EURESYS™
Excellence in vision

程序员指南

Coaxlink

Coaxlink 10.5.2



PCI EXPRESS™

CoaXPress

© EURESYS s.a.2019 - Document D204CN-Programmer Guide-Coaxlink-10.5.2.2065 built on 2019-03-06

使用条款

EURESYS s.a. 应保留硬件和软件文档以及 EURESYS s.a. 商标的所有财产权、所有权和利益。

文档中提及的所有公司和产品的名称可能是其各自所有者的商标。

未经事先通知，不得对本书中包含的 EURESYS s.a 的 硬件或软件、品牌或文档进行许可、使用、出租、租赁、翻译、复制、复印或修改。

EURESYS s.a. 可能随时自行修改产品规格或更改本文档中给出的信息，恕不另行通知。

EURESYS s.a. 对于使用其硬件或软件而引起的任何类型的收入、利润、商誉、数据、信息系统损失或损害，或与使用其硬件或软件相关的，或因本文档遗漏或错误造成的其他特殊的、偶然的、间接的、后果性的或惩罚性的损害赔偿，概不负责。

本文档随Coaxlink10.5.2(doc build) 提供。
2065© 2019 EURESYS s.a.

目录

1. 简介 5

2. GenApi 6

3. GenTL 8

3.1. 系统模块 9
3.2. 接口模块 9
3.3. 设备模块 9
3.4. 数据流模块 9
3.5. 缓冲模块 10
3.6. GenTL API 10

4. Euresys::EGenTL 11

4.1. 第一个例子 12
4.2. 相关文件 12

5. Euresys::EGrabber 14

5.1. 第一个例子 15
5.2. 获取图像 16
5.3. 配置捕获器 18
5.4. 活动 19
背景 19
计数器 20
通知 20
回调函数 21
事件识别 22
示例 23
5.5. Egraber特点 23
5.6. 事件和回调示例 24
按需回调 24
单线程和多线程回调 26
新缓冲区回调 27
5.7. 相关文件 28

6. Euresys GenApi 脚本 29
6.1. doc/BasICS 29
6.2. doc/BufftIsj.js 35
6.3. doc/grabbers.js 36
6.4. DOC/MODEL1.JS 38

7. 适用于 MultiCam 用户的 EGrabber 39

8. .NET程序集 45
8.1. 第一个例子 45
8.2. C++与.NET EGrabber的区别 46
8.3. 单线程回调 47

9. 定义 49

1. 简介

Coaxlink卡的*应用编程接口*(API) 是基于GenICam的。

GenICam 的目标是提供一个标准化、统一的编程接口，用于基于不同物理接口 (CoaXPress, GigE Vision等) 或来自不同供应商的相机和帧捕获器。

GenICam 是一组EMVA标准(GenApi和GenTL) ，以及命名事物的相关约定(标准特征的 SFNC，像素格式的PFNC) 。

- GenApi是关于描述的。GenApi的核心是*寄存器描述*的概念。寄存器描述以XML文件的形式提供。它们将低级硬件寄存器映射到高级*功能*。
  GenApi 允许应用程序以统一且一致的方式检测、配置和使用相机和帧捕获器的功能。
- Gentl是关于数据传输的。TL后缀表示*传输层*。
  GenTL 标准定义了一组C函数和数据类型，用于枚举、配置，以及从相机和帧捕获器中获取图像。此API由C头文件定义。
  帧捕获器供应商提供实现此API的库(即导出标准头文件中声明的函数的库) 。这些库被称为*GenTL发生器*或*公共传输接口*(CTI) ，并使用`cti`文件扩展名。Coaxlink卡的GenTL发生器是`coaxlink.cti`。

本文件应从头到尾进行阅读。每一章和节都是建立在前面的章节的基础上的。如果跳过部分文本，某些解释和示例可能会显得晦涩难懂。如果出现这种情况，您应该返回并阅读跳过的部分。

2. GenApi

Genapi解决了配置相机的问题。实现的方式是通用的，并且适用于不同类型的设备，包括帧捕获器。在本章中，我们对相机的论述，同样也适用于帧捕获器。

GenApi需要两件事情来工作：一个*寄存器描述*，一个是*GenApi实现*。

寄存器描述

寄存器描述，是一个XML文件，可以看作是相机的计算机可读数据表。它定义相机设置(比如`PixelFormat`和`TriggerSource`)，以及对它们进行配置的说明(例如，将`ExposureMode`设置为`Timed`，将值`0x12`写入寄存器`0xE0140`)。它还可以包含相机文档。

GenApi 实现

*GenApi 实现*是一个可以读取和解释寄存器描述文件的软件模块。

EMVA提供了一个参考实现，但使用起来相当困难，而且日志记录也很差。取而代之，我们建议使用与Coaxlink软件包捆绑在一起的Euresys实现。此实现还允许编写功能强大的配置脚本。

