
    assert(finallyDone);
    f('return');
    assertEqual(1, x);
    assert(!caught);
    assert(finallyDone);
    f();
    assertEqual(2, x);
    assert(!caught);
    assert(finallyDone);
}

function CharCode() {
    assertEqual(0x41, "ABC".charCodeAt());
    assertEqual(0x42, "ABC".charCodeAt(1));
    assertEqual(0x43, "ABC".charCodeAt(2));
    assertEqual("A", String.fromCharCode(0x41));
    assertEqual("ABC", String.fromCharCode(0x41, 0x42, 0x43));
}



// Run tests
Variables();
DynamicVariables();
DataTypes();
References();
Operators();
OuterFunction();
Loops();
Exceptions();
CharCode();

function assertEqual(expected, actual) {
    if (expected !== actual) {
        throw 'expected: ' + expected + ', actual: ' + actual;
```

```
    }
}

function assert(condition) {
    if (!condition) {
        throw 'failed assertion';
    }
}
```

doc/builtins.js

```
// This file describes the builtins (functions or objects) of Euresys GenApi
// Script. It can be executed by running 'gentl script
// egrabber://doc/builtins.js'. It can also be executed by entering
// 'g.runScript egrabber://doc/builtins.js' in eGrabber Studio's Terminal.

// The builtin object 'console' contains a function 'log' which can be
// used to output text to the standard output (if available) as well as to
// Memento with the notice verbosity level.
console.log('Hello from ' + module.filename);
console.log('If several arguments are passed,', 'they are joined with spaces');
console.log('All text is sent to both standard output and Memento');

// The builtin object 'memento' contains the following functions: error,
// warning, notice, info, debug, verbose (each corresponding to a different
// verbosity level in Memento). They are similar to console.log, except that
// the text is only sent to Memento.
memento.error('error description');
memento.warning('warning description');
memento.notice('important notification');
memento.info('message');
memento.debug('debug information');
memento.verbose('more debug information');

// For convenience, the object 'console' also contains the same methods as the
// object 'memento'; the functions are similar to their 'memento' counterparts,
// except that they also send text to the standard output if available
console.error('error description');
console.warning('warning description');
console.notice('important notification');
console.info('message');
console.debug('debug information');
console.verbose('more debug information');

// Explicit type conversion/information functions:
console.log('Boolean(0) = ' + Boolean(0));               // false
console.log('Boolean(3) = ' + Boolean(3));               // true
console.log('Number(false) = ' + Number(false));         // 0
console.log('Number(true) = ' + Number(true));           // 1
console.log('Number("3.14") = ' + Number("3.14"));       // 3.14
console.log('Number("0x16") = ' + Number("0x16"));       // 22
console.log('Number("1e-9") = ' + Number("1e-9"));       // 0.000000001
console.log('String(false) = ' + String(false));         // "false"
console.log('String(true) = ' + String(true));           // "true"
console.log('String(3.14) = ' + String(3.14));           // "3.14"
console.log('String([1, 2]) = ' + String([1, 2]));       // "1,2"
console.log('isNaN(0/0) = ' + isNaN(0/0));               // true
console.log('isNaN(Infinity) = ' + isNaN(Infinity));     // false
console.log('isRegExp(/re/) = ' + isRegExp(/re/));       // true
console.log('isRegExp("/re/") = ' + isRegExp("/re/"));   // false
console.log('Array.isArray({}) = ' + Array.isArray({})); // false
console.log('Array.isArray([]) = ' + Array.isArray([])); // true
```

```
// The builtin object 'Math' contains a few functions:
console.log('Math.floor(3.14) = ' + Math.floor(3.14));   // 3
console.log('Math.ceil(3.14) = ' + Math.ceil(3.14));     // 4
console.log('Math.abs(-1.5) = ' + Math.abs(-1.5));       // 1.5
console.log('Math.pow(2, 5) = ' + Math.pow(2, 5));       // 32
console.log('Math.log2(2048) = ' + Math.log2(2048));     // 11

// String manipulation
console.log('"Duo & Duo".replace(/Duo/,  "Quad") = "' +
            "Duo & Duo".replace(/Duo/,  "Quad") + '"'); // "Quad & Duo"
console.log('"Duo & Duo".replace(/Duo/g, "Quad") = "' +
            "Duo & Duo".replace(/Duo/g, "Quad") + '"'); // "Quad & Quad"
console.log('"Hello, Coaxlink".toLowerCase() = "' +
            "Hello, Coaxlink".toLowerCase() + '"');     // "hello, coaxlink"
console.log('"Coaxlink Quad G3".includes("Quad") = ' +
            "Coaxlink Quad G3".includes("Quad"));       // true
console.log('"Coaxlink Quad".includes("G3") = ' +
            "Coaxlink Quad".includes("G3"));            // false
console.log('"Coaxlink Quad G3".split(" ") = [' +
            "Coaxlink Quad G3".split(" ") + ']');       // [Coaxlink,Quad,G3]
console.log('"Coaxlink Quad G3".split("Quad") = [' +
            "Coaxlink Quad G3".split("Quad") + ']');    // [Coaxlink , G3]
console.log('["Mono", "Duo", "Quad"].join() = "' +
            ["Mono", "Duo", "Quad"].join() + '"');      // "Mono,Duo,Quad"
console.log('["Mono", "Duo", "Quad"].join(" & ") = "' +
            ["Mono", "Duo", "Quad"].join(" & ") + '"'); // "Mono & Duo & Quad"

// Utility functions
console.log('random(0,1): ' + random(0,1)); // random number between 0 and 1
sleep(0.5); // pause execution of script for 0.5 seconds

// The builtin function 'require' loads a script, executes it, and returns
// the value of the special 'module.exports' from that module.
var mod1 = require('./module1.js');
console.log('mod1.description: ' + mod1.description);
console.log('mod1.plus2(3): ' + mod1.plus2(3));
console.log('calling mod1.hello()...');
mod1.hello();

// 'require' can deal with:
// - absolute paths
//   var mod = require('C:\\absolute\\path\\some-module.js');
// - relative paths (paths relative to the current script)
//   var mod = require('./utils/helper.js');
// - egrabber:// paths (paths relative to the directory where eGrabber scripts
//   are installed)
//   var mod = require('egrabber://doc/builtins.js');
// - custom:// paths (path relative to the eGrabber configuration directory)
//   var mod = require('custom://cameras/manual.js');
for (var p of ['./utils/helper.js', 'egrabber://doc/builtins.js',
               'custom://cameras/manual.js']) {
    console.log(p + ' -> ' + system.resolve(p));
}
```

doc/grabbers.js

```
// This file describes the 'grabbers' object of Euresys GenApi Script. It can
// be executed by running 'gentl script egrabber://doc/grabbers.js'. It can
// also be executed by entering 'g.runScript egrabber://doc/grabbers.js' in
// eGrabber Studio's Terminal.

// The builtin object 'grabbers' is a list of objects giving access to the
// available GenTL modules/ports.
```

```
// In most cases, 'grabbers' contains exactly one element, but there are two
// exceptions:
// - When using a multi-bank camera, 'grabbers' contains the list of all
//   sub-devices.
// - When using the 'gentl script' command, 'grabbers' contains the list of all
//   devices.

console.log("grabbers.length:", grabbers.length);

// Each item in 'grabbers' encapsulates all the ports related to one data
// stream:
//  TLPort          | GenTL producer
//  InterfacePort   | frame grabber or network interface card
//  DevicePort      | local device
//  StreamPort      | data stream
//  RemotePort      | camera (if available)

var PortNames = ['TLPort', 'InterfacePort', 'DevicePort', 'StreamPort',
                 'RemotePort'];

// Ports are objects which provide the following textual information:
//  name            | one of PortNames
//  tag             | port handle type and value (as shown in memento traces)

for (var i in grabbers) {
    var g = grabbers[i];
    console.log('- grabbers[' + i + ']');
    for (var pn of PortNames) {
        var port = g[pn];
        console.log('  - ' + port.name + ' (' + port.tag + ')');
    }
}

// Ports also have the following functions to work on GenICam features:
//  get(f)                | get value of feature f
//  set(f,v)              | set value v to feature f
//  execute(f)            | execute command f
//  done(f)               | test if command f is done (execution completed)
//  features([re])        | get list of features [matching~ re]
//  $features([re])       | strict* variant of features([re])
//  featuresOf(c, [re])   | get list of features of category c [matching~ re]
//  $featuresOf(c, [re])  | strict* variant of featuresOf(c, [re])
//  categories([re])      | get list of categories [matching~ re]
//  $categories([re])     | strict* variant of categories([re])
//  categoriesOf(c, [re]) | get list of categories of category c [matching~ re]
//  $categoriesOf(c, [re])| strict* variant of categoriesOf(c, [re])
//  ee(f,[re])            | get list of enum entries [matching~ re]
//                        | of enumeration f
//  $ee(f,[re])           | strict* variant of ee(f,[re])
//  has(f)                | test if f exists
//  has(f,v)              | test if f has an enum entry v
//  available(f)          | test if f is available
//  available(f,v)        | test if f has an enum entry v which is available
//  readable(f)           | test if f is readable
//  writeable(f)          | test if f is writeable
//  implemented(f)        | test if f is implemented
//  command(f)            | test if f is a command
//  selectors(f)          | get list of features that act as selectors of f
//  attributes(...)       | extract information from the XML file describing
//                        | the port
//  interfaces(f)         | get list of interfaces of f (e.g. ["IInteger"])
//  source(f)             | get the XML source of f
//  info(f,what)          | get XML information what of f
//  declare(t,f)          | declare a virtual user feature f of type t,
```