功能

用户从GenApi中获得的是一系列(按类别组织的)命名*功能*。

设置/获取功能

设置/获取功能是简单的设置(称为*参数*-在MultiCam中)，可以是不同的类型：

- 整数(例如`Width`)
- 浮点(例如`AcquisitionFrameRate`)
- 枚举(例如`PixelFormat`)
- 布尔值(`LUTEnable`例如)
- 字符串(`DeviceVendorName`例如)

可以使用*get/set*函数，以检索/修改功能值。有些功能是只读的，有些是只写的，但大多数都允许读/写访问。

命令

还有另一种功能：*命令*（例如`AcquisitionStart`）。命令是特殊的：它们没有任何相关联的值；它们有副作用。命令功能是要被*执行*。执行命令时，相机中会发生一些动作（例如，生成软件触发）。显然*get/set*函数对命令没有意义，不能使用。

3. GenTL

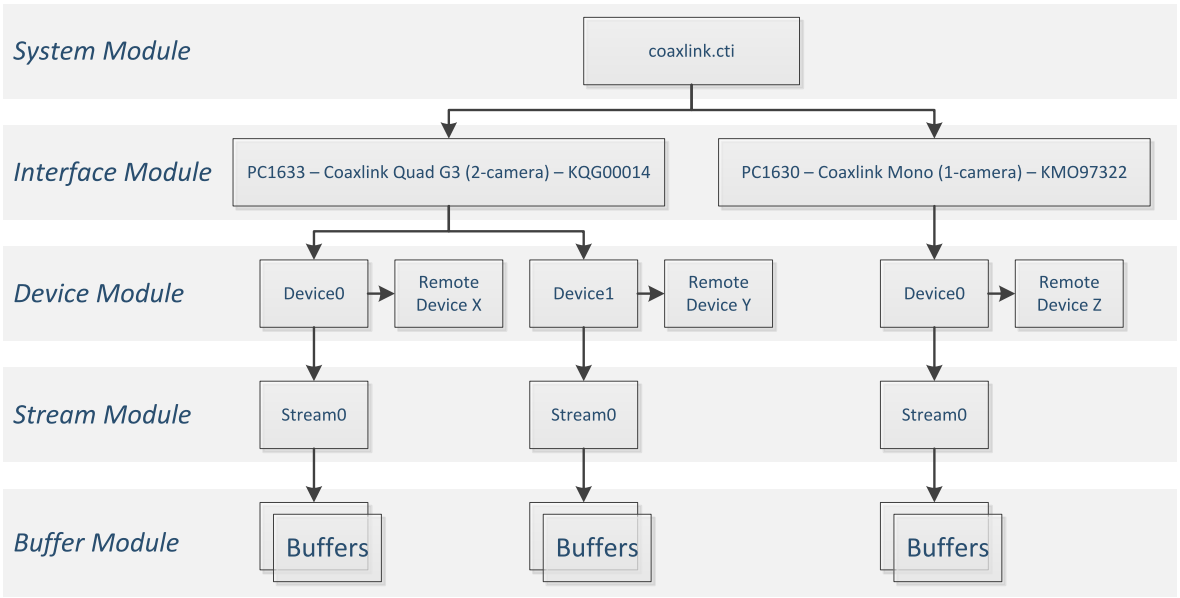
Gentl定义了5种对象类型，按父/子关系组织：

1. *系统*模块
2. *接口*模块
3. *设备*模块
4. *数据流*模块
5. *缓冲*模块

每个模块：

- 对应于系统的特定元件；
- 定义可以查询的相关信息(*信息命令*)(使用*get info*函数)；
- 允许行使该模块的功能(使用特定功能)。

此外，除缓冲模块外的所有模块，都充当允许读/写操作的*端口*。GenApi使用这些端口函数加载该模块的描述文件，并使用其GenApi功能。



Gentl模块层次结构

3.1. 系统模块

系统模块(也称为*TLSystem*)代表Gentl发生器(`coaxlink.cti`库)。此模块位于父/子树的顶部。

系统模块提供关于Gentl发生器的基本信息:例如完整的`coaxlink.cti`路径和供应商名称(Euresys)。

系统模块的真正目的,是列出系统中存在的*接口*(或帧捕获器)。系统模块最重要的功能是`TLGetNumInterfaces`(检索系统中的帧捕获器数量)和`TLOpenInterface`(访问其中一个帧捕获器)。

3.2. 接口模块

GenTL标准调用帧抓取器*接口*。系统模块为每个帧捕获器提供一个子接口:如果计算机中有两个Coaxlink卡,系统模块将提供两个子接口。

每个接口代表一个帧捕获器。接口模块中有全局帧捕获功能,如数字I/O线。这意味着控制I/O线的GenApi功能连接到接口。

每个接口还充当一个或多个*设备*的父接口。接口模块最重要的功能是`IFGetNumDevices`(检索可连接到接口的相机数量)和`IFOpenDevice`(访问其中一个设备)。

3.3. 设备模块

GenTL标准使用术语*设备*和*远程设备*来表示两个相关但不同的概念。*远程设备*是一个真正的相机,实际地连接到一个帧捕获器。这与我们在这里描述的设备模块不同。

设备模块是包含与相机相关的帧捕获器设置的模块。这包括触发器和频闪器之类的东西。

设备模块还充当一个*数据流*的父级,并且可以被视为远程设备的*同级*。设备模块最重要的功能是`DevOpenDataStream`(访问数据流)和`DevGetPort`(访问远程设备)。

3.4. 数据流模块

数据流模块处理*缓冲区*。在采集运行期间,图像从相机发送到帧捕获器,然后将其传输到主机上分配的内存缓冲区。数据流模块是图像采集发生的地方。这是大多数功能所在的地方。

缓冲处理非常灵活。可以使用任意数量的缓冲区。缓冲区要么在输入队列中,要么在输出队列中,要么暂时取消。应用程序决定空缓冲区何时排队(到输入FIFO),以及填充的缓冲区何时弹出(从输出FIFO)。

3.5. 缓冲模块

缓冲模块简单地表示给父数据流的存储器缓冲区。有用的元数据与缓冲区关联。这包括图像宽度、高度、像素格式、时间戳...这些是通过*info*命令检索的(请参见`BUFFER_INFO_CMD_LIST`，在标准GenTL 头文件中)。

缓冲模块是唯一没有读/写端口功能的模块;它没有GenApi功能。

3.6. GenTL API

GenTL可以检测、控制和使用所有相机和图像捕获器功能，但其使用非常繁琐:

- `cti` 必须动态加载文件，并且必须通过指针访问它们导出的函数。
- 函数返回必须由应用程序检查的错误代码。
- 大多数函数从非类型缓冲区读取/向非类型缓冲区写入:应用程序必须确定所需的缓冲区大小，分配临时缓冲区，将数据转换到此缓冲区/从此缓冲区转换数据，最后释放缓冲区内存。

我们不建议直接使用GenTL API，而是建议使用以下两种方法之一:

- 处理这些复杂问题的`Euresys::EGenTL`库，以便用户不必这样做;
- 或者提供高级、易用界面的`Euresys::EGrabber`库。

4. Euresys::EGenTL

Euresys::EGenTL 是一个C++类库，提供与标准GenICam GenTL相同的功能，但具有更友好的用户界面。例如，它使用了std::string，而不是原始char指针，并且将错误代码转换为异常。Euresys::EGenTL还负责定位GenTL发生器，并加载其导出的功能。

这个库完全在C++头文件中实现。因此，您只需包括相关的头文件：

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

与GenTL定义的原始、低级C函数不同，我们得到一个表示Gentl发生器的Euresys::EGenTL对象：

- 每个GenTL函数都可用作Euresys::EGenTL的成员方法。GenTL 函数名以大写前缀开头。在Euresys::EGenTL中，方法名以相同的前缀开头，但以小写字母书写。例如，GCReadPort函数作为gcReadPort方法公开，函数TLOpenInterface作为tlOpenInterface方法公开。
- 所有GenTL 函数都返回一个指示成功或失败的代码GC_ERROR。当函数返回GC_ERR_SUCCESS以外的代码时，将引发异常。这消除了在每次函数调用后，人工检查错误代码的负担。
- 由于GenTL函数返回一个GC_ERROR，输出值通过作为参数传递的指针返回。Euresys::EGenTL方法提供更自然的接口；它们直接返回输出值：

```
GC_API TLGetNumInterfaces(TL_HANDLE hTL, uint32_t *piNumIfaces);
```

```
uint32_t tlGetNumInterfaces(TL_HANDLE tlh);
```

> **注意：** *GC_API 定义为GC_IMPORT_EXPORT GC_ERROR GC_CALLTYPE。它只是一个使用调用约定和DLL导入/导出属性修饰的GC_ERROR。*

- 对于处理文本的GenTL函数，相应的Euresys::EGenTL方法从char *转换为std::string，反之亦然：

```
GC_API TLGetInterfaceID(TL_HANDLE hTL, uint32_t iIndex,
                        char *sID, size_t *piSize);
```

```
std::string tlGetInterfaceID(TL_HANDLE tlh, uint32_t index);
```

- 一些GenTL函数检索有关相机或帧捕获器的信息。这些函数用一个值填充void *缓冲区，并在INFO_DATATYPE中，指示该值的实际类型。Euresys::EGenTL使用C++模板使这些函数易于使用：

```
GC_API GCGetInfo(TL_INFO_CMD iInfoCmd, INFO_DATATYPE *piType,
                 void *pBuffer, size_t *piSize);
```

```
template<typename T> T gcGetInfo(TL_INFO_CMD cmd);
```