```
//                          | t can be one of "integer", "float", "string"
//  undeclare(f)            | undeclare (delete) a virtual user feature f
//  declared()              | get list of virtual user features
//
// * by strict we mean that the returned list contains only nodes/values
//   that are available (as dictated by 'pIsAvailable' GenICam node elements)
// ~ the returned list is filtered by regular expression matching

if (grabbers.length) {
    var port = grabbers[0].InterfacePort;
    console.log('Playing with', port.tag);
    // get(f)
    console.log('- InterfaceID: ' + port.get('InterfaceID'));
    // set(f,v)
    port.set('LineSelector', 'TTLIO11');
    // execute(f)
    port.execute('DeviceUpdateList');
    // features(re)
    console.log('- Features matching \'PCIe\':');
    for (var f of port.features('PCIe')) {
        console.log('  - ' + f);
    }
    // $ee(f)
    console.log('- Available enum entries for LineSource:');
    for (var ee of port.$ee('LineSource')) {
        console.log('  - ' + ee);
    }
    for (var ix of [0, 1, 2, 3, 9]) {
        var ee = 'Device' + ix + 'Strobe';
        // has(f, v)
        if (port.has('LineSource', ee)) {
            console.log('- ' + ee + ' exists');
        } else {
            console.log('- ' + ee + ' does not exist');
        }
        // available(f, v)
        if (port.available('LineSource', ee)) {
            console.log('- ' + ee + ' is available');
        } else {
            console.log('- ' + ee + ' is not available');
        }
    }
    // selectors(f)
    console.log('- LineSource feature is selected by',
                port.selectors('LineSource'));
    // attributes()
    console.log('- attributes()');
    var attrs = port.attributes();
    for (var n in attrs) {
        console.log('  - ' + n + ': ' + attrs[n]);
    }
    // attributes(f)
    console.log('- attributes(\'LineFormat\')');
    var attrs = port.attributes('LineFormat');
    for (var n in attrs) {
        console.log('  - ' + n + ': ' + attrs[n]);
    }
    // attributes(f)
    var fmt = port.get('LineFormat');
    console.log('- attributes(\'LineFormat\', \'' + fmt + '\')');
    var attrs = port.attributes('LineFormat', fmt);
    for (var n in attrs) {
        console.log('  - ' + n + ': ' + attrs[n]);
    }
    // optional suffixes to integer or float feature names
```

```
    if (port.available('DividerToolSelector') &&
        port.available('DividerToolSelector', 'DIV1')) {
        var feature = 'DividerToolDivisionFactor[DIV1]';
        var suffixes = ['.Min', '.Max', '.Inc', '.Value'];
        console.log('- Accessing ' + suffixes + ' of ' + feature);
        for (var suffix of suffixes) {
            console.log( '  - ' + suffix + ': ' + port.get(feature + suffix));
        }
    }
    var remotePort = grabbers[0].RemotePort;
    if (remotePort) {
        // getReg(f, len)
        // setReg(f,data)
        if (remotePort.available('LUTValueAll')) {
            // get register length
            var len = remotePort.get('LUTValueAll.Length');
            // get first 8 bytes
            if (len >= 8) {
                var data = remotePort.getReg('LUTValueAll', 8);
                for (var ix in data) {
                    console.log(' - LUTValueAll[' + ix + ']: ' + data.charCodeAt(ix));
                }
                // write data
                // remotePort.setReg('LUTValueAll', data);
            }
        }
    }
}

// Camera ports (RemotePort) also have the following functions:
//  brRead(addr)    | read bootstrap register (32-bit big endian)
//  brWrite(addr,v) | write value to bootstrap register (32-bit big endian)

if (grabbers.length) {
    var port = grabbers[0].RemotePort;
    if (port) {
        console.log('Playing with', port.tag);
        var brStandard = 0x00000000;
        var standard = port.brRead(brStandard);
        console.log('Bootstrap register "Standard" is ' + standard);
    }
}
```

doc/module1.js

```
// This file describes the special 'module' variable of Euresys GenApi Script.
// It can be executed by running 'gentl script egrabber://doc/module1.js' or by
// entering 'g.runScript egrabber://doc/module1.js' in eGrabber Studio's
// Terminal. It is also dynamically loaded by the egrabber://doc/builtins.js
// script.

// 'module' is a special per-module variable. It cannot be declared with var.
// It always exists, and contains a few items:
console.log('Started execution of "' + module.filename + '"');
console.log('This script is located in directory "' + module.curdir + '"');

// Modules can export values via module.exports (which is initialized as an
// empty object):
module.exports = { description: 'Example of Euresys GenApi Script module'
                 , plus2: function(x) {
                     return x + 2;
                   }
                 , hello: function() {
```

```
                    console.log('Hello from ' + module.filename);
                  }
                };

console.log('module.exports contains: ');
for (var e in module.exports) {
    console.log('- ' + e + ' (' + typeof module.exports[e] + ')');
}

console.log('Completed execution of ' + module.filename);
```

7. Euresys GenApi Extensions

The Euresys GenApi module provides useful extensions exposed as virtual GenApi features accessible with one of the following syntaxes.

@-commands

Virtual features prefixed by `@`, e.g., `@features FileAccessControl` to list the features in the `FileAccessControl` category.

properties

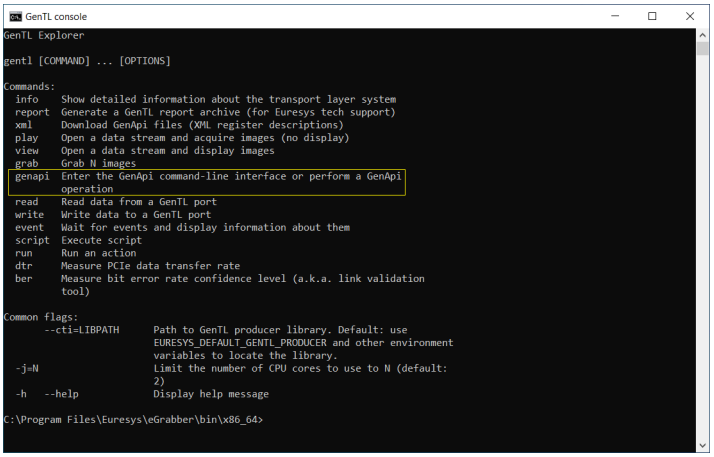
Virtual features suffixed by `.<property>`, e.g. `Width.Min` and `Width.Max` to query the minimum and maximum values of the `Width` feature.

These virtual GenApi features are only available in modules opened with the Euresys GenApi implementation. They extend the set of features exposed by the camera description file (a.k.a. the “XML” file).

Those extensions are useful in applications to build advanced queries on remote device (or GenTL module) features or to configure the behavior of the Euresys GenApi implementation. The following sections document the available commands and properties.

Learn and experiment

The command-line `gentl` tool provides an environment to experiment with and learn about the Euresys `@`-commands. This environment is available with the tool mode `genapi`.



You can quickly enter the mode by executing `gentl genapi` at the command prompt; this will open the remote device associated to the first GenTL device of the first GenTL interface of the default GenTL producer.

```
C:\Program Files\Euresys\eGrabber\bin\x86_64>gentl.exe genapi
Using interface "PC1632 - Coaxlink Quad (1-camera) - KQU00031".
Using device "Device0" (device access flags: DEVICE_ACCESS_CONTROL).
Opening remote device port... Done.
Retrieving register description for remote device (Device0)... Done.
Registering EVENT_REMOTE_DEVICE event... Done.
Writing False to @hexadecimal... Done.
remote>
```

The environment is now waiting for user commands to execute. You can obtain some help by entering the command `help`:

```
remote> help
GenApi commands:
  dump          dump exposed features from Root
  dump C        dump exposed features of a category C
  dumplimit N   set selector limit for dumps (default: 0), if N=0 selector
                values are not changed otherwise at most N selector values are
                used
  get F         get value of feature F
  set F V       set feature F to V
  execute F     execute command F
  done F        check if processing of command F is done
  attributes    list camera file attributes
  categories    list categories
  features C    list features of category C
  source F      show GenApi source about feature F
  ? F WHAT      query information WHAT about feature F (e.g., ? Gain
                tooltip)
General commands:
  quiet         set verbosity level to quiet
  normal        set verbosity level to normal
  loud          set verbosity level to loud
  help          display this help message
  exit          exit the CLI
remote>
```

Let's focus on the command `get`, which gets the value of a given feature. For example, to query the remote device feature `Width`, we can issue the command `get Width`:

```
remote> get Width
Reading Width... Done.
2592
```

Please note this environment offers other useful commands (`set`, `dump`, `source`…); we encourage you to take some time to discover and experiment them as they are useful when troubleshooting camera setup or when prototyping some queries/commands that will ultimately be coded in an eGrabber-based application.

The `gentl genapi` command prompt can auto-complete or show completion options of feature names when pressing the Tab key. For example, it's possible to type `get Wid` and press the Tab key… either there is no other feature starting with `Wid` and the full feature name `Width` is completed, or the available options are printed to the console. (The completion only applies to GenApi features, so the names of GenApi nodes which don't belong to any category referenced from the `Root` category are not auto-completed.)

Euresys GenApi @-commands

The Euresys GenApi environment provides a description of the available `@`-commands with the virtual feature `@help`. As most `@`-commands, this virtual feature implements the GenApi `IString` interface therefore it acts a GenApi string feature and its value can be queried from the `gentl genapi` command prompt by running the command `get @help`:

```
remote> get @help
Special @-commands:
  @xml                  current GenApi XML file
  @attributes           list attributes of GenApi document
  @features [C]         list features [of category C]
  @!features [C]        list available features [of category C]
  @categories [C]       list categories [of category C]
  @!categories [C]      list available categories [of category C]
  @ee E                 list enum entries of enumeration E
  @!ee E                list available enum entries of enumeration E
  @interfaces F         list interfaces of F
  @selected F           list nodes selected by F
  @selectors F          list selectors of F
```

```
  @eventnodes ID            list nodes with specified eventID
  @available F [EE]         'True' if F [or enum entry EE of F] is available, 'False' otherwise
  @readable F               'True' if F is readable, 'False' otherwise
  @writeable F              'True' if F is writeable, 'False' otherwise
  @implemented F            'True' if F is implemented, 'False' otherwise
  @command F                'True' if F is a command, 'False' otherwise
  @done F                   'True' if the command F is done, 'False' otherwise
  @source F                 the XML source of F
  @info F ELEM              the XML element ELEM of F
  @eeinfo F E ELEM          the XML element ELEM of entry E of F
  @config                   current config values
  @declare(integer) NAME    declare a virtual user feature of type Integer
  @declare(float) NAME      declare a virtual user feature of type Float
  @declare(string) NAME     declare a virtual user feature of type String
  @undeclare NAME           undeclare (delete) a virtual user feature
  @declared                 list virtual user features
  @poll                     invalidate and report features with expired polling time
  @pollable                 list features with a defined polling time
  @show F...                show features
  @help [TOPIC]             this general help text or a TOPIC specific help

Additional help topics: properties, selectors, bootstrap, fileaccess

Filtering results of @features and @ee commands:
  =~ RE      only keep results matching (PERL) regular expression RE
  =? GLOB    only keep results matching glob pattern GLOB
Glob patterns only match whole-words. When using regular expression patterns,
use the caret (^) and dollar ($) symbols to match whole-words.

Examples:
  @features
  @features =? Pay[Ll]oadSize
  @features =~ ^(CxpLinkConfiguration|LinkConfig)$
  @ee PixelFormat
  @ee TriggerSource =? CXP*
  @ee PixelFormat =~ Mono.*
```

Below are a few concrete examples you can try in the `gentl genapi` command prompt or in eGrabber Studio's Terminal.

Checking availability

```
remote> get @available PixelFormat
True
remote> get @available PixelFormat Mono8
True
```

The first command checks whether the feature `PixelFormat` is available (i.e. whether it's readable and/or writable) while the second command checks whether the entry `Mono8` of the enumeration `PixelFormat` is available.

Getting XML code

The complete XML document associated with the current GenApi context can be obtained by getting the virtual feature `@xml` and the XML snippet of a specific GenApi node or feature can be obtained with the virtual feature `@source`:

```
remote> get @source DeviceModelName
<StringReg Name="DeviceModelName" NameSpace="Standard">
<ToolTip>Device Model Name</ToolTip>
```

```
<Description>Device Model Name.</Description>
<DisplayName>Device Model Name</DisplayName>
<Address>0x2020</Address>
<Length>32</Length>
<AccessMode>RO</AccessMode>
<pPort>Device</pPort>
</StringReg>
```

GenApi module configuration

The GenApi module configuration can be displayed with the command `get @config`:

```
remote> get @config
@hexadecimal = False (bool)
@rollbackSelectors = True (bool)
@portCache = True (bool)
@checkRange = True (bool)
@checkRangeOnRead = False (bool)
@useRepresentation = True (bool)
@setTLParamsLocked = True (bool)
@checkFloatToIntOverflow = True (bool)
@checkError = True (bool)
```

Each configuration option can be individually read or modified.

Configuration name	Configuration description	Default value
`hexadecimal`	default integer representation (1)(5)	`False`
`rollbackSelectors`	set selector values back to original value (2)	`True`
`portCache`	enable/disable the cache of registers (3)	`True`
`checkRange`	enable/disable number range checking when writing (4)	`True`
`checkRangeOnRead`	enable/disable number range checking when reading (4)	`False`
`useRepresentation`	enable/disable use of node representation (5)	`True`
`setTLParamsLocked`	set `TLParamsLocked` automatically or not (6)	`True`
`checkFloatToIntOverflow`	enable/disable overflow checking (7)	`True`
`checkError`	enable/disable `<pError>` element handling (8)	`True`

Notes:

- **(1)** This flag is ignored if `@useRepresentation` is `True`, cf. (5).

- (2) When using the Euresys selector syntax, the selector values are restored by default; this behavior can be disabled by setting `@rollbackSelectors` to `False`.
  - The selector syntax is a compact syntax that takes care of setting/restoring selector values before/after reading or writing a feature, in an atomic, transaction-like access. For example on a Coaxlink Card, to check the `LineStatus` of the first isolated input of the first I/O set (`IIN11`) on the interface module, we need to first set the `LineSelector` feature to the physical I/O line we want to query (i.e. `IIN11`) before reading the `LineStatus` feature:if> set LineSelector IIN11 if> get LineStatus FalseWith the Euresys selector syntax, we can combine those operation in one query:if> get LineStatus[IIN11] FalseThis command will determine the selector feature associated to `LineStatus` i.e. `LineSelector`, backup the current selector value, set the new selector value `IIN11`, get the value of the feature `LineStatus` and finally restore the selector to its original value. Those operations are executed as one transaction: no concurrent thread can change the `LineSelector` in between.
- (3) The GenApi register mechanism can be completely disabled when setting `@portCache` to `Flase`; please note that disabling the cache may result in very poor performance of the GenApi feature access.
- (4) GenApi integer or float nodes have minimal and maximal values that determine a range and may also have an increment value. Those elements impose constraints on the values of a feature:
  - when @checkRange is True, before writing a value to a feature, the value must satisfy the constraints; otherwise, the “set” operation fails;
  - when @checkRangeOnRead is True, the current value of feature is validated before being returned to the user; if the constraints are not satisifed, the “get” operation fails.
- (5) GenApi integer or float nodes may have a `<Representation>` element that gives a hint about how to display the value:
  - when @useRepresentation is True, the node representation is used (or its default representation when omitted in the XML);
  - when @useRepresentation is False, the node representation is ignored and integer values are displayed in hexadecimal format if @hexadecimal is True, in decimal format otherwise.
- (6) When the standard feature `TLParamsLocked` is available on the remote device, it is automatically set to 1 prior to executing the command `AcquisitionStart` and it is automatically reset to `0` after executing the command `AcquisitionStop`; this behavior can be disabled by setting `@setTLParamsLocked` to `False`.
- (7) When the GenApi runtime needs to convert the value of a float node into an integer value (signed 64-bit integer), there are two possibilities: either the rounded value can fit in the 64-bit integer range, or it cannot. The latter is known as overflow and `@checkFloatToIntOverflow` controls the behavior of the GenApi runtime in this case:
  - when @checkFloatToIntOverflow is True, a conversion overflow error is raised;
  - when @checkFloatToIntOverflow is False, converted values are “saturated” i.e. they are limited to the 64-bit integer range.

- **(8)** GenApi nodes can have a `<pError>` element referencing an enumeration node; the enumeration node must have one entry with the integer value `0` (indicating no error) and other entries defining specific errors; after setting the value of a node, the GenApi runtime checks the enumeration integer value and raises an error message based on the corresponding entry `<DisplayName>` (or `ToolTip`) element if it's not `0`; this behavior can be disabled by setting `@checkError` to `False`.

Virtual features

In addition to the `@`-commands listed earlier, the GenApi module handles two additional kinds of virtual features: *user features* and *built-in features*. They are further decribed in the following sections.

Virtual user feature

The Euresys GenApi module supports the creation of virtual user features that act as GenApi floating nodes (i.e. not bound to any register); those virtual user features can be used in " Euresys GenApi scripts" on page 29 to maintain state or share data between script runs performed on the **same** GenApi contexts:

- To create a virtual user feature: @declare(integer), @declare(float), @declare(string)
- To remove a virtual user feature: @undeclare
- To access (set/get) a virtual user feature, prefix the user feature name with a $ and use it as a regular floating node

Examples:

```
remote> set @declare(integer) MyInteger
remote> get $MyInteger
0
remote> set $MyInteger 10
remote> get $MyInteger
10
remote> set @undeclare MyInteger
remote> get $MyInteger
not found ($MyInteger)
```

Built-in Virtual features

The built-in features are helper GenApi features that can be used in applications or in " Euresys GenApi scripts" on page 29.

Bootstrap register helpers

On a remote device module, the following virtual features are present:

- @braddr: an integer node representing a bootstrap register address, acting as a selector for @brname and @brrepr;
- @brname: a read-only string node (selected by @braddr) giving the name of a bootstrap register

- @brrepr: a read-only string node (selected by @braddr) giving a representation of the current value of a bootstrap register; the representation tries to decode bitfields when applicable

For example, with a CoaXPress camera:

```
remote> get @brname[0x0000]
Standard
remote> get @brrepr[0x0000]
0xc0a79ae5 <CoaXPress>
remote> get @brname[0x2000]
DeviceVendorName
remote> get @brname[0x4014]
ConnectionConfig
remote> get @brrepr[0x4014]
0x00040048 <4 connections, 6.250 Gbps (CXP-6)>
remote>
```

Note: depending on the remote device transport layer type, the helpers handle CoaXPress or GigE Vision bootstrap registers.

File access control helpers

The SFNC defines a category `FileAccessControl` containing a set of features providing the building blocks to access files in a device. Those features provide a low-level API that is not meant to be used directly, please refer to the “File Access Control” section of the SFNC specification for details.

The Euresys GenApi runtime uses the `FileAccessControl` features to build a higher-level API as virtual features; the standard feature `FileSelector` (of the category `FileAccessControl`) is used to select the file to use.

- @filectlsel: an enumertation node, acting as a selector for @filectl; possible values are OpenRead, OpenWrite, OpenReadWrite, Close, and Delete
- @filectl: a command node for executing or checking the execution status of the action selected by @filectlsel on a file selected by FileSelector
  - @filectl[OpenRead] opens the file selected by FileSelector in Read mode
  - @filectl[OpenWrite] opens the file selected by FileSelector in Write mode
  - @filectl[OpenReadWrite] opens the file selected by FileSelector in ReadWrite mode
  - @filectl[Close] closes the file selected by FileSelector
  - @filectl[Delete] deletes the file selected by FileSelector
- @fileoffset: an integer node representing an offset in the register node @file, acting as a selector for @file
- @file: a register node exposing the whole file selected by FileSelector as a contiguous register for reading/writing; the @fileoffset selector can be used to access a region of the contents starting at the given offset (e.g. @file[0x100]) (1)
  - @file exposes the whole file (unless @fileoffset is not 0);
  - @file[0x100] exposes the file contents starting at offset 0x100 until the end of the file (2)

Notes:

- **(1)** Accessing the `@file` register is easy thanks to the GenApi register API. For example, using an eGrabber object g in C++:// read "File1" contentsg.setString<Euresys::RemoteModule> ("FileSelector", "File1");g.execute<Euresys::RemoteModule>("@filectl[OpenRead]");int64_t file1Size = g.getInteger<Euresys::RemoteModule>("@file[0].Length");std::vector<uint8_t> file1 (static_cast<size_t>(file1Size));g.getRegister<Euresys::RemoteModule>("@file[0]", &file1[0], file1.size());// "File1" contents is now in vector file1
- **(2)** The length of the `@file` register depends on the `@fileoffset` value. `@file[0x100].Length` equals `@file[0].Length - 0x100`.

Euresys feature properties

Depending on the GenApi interfaces of a node (e.g., `Integer`, `Float`, `String`, `Register`), different properties can be queried. For example, it's possible to query the minimum and maximum values of the `Width` feature as follows:

```
remote> get Width.Min
32
remote> get Width.Max
2592
```

The table below shows the available properties.

Property name	Property description
`Min`	minimum value of the number or the enumeration
`Max`	maximum value of the number or the enumeration
`Inc`	increment of the number or the enumeration (not always available)
`Done`	command completion status - is the command done (idle) or is it still running
`Address`	register address
`Length`	register length in bytes
`ValueDisplayName`	enumeration entry display name (or the actual entry name if not defined)
`ValueToolTip`	enumeration entry tooltip text (or an empty string if not defined)
`Representation`	representation of the number (hint about how to display the value)
`DisplayNotation`	notation of the float number (how to display the floating point value)
`DisplayPrecision`	precision of the float number (total number of digits)
`Features`	features of the category
`Unit`	unit of the number (if defined)
`Entry.<name>`	numerical value of the enumeration entry with the given `<name>`

Examples:

- Width.Min: the minimum value of Width
- PixelFormat.Entry.Mono8: the numerical value of the PixelFormat entry Mono8

8. eGrabber for MultiCam users

Concepts

MultiCam	eGrabber
Board	Interface
Channel	Device + Data stream
Surface	Buffer
Surface cluster (`MC_Cluster`)	Buffers announced to the data stream
-	Remote device (camera)
MultiCam parameters	GenApi " GenApi" on page 6
-	GenApi " GenApi" on page 6
CAM file	Euresys GenApi script
-	CallbackOnDemand
Callback functions	CallbackSingleThread
-	CallbackMultiThread

Initialization

```
MCSTATUS status = McOpenDriver(NULL);
if (status != MC_OK) {
    ...
}
```

```
Euresys::EGenTL gentl;
```

Channel creation

```
MCSTATUS status;
MCHANDLE channel;

status = McCreate(MC_CHANNEL, &handle);
if (status != MC_OK) {
    ...
}
status = McSetParamInt(channel, MC_DriverIndex, CARD_INDEX);
if (status != MC_OK) {
    ...
}
status = McSetParamInt(channel, MC_Connector, CONNECTOR);
if (status != MC_OK) {
    ...
}
```

```
Euresys::EGrabber<> grabber(gentl, CARD_INDEX, DEVICE_INDEX);
```

Surface creation (automatic)

```
status = McSetParamInt(channel, MC_SurfaceCount, BUFFER_COUNT);
if (status != MC_OK) {
    ...
}
```

```
grabber.reallocBuffers(BUFFER_COUNT);
```

Surface creation (manual)

```
for (size_t i = 0; i < BUFFER_COUNT; ++i) {
    MCHANDLE surface;
    MCSTATUS status;
    void *mem = malloc(BUFFER_SIZE);
    if (!mem) {
        ...
    }
    status = McCreate(MC_DEFAULT_SURFACE_HANDLE, &surface);
    if (status != MC_OK) {
        ...
    }
    status = McSetParamInt(surface, MC_SurfaceSize, BUFFER_SIZE);
    if (status != MC_OK) {
        ...
    }
    status = McSetParamPtr(surface, MC_SurfaceAddr, mem);
    if (status != MC_OK) {
        ...
    }
    status = McSetParamPtr(surface, MC_SurfaceContext, USER_PTR[i]);
    if (status != MC_OK) {
        ...
    }
    status = McSetParamInst(channel, MC_Cluster + i, surface);
    if (status != MC_OK) {
        ...
    }
}
```

```
for (size_t i = 0; i < BUFFER_COUNT; ++i) {
    void *mem = malloc(BUFFER_SIZE);
    if (!mem) {
        ...
    }
    grabber.announceAndQueue(Euresys::UserMemory(mem, BUFFER_SIZE, USER_PTR[i]));
}
```

Surface cluster reset

```
MCSTATUS status;
for (size_t i = 0; i < BUFFER_COUNT; ++i) {
    MCHANDLE surface;
    status = McGetParamInst(channel, MC_Cluster + i, &surface);
    if (status != MC_OK) {
        ...
    }
    status = McSetParamInt(surface, MC_SurfaceState, MC_SurfaceState_FREE);
    if (status != MC_OK) {
```

```
        ...
    }
}
status = McSetParamInt(channel, MC_SurfaceIndex, 0);
if (status != MC_OK) {
    ...
}
```

```
grabber.resetBufferQueue();
```

Frame grabber configuration

MultiCam	eGrabber
McSetParamStr(H, MC_CamFile, filepath)	grabber.runScript(filepath)
-	grabber.runScript(script)
McSetParamInt(H, id, value) or McSetParamNmInt(H, name, value)	grabber.setInteger<M>(name, value)
McSetParamFloat(H, id, value) or McSetParamNmFloat(H, name, value)	grabber.setFloat<M>(name, value)
McSetParamStr(H, id, value) or McSetParamNmStr(H, name, value)	grabber.setString<M>(name, value)

where H is a MC_HANDLE (the global MC_CONFIGURATION handle, a board handle, or a channel handle), and M specifies the target module (either SystemModule, InterfaceModule, DeviceModule, or StreamModule).

Camera configuration

MultiCam	eGrabber
-	grabber.runScript(filepath)
-	grabber.runScript(script)
-	grabber.setInteger<RemoteModule>(name, value), grabber.setFloat<RemoteModule>(name, value), or grabber.setString<RemoteModule>(name, value)

Script files

```
; CAM file
ChannelParam1 = Value1;
ChannelParam2 = Value2;
```

```
// Euresys GenApi Script
var grabber = grabbers[0];
grabber.DevicePort.set('DeviceFeature1', Value1);
grabber.DevicePort.set('DeviceFeature2', Value2);
grabber.RemotePort.set('CameraFeatureA', ValueA);
```

Acquisition start/stop

```
// start "live"
McSetParamInt(channel, MC_GrabCount, MC_INFINITE);
McSetParamInt(channel, MC_ChannelState, MC_ChannelState_ACTIVE);
// stop
McSetParamInt(channel, MC_ChannelState, MC_ChannelState_IDLE);
// grab 10 images
McSetParamInt(channel, MC_GrabCount, 10);
McSetParamInt(channel, MC_ChannelState, MC_ChannelState_ACTIVE);
```