4.1. 第一个例子

此程序使用Euresys::EGenTL来迭代系统中存在的Coaxlink卡，并显示它们的id：

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

void listCards() {
    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. 包含EGenTL.h，其中包含Euresys::EGenTL类的定义。
2. 创建一个Euresys::EGenTL对象。这涉及以下操作：
   - 定位并动态加载Coaxlink GenTL 生成器()；coaxlink.cti
   - 检索指向coaxlink.cti导出的函数的指针，并通过Euresys::EGenTL方法使它们可用；
   - 初始化coaxlink.cti(这是通过调用GenTL初始化函数GCInitLib完成的)。
3. 打开Gentl生成器。这将返回GenTL::TL_HANDLE类型的句柄。GenTL命名空间是在标准GenTL头文件中定义的，该头文件在步骤1中已经被EGenTL.h自动包含。
4. 找出系统中有多少张卡。
5. 检索第N张卡的ID。
6. Euresys::EGenTL 使用异常来报告错误，并将代码封装在一个try ... catch块中。

程序输出示例

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

4.2. 相关文件

include/EGenTL.h	主标题。包括所有其他标题。定义 Euresys::EGenTL
include/GenTL_v1_5.h	标准GenTL标题。定义标准类型、函数和常量。

include/GenTL_v1_5_EuresysCustom.h	定义特定Coaxlink常量

5. Euresys::EGrabber

Euresys::EGrabber 是提供高级接口的C++类库。它是构建在Euresys::EGenTL库的顶部，建议大多数用户使用。

还提供了在Euresys::EGrabberC++类之上构建的.NET程序集。在本文中，我们主要关注C++ API。C++和.NET接口之间的细微差别在专章中列出。

要使用这里描述的类，需要包含主Euresys::EGrabber文件：

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

Euresys::EGrabber 是一个仅包含头的库(不作为lib或dll文件提供)。它包括几个类，其中最重要的类也被命名为Euresys::EGrabber：

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

在本文中，我们将这个类称为*捕获器*。捕获器封装一组相关的GenTL模块：

- 接口：表示全局(共享)帧捕获器设置和功能的模块。包括数字I/O控制、PCIe和固件状态...
- 设备(或*本地*设备，而不是*远程*设备)：包含与相机相关的帧捕获器设置和功能的模块。这主要包括相机和照明控制功能：闪光灯，触发器...
- 数据流：处理图像缓冲区的模块。
- 一个远程设备：CoaXPress 相机。
- 若干缓冲区。

如果对这些概念不清楚，请回到关于GenTL模块的章节。

5.1. 第一个例子

此示例创建一个捕获器，并显示其包含的接口、设备和远程设备模块的基本信息：

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

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

void showInfo() {
    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. 包含EGrabber.h它定义了Euresys::EGrabber类，并包含我们需要的其他头文件(例如EGenTL.h和标准的GenTL头文件)。
2. 创建一个Euresys::EGenTL对象。这包含以下操作：
   - 定位并动态加载Coaxlink GenTL 生成器(coaxlink.cti)；
   - 检索指向coaxlink.cti导出的函数的指针，并通过Euresys::EGenTL方法使它们可用；
   - 初始化coaxlink.cti(这是通过调用GenTL初始化函数GCInitLib完成的)。
3. 创建一个Euresys::EGrabber对象。构造函数需要在步骤2中创建gentl对象。并把要使用的接口和设备的索引，作为可选参数。
   后面的尖括号的用途(<>)稍后EGrabber将变得清晰。就目前而言，可以忽略它们。
4. 使用"GenApi" 于页面6来查找Coaxlink卡的ID。Euresys::InterfaceModule指示我们需要来自接口模块的答复。

- 同样地，找出设备的ID。这一次，我们使用Euresys::DeviceModule瞄准设备模块。
- 最后，读取相机分辨率。Euresys::RemoteModule指示必须从相机中检索该值。
- Euresys::EGrabber 使用异常来报告错误，并将代码封装在一个try ... catch块中。

程序输出示例

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

5.2. 获取图像

此程序使用Euresys::EGrabber从连接到Coaxlink卡的相机获取图像：

```
#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. 创建一个Euresys::EGrabber对象。这里省略了构造函数的第二个和第三个参数。捕获器将使用系统中存在的第一接口的第一设备。
2. 分配3个缓冲区。捕获器自动确定所需的缓冲区大小。
3. 启动捕获器。在这里，我们要求捕获器填充10个缓冲区。如果我们不希望捕获器在特定数量的缓冲之后停止，我们可以这样grabber.start(GenTL::GENTL_INFINITE)，或者简单地grabber.start()。
   启动捕获机涉及以下操作：
   - 该AcquisitionStart命令在相机上执行；
   - 调用DSStartAcquisition函数，以启动数据流。

   在本例中，我们假设相机和帧捕获器正确配置。对于实际的应用程序，在开始采集之前(以及为此分配缓冲区之前)运行配置脚本会更安全。这将在另一个例子中显示。
4. 等待捕获器填满缓冲区。结果是Euresys::ScopedBuffer。术语*scoped*用于指示缓冲器的寿命，是当前作用域(即当前块)。

5. 检索缓冲区地址。这是通过调用缓冲区的getInfo方法来完成的。此方法以BUFFER_INFO_CMD作为参数。在这种情况下，我们请求BUFFER_INFO_BASE(标准 GenTL 头文件中定义)：

```
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;
```

请注意，getInfo这是一种模板方法，当我们调用它时，我们必须指定我们期望的值类型。BUFFER_INFO_BASE返回指针；这就是我们使用getInfo<void *>的原因。

6. 执行相同操作以检索缓冲区的时间戳。这次，我们使用uint64_t版本(getInfo)来匹配BUFFER_INFO_TIMESTAMP的类型。

   注意：*对于Coaxlink，时间戳总是64位的，并表示为计算机启动后流逝的微秒数。*

7. 我们到达了for块的终点。局部变量buf超出范围并被销毁：调用ScopedBuffer析构函数。这导致buf中包含的GenTL缓冲区,重新排队(返回)到捕获器的数据流中。

程序输出示例

```
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
```

我们可以看到,分配的三个缓冲区(A在0x7f3c32c54010，B在0x7f3c2c4bf010，以及C在0x7f3c2c37e010)以循环方式使用：a→b→c→a→b→c→...这是由于：

- 输入和输出缓冲队列的FIFO性质：
  - Coaxlink 驱动程序从输入队列的前面，弹出一个缓冲区，并将其提供给Coaxlink卡进行DMA传输；
  - 传输完成后，将缓冲区推送到输出队列的后面；
- ScopedBuffer的使用：
  - ScopedBuffer构造器从输出队列的前面弹出一个缓冲区(即，它占用最早的缓冲区)；
  - ScopedBuffer析构函数将缓冲区推到输入队列后面(因此，在输入队列中已经有所有缓冲区之后，这个缓冲区将用于新的传输)。

5.3. 配置捕获器

配置是任何图像采集程序的一个非常重要的方面。

- 相机和帧捕获器都必须根据应用要求进行配置。
- 相机配置必须与帧捕获器配置兼容，反之亦然。

配置，基本上归结为，在捕获器模块上执行的一系列设置/获取操作：远程设备(即相机)接口，设备、或数据流模模块。

该程序将捕获器配置为所谓的*RG*控制模式(异步复位相机控制、帧捕获器控制的曝光)。

```
#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.setString<Euresys::DeviceModule>("CameraControlMethod", "RG");              // 4
    grabber.setString<Euresys::DeviceModule>("CycleTriggerSource", "Immediate");        // 5
    grabber.setFloat<Euresys::DeviceModule>("CycleMinimumPeriod", 1e6 / FPS);           // 6
}

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

1. 在相机上启用触发。
2. 告诉相机在CoaXPress链路上查找触发。
3. 将相机配置为使用`TriggerWidth`曝光模式。
4. 将帧捕获器相机控制方法设置为`RG`。在此模式下，相机循环由帧捕获器启动，曝光的持续时间也由帧捕获器控制。
5. 告诉帧捕获器在不等待硬件或软件触发的情况下自行启动(以由`CycleMinimumPeriod`定义的速率)相机循环。
6. 配置帧速率。

但是有一种更好的方法来配置捕获器。使用脚本文件，程序变成：

```
#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;
    }
}
```

配置脚本是：

```
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.set("CameraControlMethod", "RG");
grabber.DevicePort.set("CycleTriggerSource", "Immediate");
grabber.DevicePort.set("CycleMinimumPeriod", 1e6 / FPS);
```

使用脚本文件有几个优点：

- 无需重新编译应用程序即可更改配置。这样可以缩短开发周期，并可以在实验室或现场更新配置。
- 配置脚本可以由GenICam 浏览器和命令行`gentl`工具加载。这使得在用户应用程序之外验证配置成为可能。
- 配置脚本可以很容易地被用不同编程语言编写的几个应用程序共享: C++、C语言、VB.NET...
- Euresys Genapi脚本的全部功能可用。

5.4. 活动

背景

CoaxLink卡生成不同类型的事件：

- *新缓冲区*事件：指示缓冲区已被数据流填充的事件。
- *数据流*事件：与数据流及其帧存储相关的事件。
- *相机和照明控制器*事件：与相机及其照明设备的实时控制(由设备执行)相关的事件。
- *I/O工具箱*事件：与数字I/O线和其他I/O工具相关的事件(来自接口)。
- *CoaXPress 接口*事件：与CoaXPress接口相关的事件(来自接口)。