```
// start "live"
grabber.start();
// stop
grabber.stop();
// grab 10 images
grabber.start(10);
```

Synchronous (blocking) buffer reception

```
MCSTATUS status;
MCSIGNALINFO info;
// wait for a surface
status = McWaitSignal(channel, MC_SIG_SURFACE_PROCESSING, timeout, &info);
if (status != MC_OK) {
    ...
}
MCHANDLE surface = info.SignalInfo;
// process surface
...
// make surface available for new images
status = McSetParamInt(surface, MC_SurfaceState, MC_SurfaceState_FREE);
if (status != MC_OK) {
    ...
}
```

```
// wait for a buffer
Buffer buffer = grabber.pop(timeout);
// process buffer
...
// make buffer available for new images
buffer.push(grabber);
```

```
{
    // wait for a buffer
    ScopedBuffer buffer(grabber, timeout);
    // process buffer
    ...
    // ScopedBuffer destructor takes care of making buffer available for new images
}
```

Callbacks

```
class MyChannel {
    public:
        MyChannel() {
            // create and configure channel
            ...
```

```
            // enable "SURFACE_PROCESSING" events
            status = McSetParamInt(channel, MC_SignalEnable + MC_SIG_SURFACE_PROCESSING,
                                   MC_SignalEnable_ON);
            if (status != MC_OK) {
                ...
            }
            // enable "END_EXPOSURE" events
            status = McSetParamInt(channel, MC_SignalEnable + MC_SIG_END_EXPOSURE,
                                   MC_SignalEnable_ON);
            if (status != MC_OK) {
                ...
            }
            // register "extern C" callback function
            MCSTATUS status = McRegisterCallback(channel, GlobalCallbackFunction, this);
            if (status != MC_OK) {
                ...
            }
        }

        void onEvent(MCSIGNALINFO *info) {
            switch (info->Signal) {
                case MC_SIG_SURFACE_PROCESSING:
                    MCHANDLE surface = info.SignalInfo;
                    // process surface
                    ...
                    break;
                case MC_SIG_END_EXPOSURE:
                    // handle "END_EXPOSURE" event
                    ...
                    break;
            }
        }

    private:
        MCHANDLE channel;
};

void MCAPI GlobalCallbackFunction(MCSIGNALINFO *info) {
    if (info && info->Context) {
        MyGrabber *grabber = (MyGrabber *)info->Context;
        grabber->onEvent(info);
    }
};
```

```
class MyGrabber : public EGrabber<CallbackSingleThread> {
    public:
        MyGrabber(EGenTL &gentl) : EGrabber<CallbackSingleThread>(gentl) {
            // configure grabber
            ...
            // enable "NewBuffer" events
            enableEvent<NewBufferData>();
            // enable "Cic" events
            enableEvent<CicData>();
        }

    private:
        virtual void onNewBufferEvent(const NewBufferData& data) {
            ScopedBuffer buffer(*this, data);
            // process buffer
            ...
        }
        virtual void onCicEvent(const CicData &data) {
            // handle "Cic" event
            ...
        }
```

```
};
```

Synchronous (blocking) event handling

```
class MyChannel {
    public:
        MyChannel() {
            // create and configure channel
            ...
            // enable "END_EXPOSURE" events
            status = McSetParamInt(channel, MC_SignalEnable + MC_SIG_END_EXPOSURE,
                                           MC_SignalEnable_ON);
            if (status != MC_OK) {
                ...
            }
        }

        void waitForEvent(uint32_t timeout) {
            // wait for an event
            MCSTATUS status = McWaitSignal(channel, MC_SIG_END_EXPOSURE, timeout, &info);
            if (status != MC_OK) {
                ...
            }
            // handle "END_EXPOSURE" event
            ...
        }

    private:
        ...
};
```

```
class MyGrabber : public EGrabber<CallbackOnDemand> {
    public:
        MyGrabber(EGenTL &gentl) : EGrabber<CallbackOnDemand>(gentl) {
            // configure grabber
            ...
            // enable "Cic" events
            enableEvent<CicData>();
        }

        void waitForEvent(uint64_t timeout) {
            // wait for an event
            processEvent<CicData>(timeout);
        }

    private:
        // onCicEvent is called by processEvent when a "Cic" event occurs
        virtual void onCicEvent(const CicData &data) {
            // handle "Cic" event
            ...
        }
};
```

9. .NET assembly

eGrabber can be used in .NET languages (C#, VB.NET, etc.) via two .NET assemblies named `EGrabber.NETFramework.dll` and `EGrabber.NET.dll`.

- EGrabber.NETFramework.dll assembly only works on Windows and requires .NETFramework.
- EGrabber.NET.dll assembly is cross-platform and works with .NET (previously called .NET Core).

The assembly used at runtime must be the same as the assembly used at compile-time.

A first example

This example creates a grabber and displays basic information about the interface, device, and remote device modules it contains. This is the C# version of " eGrabber" on page 12:

```
using System;
using Euresys.EGrabber;

namespace FirstExample {
    class ExampleShowInfo {
        const int CARD_IX = 0;
        const int DEVICE_IX = 0;

        static void ShowInfo() {
            using (var gentl = new EGenTL()) {                                          // 1
                using (var grabber = new EGrabber(gentl, CARD_IX, DEVICE_IX)) {         // 2
                    string card = grabber.Interface.Get<string>("InterfaceID");         // 3
                    string dev = grabber.Device.Get<string>("DeviceID");                // 4
                    ulong width = grabber.Remote.Get<ulong>("Width");                   // 5
                    ulong height = grabber.Remote.Get<ulong>("Height");                 // 5

                    System.Console.WriteLine("Interface:    {0}", card);
                    System.Console.WriteLine("Device:       {0}", dev);
                    System.Console.WriteLine("Resolution:   {0}x{1}", width, height);
                }
            }
        }

        static void Main() {
            try {                                                                       // 6
                ShowInfo();
            } catch (System.Exception e) {                                              // 6
                System.Console.WriteLine("error: {0}", e.Message);
            }
        }
    }
}
```

1. Create an `EGenTL` object. This will locate, open, and initialize the GenTL producer (e.g., `coaxlink.cti`).
2. Create an `EGrabber` object. The constructor needs the `gentl` object we've just created. It also takes as optional arguments the indices of the interface and device to use.

3. Use " GenApi" on page 6 to query the ID of the Coaxlink card. We want an answer from the " Interface module" on page 8, so the Get is done on grabber.Interface. Notice the <string> to indicate that we want to query the GenICam IString interface of the feature and we want to get back a string value.
4. Similarly, query the ID of the device. This time, we use grabber.Device to target the " Device module" on page 9.
5. Finally, read the camera resolution. This time, we use grabber.Remote since the values must be retrieved from the camera. Notice the <ulong> to indicate that we want to query the GenICam IInteger interface of the feature and we want to get back a ulong value.
6. eGrabber uses exceptions to report errors, so we wrap our code inside a try ... catch block.

Example of program output:

```
Interface:    PC1633 - Coaxlink Quad G3 (2-camera) - KQG00014
Device:       Device0
Resolution:   4096x4096
```

Differences between C++ and .NET eGrabber

eGrabber classes

C++	.NET
EGrabber<>	EGrabber
EGrabber<CallbackOnDemand>	EGrabber
EGrabber<CallbackSingleThread>	-
EGrabber<CallbackMultiThread>	-
Buffer	Buffer
ScopedBuffer	ScopedBuffer used in a using-statement

EGrabber methods

- In the C++ API, method names are written in camelCase. In C#, method names are written in CamelCase. For example, the C++ reallocBuffers becomes ReallocBuffers in C#.

- Other differences:C++.NETgetInfo<MODULE,TYPE>(cmd)Module.GetInfo<TYPE>(cmd)getInteger<MODULE>(f)Module.Get<long>(f), Module.Get<int>(f), ⋯-Module.Get<bool>(f)getFloat<MODULE>(f)Module.Get<double>(f), Module.Get<float>(f)getString<MODULE>(f)Module.Get<string>(f)getStringList<MODULE>(f)Module.Get<string[]>(f)setInteger<MODULE>(f, v)Module.Set<long>(f, v), Module.Set<int>(f, v), ⋯-Module.Set<bool>(f, v)setFloat<MODULE>(f, v)Module.Set<double>(f, v), Module.Set<float>(f, v)setString<MODULE>(f, v)Module.Set<string>(f, v)execute<MODULE>(f)Module.Execute(f)enableEvent<EVENT_DATA>()EnableEvent(EventType.EVENT_DATA)disableEvent<EVENT_DATA>()DisableEvent(EventType.EVENT_DATA)where
  - MODULE is SystemModule, InterfaceModule, DeviceModule, StreamModule, or RemoteModule;
  - Module is System, Interface, Device, Stream, or Remote;
  - EVENT_DATA is NewBufferData, DataStreamData, IoToolboxData, CicData, CxpInterfaceData, DeviceErrorData, CxpDeviceData, or RemoteDeviceData.

Callbacks

In .NET, callbacks are defined as methods with two arguments (the grabber and the actual event data), and are registered with `RegisterEventCallback`:

```
grabber.RegisterEventCallback<NewBufferData>(OnNewBufferData);
grabber.RegisterEventCallback<DataStreamData>(OnDataStreamData);
grabber.RegisterEventCallback<IoToolboxData>(OnIoToolboxData);
grabber.RegisterEventCallback<CicData>(OnCicData);
grabber.RegisterEventCallback<CxpInterfaceData>(OnCxpInterfaceData);
grabber.RegisterEventCallback<DeviceErrorData>(OnDeviceErrorData);
grabber.RegisterEventCallback<CxpDeviceData>(OnCxpDeviceData);
grabber.RegisterEventCallback<RemoteDeviceData>(OnRemoteDeviceData);
```

A complete example is given in the next section.

Processing events using callbacks

This program displays basic information about CIC events generated by a grabber:

```
using System;
using Euresys.EGrabber;

namespace Callbacks {
    class CallbackExample {
        static void ShowEvents(EGrabber grabber) {
            grabber.RunScript("config.js");                                  // 1
            grabber.RegisterEventCallback<CicData>(OnCicData);               // 2
            grabber.EnableEvent(EventType.CicData);                          // 5

            grabber.ReallocBuffers(3);                                       // 6
            grabber.Start();                                                 // 6
            while (true) {                                                   // 6
                grabber.ProcessEvent(EventType.Any);                         // 6
            }
        }

        static void OnCicData(EGrabber grabber, CicData data) {              // 3
            System.Console.WriteLine("Timestamp: {0} us, {1}",               // 4
                data.Timestamp, data.NumId);
        }
```

```
        static void Main() {
            try {
                using (var gentl = new EGenTL()) {
                    using (var grabber = new EGrabber(gentl)) {
                        ShowEvents(grabber);
                    }
                }
            } catch (System.Exception e) {
                System.Console.WriteLine("error: {0}", e.Message);
            }
        }
    }
}
```

1. Run a `config.js` script which should:
   - properly configure the camera and frame grabber;
   - enable notifications for CIC events.
2. Register the callback function for handling the *CIC* events.
3. The *CIC* event callback has two arguments: the grabber that fired the event, and the actual event data.
4. In the body of the callback function, simply display basic information about the event.
5. Enable `CicData` events on the grabber.
6. Start the grabber and enter an infinite loop to process any incoming event. *CIC* events will be notified in the same thread.

Example of program output:

```
timestamp: 2790824897 us, EVENT_DATA_NUMID_CIC_CAMERA_TRIGGER_RISING_EDGE
timestamp: 2790824897 us, EVENT_DATA_NUMID_CIC_STROBE_RISING_EDGE
timestamp: 2790824902 us, EVENT_DATA_NUMID_CIC_CXP_TRIGGER_ACK
timestamp: 2790825897 us, EVENT_DATA_NUMID_CIC_STROBE_FALLING_EDGE
timestamp: 2790830397 us, EVENT_DATA_NUMID_CIC_CAMERA_TRIGGER_FALLING_EDGE
timestamp: 2790830401 us, EVENT_DATA_NUMID_CIC_CXP_TRIGGER_ACK
timestamp: 2790842190 us, EVENT_DATA_NUMID_CIC_ALLOW_NEXT_CYCLE
timestamp: 2790842190 us, EVENT_DATA_NUMID_CIC_CAMERA_TRIGGER_RISING_EDGE
timestamp: 2790842191 us, EVENT_DATA_NUMID_CIC_STROBE_RISING_EDGE
timestamp: 2790842195 us, EVENT_DATA_NUMID_CIC_CXP_TRIGGER_ACK
```

10. Python

eGrabber can also be used in Python.

Installation

The Python bindings for eGrabber are provided in a Pyhon wheel installation package (a `.whl` file) located in the `python` subdirectory of the eGrabber installation directory[1] . Depending on your Python setup, installation can be as easy as:

```
python -m pip install <PATH_TO_EGRABBER_WHL>
```

A first example

This example creates a grabber and displays basic information about the interface, device, and remote device modules it contains. This is the Python version of " eGrabber" on page 12:

```
from egrabber import *                                  # 1

card_ix = 0
device_ix = 0

def showInfo():
    gentl = EGenTL()                                    # 2
    grabber = EGrabber(gentl, card_ix, device_ix)       # 3

    card = grabber.interface.get('InterfaceID')         # 4
    dev = grabber.device.get('DeviceID')                # 5
    width = grabber.remote.get('Width')                 # 6
    height = grabber.remote.get('Height')               # 6

    print('Interface:    %s' % card)
    print('Device:       %s' % dev)
    print('Resolution:   %ix%i' % (width, height))

try:                                                    # 7
    showInfo()
except Exception as e:                                  # 7
    print('error: %s' % e)
```

1. Import the `egrabber` module.
2. Create an `EGenTL` object. This will locate, open, and initialize the GenTL producer (e.g., `coaxlink.cti`).
3. Create an `EGrabber` object. The constructor needs the `gentl` object created in step 2. It also takes as optional arguments the indices of the interface and device to use.

[1] On Windows, eGrabber is typically installed in `C:\Program Files\Euresys\eGrabber`.

On Linux, eGrabber is installed in `/opt/euresys/egrabber`.

On macOS, eGrabber is installed in `/usr/local/opt/euresys/egrabber`.

4. Use " GenApi" on page 6 to query the ID of the Coaxlink card. We want an answer from the " Interface module" on page 8, so the get is done on `grabber.interface`.
5. Similarly, query the ID of the device. This time, we use `grabber.device` to target the " Device module" on page 9.
6. Finally, read the camera resolution. This time, we use `grabber.remote` since the values must be retrieved from the camera.
7. egrabber uses exceptions to report errors, so we wrap our code inside a `try: ... except: ...` block.

Example of program output:

```
Interface:    PC1633 - Coaxlink Quad G3 (2-camera) - KQG00014
Device:       Device0
Resolution:   4096x4096
```

Differences between C++ and Python eGrabber

eGrabber classes

C++	Python
`EGrabber<>`	`EGrabber`
`EGrabber<CallbackOnDemand>`	`EGrabber`
`EGrabber<CallbackSingleThread>`	-
`EGrabber<CallbackMultiThread>`	-
`Buffer`	`Buffer`
`ScopedBuffer`	`Buffer` used in a with-block

EGrabber methods

- In the C++ API, method names are written in `camelCase`. In Python, method names are written in `lower_case`. For example, the C++ `reallocBuffers` becomes `realloc_buffers` in Python.
- Other differences:C++PythongetInteger<MODULE>(f)module.get(f, dtype=int)getFloat<MODULE>(f)module.get(f, dtype=float)getString<MODULE>(f)module.get(f, dtype=str)getStringList<MODULE>(f)module.get(f, dtype=list)-module.get(f, dtype=bool)-module.get(f)setInteger<MODULE>(f, v)module.set(f, v)setFloat<MODULE>(f, v)module.set(f, v)setString<MODULE>(f, v)module.set(f, v)execute<MODULE>(f)module.execute(f)where
  - MODULE is SystemModule, InterfaceModule, DeviceModule, StreamModule, or RemoteModule;
  - module is system, interface, device, stream, or remote.

11. Sample programs

Sample programs for eGrabber are provided in a dedicated package named `egrabber-<OS>-sample-programs-<YY.MM.RE.BU>.<EXT>` where `<OS>` is the operating system (`linux`, `macos`, or `win`) and `<YY.MM.RE.BU>` is the version number of the package.

These sample programs will use Coaxlink by default. To use them with Grablink:

- either define EURESYS_DEFAULT_GENTL_PRODUCER=grablink in your environment;
- or pass Grablink() to the EGenTL constructor.

Similarly, to use Gigelink:

- either define EURESYS_DEFAULT_GENTL_PRODUCER=gigelink in your environment;
- or pass Gigelink() to the EGenTL constructor.

And for Playlink (which requires a license[1]):

- either define EURESYS_DEFAULT_GENTL_PRODUCER=playlink in your environment;
- or pass Playlink() to the EGenTL constructor.

Sample program	Description	Language	OS
`cpp/egrabber-snippets`	Collection of " eGrabber C++ code snippets" on page 63 for eGrabber	C++	Windows, Linux, macOS
`cpp/display-latest-buffer`	Win32 application showing image acquisition and display, discarding buffers when processing is slower than image acquisition	C++	Windows
`cpp/egrabber-mfc`	MFC application showing image acquisition and display	C++	Windows
`cpp/sdl2/display-all-buffers`	SDL2 application showing image acquisition and display of all acquired buffers	C++	Windows, Linux, macOS
`cpp/sdl2/display-latest-buffer`	SDL2 application showing image acquisition and display, discarding buffers when processing is slower than image acquisition (using OnDemand callback model)	C++	Windows, Linux, macOS
`cpp/sdl2/display-latest-multipart`	SDL2 application showing image acquisition and display based on display-latest-buffer, suited to multi-part sources such as GenDC, discarding buffers when processing is slower	C++	Windows, Linux, macOS
`cpp/sdl2/display-latest-buffer-mt`	SDL2 application showing image acquisition and display, discarding	C++	Windows, Linux, macOS

[1] PC4401 eGrabber Recorder and Playlink or PC4401-EV eGrabber Recorder and Playlink (30-day evaluation).

Sample program	Description	Language	OS
	buffers when processing is slower than image acquisition (using MultiThread callback model)		
cpp/sdl2/egrabber-cuda-sdl2	SDL2 application showing image acquisition with eGrabber and processing with CUDA (on Nvidia GPU) Supported CUDA versions: v12.x, v13.x	C++	Windows, Linux
cpp/amd-direct-gma	OpenGL application showing image acquisition, direct transfer to AMD GPU memory, and display	C++	Windows
cpp/nvidia-cuda	OpenGL console application showing image acquisition with eGrabber and processing with CUDA (on Nvidia GPU) Supported CUDA versions: from v8.x up to v11.5	C++	Windows, Linux
cpp/ffc-wizard	Console application showing how to compute coefficients for the Coaxlink FFC (flat-field correction)	C++	Windows, Linux, macOS
cpp/grablink-serial-communication-mfc	Simple application demonstrating Camera Link serial communication through the clseregl library on a Grablink Duo board	C++	Windows
cs/egrabber	Console application showing how to use eGrabber and callbacks in C#	C#	Windows
cs/grabn	Console application showing image acquisition	C#	Windows
cs/grabn.NET	.NET console application showing image acquisition	C#	Windows
cs/display-latest-buffer	Windows Forms application showing image acquisition and display, discarding buffers when processing is slower than image acquisition	C#	Windows
cs/egrabber-wpf	WPF application showing image acquisition and display	C#	Windows
cs/egrabber-cuda.NET	.NET console application showing image acquisition with eGrabber and processing with CUDA (on Nvidia GPU)	C#	Windows, Linux
cs/*-recorder*	Collection of " Sample programs" on page 61" eGrabber Recorder C# sample programs" on page 66 for eGrabber Recorder	C#	Windows
cs/grablink-serial-	Simple application demonstrating	C#	Windows

Sample program	Description	Language	OS
communication	Camera Link serial communication through the clseregl library on a Grablink Duo board		
python/*	Collection of " Sample programs" on page 61" eGrabber Python sample programs" on page 65 for eGrabber	Python	Windows, Linux, macOS
python/*-recorder	Collection of " Sample programs" on page 61" eGrabber Recorder Python sample programs" on page 66 for eGrabber Recorder	Python	Windows
python/display-all-buffers*	Collection of extra " Sample programs" on page 61" eGrabber Python sample programs" on page 65 for eGrabber showing how to process acquired data with numpy, opencv, Pillow, etc.	Python	Windows, Linux, macOS
python/display-latest-buffer	Simple application showing image acquisition and display, discarding buffers when processing is slower than image acquisition	Python	Windows, Linux, macOS
vb/grabn	Console application showing image acquisition	VB.NET	Windows
vb/display-latest-buffer	Windows Forms application showing image acquisition and display, discarding buffers when processing is slower than image acquisition	VB.NET	Windows

Additional files	Description	Language	OS
tools/stripeGeometry.py	Python script showing the effect of the image transfer settings: StripeArrangement, StripeHeight, StripePitch, StripeOffset, and BlockHeight	Python	Windows, Linux, macOS
LICENSE	License text for eGrabber sample programs	All	Windows, Linux, macOS

eGrabber C++ code snippets

cpp/egrabber contains the following code snippets:

Snippet	Description
100-grabn	Simple Grab N frames using ScopedBuffer class
101-singleframe	Single frame grabbing using ScopedBuffer class
102-action-grab	Single frame triggered by an action
105-area-scan-grabn	Set image size and Grab N frames (area-scan)

Snippet	Description
106-line-scan-grabn	Set image size and Grab N frames (line-scan)
110-get-string-list	Basic usage of EGrabber method getStringList
120-converter	Measure FormatConverter speed
130-using-buffer	Simple Grab N frames using Buffer class
140-genapi-command	Queries on GenApi commands
150-discover	Discover and create eGrabbers or cameras with EGrabberDiscovery
200-grabn-callbacks	Grab N frames and get DataStream events with callbacks
201-grabn-pop-oneof	Grab N frames and get DataStream events using pop (OneOf<>)
210-show-all-grabbers	Show available grabbers
211-show-all-grabbers-ro	Show available grabbers (devices are opened with DEVICE_ACCESS_READONLY)
212-create-all-grabbers	Create available grabbers
220-get-announced-handles	Get info and handles of announced buffers
221-queue-buffer-ranges	Create and use 2 sets of buffers configured differently
230-script-vars	Pass data between native code and Euresys script
231-script-var	Create and use virtual features from native code and Euresys scripts
240-user-memory	Grab into user allocated buffer
241-multi-part	Grab N multi-part buffers using Buffer class
250-using-lut	Configure and enable the LUT processor
260-recorder-read-write	Write/Read buffers to/from a Recorder container
261-recorder-parameters	Show Recorder parameters
270-multicast-master	Sending packets on multicast group
271-multicast-receiver	Save the image received on the multicast group
300-events-mt-cic	CIC events on EGrabber Multi-Thread Configuration
301-events-st-all	All events on EGrabber Single-Thread Configuration
302-cxp-connector-detection	Show CoaXPress events related to connection and device discovery
310-high-frame-rate	Grab in high frame rate mode for 10 seconds
311-high-frame-rate	Process images as soon as available in high frame rate mode for 10 seconds
312-part-timestamps	Show timestamp of each buffer part in HFR mode
320-cl-serial-cli-genapi	Command-line interface using GenApi commands for serial communication with a Camera Link camera
321-gencp-serial	Simple Grab N frames with a GenCP camera
330-metadata-insertion	Insert buffer and line metadata into a buffer and get them
340-dma-roi	Grab N frames but store a smaller region in the user

Snippet	Description
	buffers
341-dma-deinterlace	Grab and deinterlace N frames
342-dma-roi-deinterlace	Grab N frames but store a deinterlaced smaller region in the user buffers
500-grabn-cuda-process	Grab N frames and process them with CUDA operations
501-all-grabbers-cuda-process	Use all available interfaces and devices to grab N frames and process them with CUDA operation
502-grabn-cuda-copy-and-process	Grab N frames, copy the buffers to the CUDA device and process them with CUDA operations
503-grabn-cuda-rdma-process	Grab N frames in the GPU memory with RDMA and process them with CUDA operations
504-grabn-cuda-dma-buf-fd-process	Grab N frames in the GPU memory with DMA-BUF fd and process them with CUDA operations
600-thread-start-stop-callbacks	Perform specific operations on a callback thread when it starts/stops
610-line-scan-array	Array of (contiguous) buffers on Line-Scan with EGrabber Single-Thread
620-multiple-camera	Acquire data from all cameras
650-multistream	Acquire data from 4 data streams on the same device
700-memento	Generate memento waves
800-process-latest-buffer	Simulate a busy environment and acquire images, discarding some buffers when busy

eGrabber Python sample programs

Sample	Description
100-grabn	Simple Grab N using ‘with Buffer’
120-converter	Python version of the C++ 120-converter eGrabber sample program
130-using-buffer	Simple Grab N with manual buffer management
140-genapi-command	Queries on GenApi commands
150-discover	Discover and create eGrabbers or cameras with EGrabberDiscovery
200-grabn-callbacks	Grab N frames and get DataStream events with callbacks
201-grabn-pop-oneof	Grab N frames and get DataStream events with pop_one_of
210-show-all-grabbers	Show available grabbers
240-user-memory	Grab into user allocated buffer
300-events-mt	Grab frames for a few seconds and get DataStream events with callbacks, processing them in a separate thread
310-high-frame-rate	Grab in high frame rate mode for 10 seconds
320-cl-serial-cli-genapi	Command-line interface using GenApi commands for serial communication with a Camera Link camera

Sample	Description
322-cl-serial-cli-clseregl	Command-line interface using the clseregl library for serial communication with a Camera Link camera
cuda-grab-and-invert	Allocate an eGrabber buffer mapped into the CUDA device memory and run a kernel to invert its luminance values
cuda-rdma	Allocate eGrabber buffers directly in a CUDA device memory using eGrabber NvidiaRdmaMemory
display-all-buffers	Image acquisition and display
display-all-buffers-capture-opencv	Acquire and convert frames to RGB8 to produce an avi file with opencv and numpy
display-all-buffers-numpy-opencv	Create numpy arrays from acquired Mono8 data, transpose arrays and use opencv to show images
display-all-buffers-tkinter-pillow	Simple tkinter application showing acquired data processed by a Pillow contour filter
display-latest-buffer	Image acquisition and display. When the acquisition is faster than the display processing, buffers are discarded

eGrabber Recorder Python sample programs

Sample	Description
260-recorder-read-write	Write/Read buffers to/from a Recorder container
261-recorder-parameters	Show Recorder parameters
262-recorder-export	Export images from the container created by sample260.py to an MKV file, and then use opencv to read the MKV file and display the images
360-recorder-write-with-callback	Write to a Recorder container using EGrabber callback

eGrabber Recorder C# sample programs

Sample	Description
260-recorder-read-write	Write/Read buffers to/from a Recorder container
261-recorder-parameters	Show Recorder parameters

12. GenTL producers configuration

The eGrabber configuration files allow to configure the internal behavior of some functionalities. There is one configuration file per GenTL producer: `coaxlink.ini`, `grablink.ini`, `gigelink.ini` and `playlink.ini`. For Playlink, there is also a file `playlink.cfg`, which serves to define its data streams from eGrabber Recorder containers.

The configuration files are located in:

- Windows: %PUBLIC%\Documents\Euresys\eGrabber (C:\Users\Public\Documents\Euresys\eGrabber)
- Linux: /etc/opt/euresys/egrabber
- macOS: /usr/local/etc/opt/euresys/egrabber

These files follow the INI format of `key=value` pairs inside a `[Section]`. Depending on the producer, different sections and parameters are available.

The values read from the configuration files can be consulted in the `System` module, under the `ConfigurationFile` category.

coaxlink.ini and grablink.ini

- [Genapi]/MathParserMode: Configure the behavior of GenApi SwissKnife parser
  - Standard: permissive mode that operates like the GenICam reference implementation (left associativity)
  - Strict: evaluation of the formulas must be non ambiguous, using parentheses is required when several logical operators are used at the same level
  - Auto: same as Strict
- [System]/InterfaceOrder: Sort the discovered interfaces according to a specific ordering. Possible values are:
  - Ascending: sort interfaces based on the card ID, in ascending order
  - Descending: sort interfaces based on the card ID, in descending order
  - PciPosition: sort interfaces based on the PCI position value
  - PciSlot: sort interfaces based on the slot ID (if available, Windows only, and subject to drivers and hardware limitations)
  - System: do not sort interfaces, use the default ordering obtained during discovery

Notes:

- the card ID is defined as the card full name concatenated with its serial number: ProductCode - Name (Variant) - SerialNumber
- the PCI position is an internal numeric value calculated from the bus and slot information

Sample coaxlink.ini