*新缓冲区*事件在GenTL中是标准的。当帧捕获器填充缓冲区时，会发生这种情况。附加到*新缓冲区*事件的信息，包括缓冲区的句柄和时间戳。

其他类型的事件仅限于Coaxlink，可以视为特定事件的类别。例如，在*CIC*事件类别中，我们有：

- `CameraTriggerRisingEdge`（相机触发启动）
- `CameraTriggerFallingEdge`（相机触发结束）
- `StrobeRisingEdge`（闪光灯启动）
- `StrobeFallingEdge`（闪光灯结束）
- `AllowNextCycle`（CIC已为下一个相机循环做好准备）
- ...

在事件的*I/O工具箱*类别中，我们有：

- `LIN1`（线路输入工具1）
- `LIN2`（线路输入工具2）
- `MDV1`（乘数器/除法器工具1）
- ...

计数器

Coaxlink固件统计每个事件（*新缓冲区事件*除外）的每次发生，并使这个计数器在名为`EventCount`的GenApi功能中可用。每个事件都有自己的计数器，`EventCount`的值取决于所选事件：

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

或者，使用*选定功能*符号：

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

通知

正如我们刚才看到的，当一个事件发生时，一个专用的计数器就会递增。Coaxlink还可以通过`Euresys::EGrabber`执行用户定义的回调函数来通知应用程序此事件。但首先，需要启用一个或多个事件的通知：

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

或：

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

使用配置脚本，这很容易为所有事件启用通知：

```
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. 定义一个名为`enableAllEvents`的助手函数，并将模块(或端口)`p`作为参数。
2. 使用`$ee`函数,检索`EventSelector`可以获取的值列表。这是模块`p`生成的事件列表。(`ee`表示*枚举项*。)
3. 对于每个事件，启用通知。(`+`运算符连接字符串，所以，如果`e`是`'LIN1'`，则表达式`'EventNotification[' + e + ']'`的计算结果为`'EventNotification[LIN1]'`。)
4. 调用步骤1中为接口模块定义的`enableAllEvents`函数。这将启用*I/O工具箱*和*CoaXPress接口*类别中所有事件的通知。
5. 同样，启用来自设备模块的所有事件的通知(*CIC*事件)。
6. 最后，启用所有*数据流*事件的通知。

回调函数

当一个事件发生且为该事件启用了事件通知时，`Euresys::EGrabber`会执行几个回调函数中的一个。

这些回调函数是在重写的虚拟方法中定义的:

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

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

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

        // callback function for CIC events
        virtual void onCicEvent(const CicData &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) {
            ...
        }
};
```

正如你所看到的，可以为每一类事件定义不同的回调函数。

在.NET中，回调函数是通过创建委托(而不是重写虚拟方法)来定义的。在有关.NET程序集的章节中，将给出一个示例。

事件识别

当向应用程序通知事件时，执行的回调函数指示该事件的类别。实际发生的事件由一个数字ID标识，称为numid，并在include/GenTL_v1_5_EuresysCustom.h中定义：

```
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_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_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 */
    ...
};
```

作为参考，下表列出了(每个生成事件的模块和每个事件类别)与它们的关系：

- 回调函数的名称
- 传递给回调函数的数据类型
- 共同numid前缀

  注意： *下面是一个简单的命名方案：一个名为某个类别的事件类别，有一个名为on**SomeCategory**Event的回调函数，该函数将**SomeCategory**Data结构用作自变量，并使用EVENT_DATA_NUMID_**SOME_CATEGORY**_作为公共numid前缀。*

数据流模块-新缓冲区类别

回调函数	数据类型	numid 前缀
onNewBufferEvent	NewBufferData	-

注意： *在新的缓冲区事件类别中，只有一个事件，所以我们不需要numid。*

数据流模块-数据流类别

回调函数	数据类型	numid 前缀
onDataStreamEvent	DataStreamData	EVENT_DATA_NUMID_DATASTREAM_

设备模块-CIC类别

回调函数	数据类型	numid 前缀
onCicEvent	CicData	EVENT_DATA_NUMID_CIC_

接口模块-I/O工具箱类别

回调函数	数据类型	numid 前缀
onIoToolboxEvent	IoToolboxData	EVENT_DATA_NUMID_IO_TOOLBOX_

接口模块-CXP 接口类别

回调函数	数据类型	numid 前缀
onCxpInterfaceEvent	CxpInterfaceData	EVENT_DATA_NUMID_CXP_INTERFACE_

示例

我们很快就会展示一些完整的示例程序，说明事件和回调，但在此之前，还有一件事需要解释：执行回调函数的上下文。这是下一节的主题。

5.5. Egraber特点

什么时候应该调用回调函数？ 从哪个上下文(即，哪个线程)？ 考虑到这些问题，可以定义几个*回调模型*：

- 应用程序询问捕获器：*"您有缓冲区或事件给我吗？ 如果是，现在执行回调函数。"*这是一种轮询、同步操作模式，当应用程序请求回调时将执行回调(在应用程序线程中)。
  我们将此回调模型称为*按需*。
- 应用程序要求捕获器创建一个单独的专用线程，并在该线程中等待事件。当事件发生时，捕获器执行相应的回调函数，也在这个回调线程中。
  我们将这个回调模型称为*单线程*。
- 应用程序要求捕获器为每个回调函数创建一个专用线程。在每个线程中，捕获器都会等待特定类别的事件。当事件发生时，在该线程中执行相应的回调函数。
  我们将这个回调模型称为*多线程*。

这三种回调模型都有意义，每一种都最适合某些应用程序。

- *按需*模型是最简单的。虽然它的实现可能会使用工作线程，但是从用户的角度来看，它不添加任何线程。这意味着应用程序不需要担心线程同步、互斥等问题。
- 如果用户想要一个专用的回调线程，那么*单线程*模型会为他创建回调线程。当我们只有一个线程时，事情就很简单了。在这个模型中，抓取器用于(至少)两个线程，所以我们需要开始担心同步和共享数据。
- 在*多线程*模型中，每一类事件都有自己的线程。这样做的好处是一种类型的事件(及其回调函数的执行)不会延迟其他类型事件的通知。例如，在`onNewBufferEvent`中，执行大量图像处理的线程，不会延迟`onCicEvent`对CIC事件的通知(例如，指示曝光完成以及对象或相机可以移动的事件)。
  在该模型中，捕获器用于多个线程，因此同步的需求就像在*单线程*模型中一样。
  当然，应用程序还须知道，它可能会收到*X*类型事件的通知，这些通知比*Y*类型事件的通知要早，而Y类型的事件已经被接收和处理。毕竟，这是*单线程*和*多线程*模型之间的区别因素。

为了提供给用户最大的灵活性，我们支持所有三个回调模型。这就是为什么`Euresys::EGrabber`有不同的特点。到目前为止，我们还无法理解`EGrabber<>`中尖括号的含义。`EGrabber`类实际上是一个*模板类*，即由另一种类型参数化的类:

- 在这种情况下，模板参数是要使用的回调模型:即其中之一:`CallbackOnDemand`，`CallbackSingleThread`，`CallbackMultiThread`。
- 空的`<>`用于选择默认模板参数，即`CallbackOnDemand`。
- 可实例化的捕获器类型有：
  - `EGrabber<CallbackOnDemand>`
  - `EGrabber<CallbackSingleThread>`
  - `EGrabber<CallbackMultiThread>`
  - `EGrabber<>` 相当于 `EGrabber<CallbackOnDemand>`
- `EGrabber`头文件，还定义以下类型别名(同义词)：

```
typedef EGrabber<CallbackOnDemand>     EGrabberCallbackOnDemand;
typedef EGrabber<CallbackSingleThread> EGrabberCallbackSingleThread;
typedef EGrabber<CallbackMultiThread>  EGrabberCallbackMultiThread;
```

- .NET泛型与C++模板不太匹配，所以在.NET中，模板`EGrabber`类不存在，我们必须使用其中的一个:`EGrabberCallbackOnDemand`，`EGrabberCallbackSingleThread`，`EGrabberCallbackMultiThread`。

5.6. 事件和回调示例

按需回调

此程序显示捕获器生成的CIC事件的基本信息：

```
#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. 此*using*指令允许写入`Xyz`而不是`Euresys::Xyz`。这有助于保持线路相对较短
2. 定义一个新类`MyGrabber`(从`EGrabber<CallbackOnDemand>`派生)。
3. `MyGrabber`的构造函数通过调用`EGrabber<CallbackOnDemand>`的构造函数初始化其基类。
4. 运行一个`config.js`脚本，该脚本应：
   - 正确配置相机和帧捕获器；
   - 为CIC事件启用通知。
5. 启用`onCicEvent`回调。
6. `onCicEvent`回调函数接收一个`const CicData &`。此结构在`include/EGrabberTypes.h`中定义。它包含一些有关所发生事件的信息。在这里，我们显每个事件的`timestamp`和`numid`。`numid`指示发生的CIC事件。
7. 调用`processEvent<CicData>(1000)`：
   - 捕获器开始等待CIC事件；
   - 如果在`1000`ms内发生事件，捕获器执行`onCicEvent`回调函数；
   - 否则，将引发*超时*异常。

程序输出示例：

```
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)
...
```

单线程和多线程回调

这个程序显示关于捕获器生成的CIC事件的基本信息，这次使用`CallbackSingleThread`模型。