```
[System]
InterfaceOrder=Ascending

[Genapi]
MathParserMode=Auto
```

gigelink.ini

- [Genapi]/MathParserMode:
  - Standard: permissive mode that operates like the GenICam reference implementation (left associativity)
  - Strict: evaluation of the formulas must be non ambiguous, using parentheses is required when several logical operators are used at the same level
  - Auto: same as Standard
- [Interface]/CameraSubnet: restrict discovery to cameras whose IPv4 address matches the given IPv4 subnet, e.g. 192.168.100.0/24 for cameras ranging from 192.168.100.1 to 192.168.100.254.
- [Interface]/DiscoveryTimeout: define how long to wait for a discovery reply (value in milliseconds)
- [Interface]/AllowBroadcastAck: devices are allowed to send discovery replies in IPv4 broadcast packets. This helps the discovery of cameras with misconfigured network (y/Y or n/N)
- [Stream]/UseFilterDriver: enable the use of gigelink.sys filter driver for GVSP protocol processing (y/Y or n/N)
- [Stream]/PacketResend: enable packet retransmission with the given maximum number of requests per block (n/N for disabled, an integer value otherwise)
- [Stream]/GenDCExtraPackets: extra packets to use in GenDC container transmission (integer value)
- [Stream]/RdmaConnectionTimeout: define how long to wait for device acknowledge during NetworkDirect+InfiniBand connection establishment (value in milliseconds)
- [Stream]/ForceRdma: try to create a RDMA stream in the given list of IP addresses (a comma-separated list of IP addresses)

Sample gigelink.ini

```
[Genapi]
MathParserMode=Auto

[Interface]
CameraSubnet=169.254.0.0/16
DiscoveryTimeout=2500
AllowBroadcastAck=n

[Stream]
UseFilterDriver=y
PacketResend=5
```

```
GenDCExtraPackets=2
RdmaConnectionTimeout=3500
ForceRdma=169.254.0.3,169.254.0.4
```

playlink.ini

- [Genapi]/MathParserMode: Configure the behavior of GenApi SwissKnife parser
  - Standard: permissive mode that operates like the GenICam reference implementation (left associativity)
  - Strict: evaluation of the formulas must be non ambiguous, using parentheses is required when several logical operators are used at the same level
  - Auto: same as Strict

Sample playlink.ini

```
[Genapi]
MathParserMode=Auto
```

playlink.cfg

```
# Playlink configuration file
# ---------------------------
#
# Playlink (playlink.cti) is a Euresys GenTL producer that turns Euresys Recorder
# containers into GenTL data streams.
#
# This config file allows you to configure playlink.cti.
# Each line starting with an '#' is a comment line and is ignored.
# Blank lines are ignored too.
# Otherwise, each line maps a GenTL data stream to a Euresys Recorder container
# path.
# The GenTL hierarchy of the data stream is explicitly defined by the Interface,
# Device and Stream module IDs.
# The next line shows the syntax used to declare and configure the GenTL modules:

# /{Interface name}/Device{id}/Stream0 = {Path to Euresys Recorder container}

# - {Interface name} is to be replaced with the desired name.
#   Valid characters for the name of the interface are:
#   + Latin letters (no diacritics)
#   + Digits
#   + The following special characters: '+', '-', '(', ')', ' '
# - {id} is to be replaced with the id of the device (only digits are allowed)
# - {Path to Euresys Recorder container} is to be replaced with the actual path
#   to the Euresys Recorder container.
#   The path may contain any character except these ones:
#   + Question mark: '?'
#   + Carriage return: '\r'
#   + Line feed: '\n'
#   An absolute path is expected.
# - Since the stream name is fixed, there can only be one stream per device.

# For example:
# /My Interface/Device0/Stream0 = E:\Containers\foo

# Will create an interface named "My Interface", a device named "Device0" and
# a datastream named "Stream0". The created stream will output buffers from
```

```
# the Euresys Recorder container at "E:\Containers\foo"

# An interface can contain one or more devices, e.g.:

# /My (Interface)/Device42/Stream0=/mnt/containers/bar
# /My (Interface)/Device666/Stream0 = /mnt/containers/baz

# Each configuration line can also declare key-value attributes that further configure
# the device.
# Attributes are introduced by a '?' right after the path.
# Each attribute is a key-value pair of the form {key}={value}.
# Attributes are separated by a '&'.
# Here is an example:

# /Iface/Device0/Stream0 =
C:/con/te/partiro?PlaybackLoop=true&PlaybackFrameRateMode=auto&PlaybackFrameRateMultiplier=2

# There exists many attributes and they expect values of different types:
# "PlaybackLoop": {true, false}
# "PlaybackMode": {container, chapter}
# "PlaybackFrameRateMode": {auto, fixed}
# "PlaybackFixedFrameRate": float
# "PlaybackFrameRateMultiplier": float
# "RewindOnStop": {true, false}
```

13. Definitions

Acronyms

CIC
Camera and Illumination Controller

CTI
Common Transport Interface

CXP
CoaXPress

EMVA
European Machine Vision Association

PFNC
Pixel Format Naming Convention

SFNC
Standard Features Naming Convention

Glossary

Buffer module

GenTL module that represents a memory buffer. Buffers must be announced to the data stream that will fill them with image data.

Callback model

Defines when and where (in which thread) callback functions are executed.

One of `CallbackOnDemand`, `CallbackSingleThread`, `CallbackMultiThread`.

Camera and illumination controller

Component of Coaxlink and Grablink cards that controls a camera and its associated illumination devices.

Hosted in the device module.

CoaXPress

High-speed digital interface standard that allows the transmission of data from a device (e.g., a camera) to a host (e.g., a frame grabber inside a computer) over one or several coaxial cables or optical fibers.

Common transport interface

GenTL producer.

Data stream module

GenTL module that handles buffers.

Device module

GenTL module that contains the frame grabber settings relating to the camera.

Parent of the data stream module.

Sibling of the *remote device*.

GenApi

The GenICam standard that deals with camera and frame grabber configuration.

GenApi feature

Camera or frame grabber feature, defined in a register description.

Either a *set/get* parameter, or a command with side effects.

GenICam

Set of EMVA standards. It consists of GenApi, GenTL, the SFNC and the PFNC.

GenTL

The GenICam standard that deals with data transport. TL stands for Transport Layer.

GenTL producer

Software library that implements the GenTL API.

File with the `cti` extension (e.g., `coaxlink.cti`, `grablink.cti`, `gigelink.cti`, or `playlink.cti`).

Grabber

An `EGrabber` instance.

Info command

Numerical identifier used to query a specific piece of information from a GenTL module. Info commands are defined either in the standard GenTL header file, or in a vendor-specific header file (e.g., info commands specific to Coaxlink, Grablink, and Gigelink are defined in `include/GenTL_EuresysCustom.h`).

Interface module

GenTL module that represents a frame grabber.

Parent of the device module.

I/O toolbox

Component of Coaxlink and Grablink cards that controls digital I/O lines and implements tools such as rate converters, delay lines, etc.

Hosted in the interface module.

Multi-bank camera

Camera composed of several independent CoaXPress sub-Devices, usually organized as a *master* sub-Device and one or more *slave* sub-Devices.

Register description

XML file mapping low-level hardware registers to camera or frame grabber features.

Remote device

Camera connected to a frame grabber.

The term *remote* is used to distinguish this from the GenTL device module.

Sub-grabber

Part of an `EGrabber` instance for a multi-bank camera.

A camera composed of `N` banks is operated via an `EGrabber` instance that internally uses `N` sub-grabbers.

System

GenTL module that represents the GenTL producer.

Also known as TLSystem.

Parent of the interface module.

Timestamp

The time at which an event occurs.

For Coaxlink, Grablink, and Gigelink, timestamps are always 64-bit integers and are expressed as the number of microseconds that have elapsed since the computer was started.