```
#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;
    }
}
```

此程序与`CallbackOnDemand`版本之间的差异非常小：

1. `MyGrabber`从`EGrabber<CallbackSingleThread>`派生而来，而不是`EGrabber<CallbackOnDemand>`。
2. 因此`MyGrabber`的构造函数，通过调用`EGrabber<CallbackSingleThread>`的构造函数，以初始化其基类。
3. `EGrabber`创建一个回调线程(在其中调用`processEvent`)，所以我们不必这样做。这里，我们只需进入一个无限循环。

如您所见，从`CallbackOnDemand`移动到`CallbackSingleThread`是非常简单的。如果您需要`CallbackMultiThread`变量，只需将`MyGrabber`的基类更改为`EGrabber<CallbackMultiThread>`(调用适当的构造函数)。

新缓冲区回调

这个程序演示了如何访问与*新缓冲区*事件相关的信息。它可使用 `CallbackMultiThread`，但也可以使用另一个回调方法。

```
#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>(GenTL::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. 在`onNewBufferEvent`中，创建临时`ScopedBuffer`对象`buf`。`ScopedBuffer`构造函数采用了两个参数：
   - 拥有缓冲区的捕获器：因为我们在一个从`EGrabber`派生的类中，所以我们只是传递`*this`；
   - 有关缓冲区的信息：在`data`中提供。
2. 检索缓冲区的时间戳，该时间戳被定义为*相机开始向帧捕获器发送数据*的时间。
3. 正如在有关事件识别章节中所解释，*新的缓冲区*事件与其他类型的事件略有不同：它们是标准的(根据GenTL)，没有关联的`numid`。
   因此，`NewBufferData`结构(传递给`onNewBufferEvent`)没有`numid`字段。但是，它确实有一个`timestamp`字段，指示*通知驱动程序数据传输到缓冲器已经完成*时间。此*事件时间戳*不可避免地大于步骤2中检索到的*缓冲区时间戳*。
4. 我们到达创建局部变量`buf`的块的末尾。它超出了范围并被销毁：调用`ScopedBuffer`析构函数。这会导致包含在`buf`中的GenTL缓冲区，重新排队(返回)到捕获器的数据流。

程序输出示例

```
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
```

5.7. 相关文件

include/EGrabber.h	主标题。包括除include/RGBConverter.h之外的所有其他标题。定义Euresys::EGrabber，Euresys::Buffer，Euresys::ScopedBuffer
include/EGrabberTypes.h	定义与...相关的数据类型 Euresys::EGrabber
include/EGenTL.h	定义 Euresys::EGenTL
include/GenTL_v1_5.h	标准GenTL标题。定义标准类型、函数和常量
include/GenTL_v1_5_EuresysCustom.h	定义特定Coaxlink常量
include/RGBConverter.h	定义Euresys::RGBConverter帮助程序类

6. Euresys GenApi 脚本

Euresys GenApi脚本语言记录在几个GenApi脚本中。为了方便起见，此处包括了它们。

6.1. doc/BasICS

```
// 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 coaxlink://doc/basics.js'.


// 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';
    }
    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]/;
    }
}

// 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));
    // 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);
}



// Run tests
References();
Operators();
OuterFunction();
Loops();
Exceptions();

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

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

6.2. doc/BufftIsj.js

```
// This file describes the builtins (functions or objects) of Euresys GenApi
// Script. It can be executed by running 'gentl script
// coaxlink://doc/builtins.js'.

// The builtin object 'console' contains a single function, log, which can be
// used to output text to the standard output.
console.log('Hello from ' + module.filename);
console.log('If several arguments are passed,', 'they are joined with spaces');

// 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 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');

// 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"));       // 1e-9
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('floor(3.14) = ' + Math.floor(3.14));
console.log('abs(-1.5) = ' + Math.abs(1.5));
console.log('pow(2, 5) = ' + Math.pow(2, 5));
console.log('log2(2048) = ' + Math.log2(2048));

// 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 second

// 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');
// - coaxlink:// paths (paths relative to the directory where coaxlink scripts
//   are installed)
//   var mod = require(coaxlink://doc/builtins.js);
```

6.3. doc/grabbers.js

```
// This file describes the 'grabbers' object of Euresys GenApi Script. It can
// be executed by running 'gentl script coaxlink://doc/grabbers.js'.

// 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. However, when using
// the 'gentl script' command, 'grabbers' contains the list of all devices.
// This makes it possible to configure several cameras and/or cards.

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

// Each item in 'grabbers' encapsulates all the ports related to one data
// stream:
//  TLPort          | GenTL producer
//  InterfacePort   | Coaxlink 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 f
//  set(f,v)        | set value v to f
//  execute(f)      | execute f
//  features([re])  | get list of features [matching regular expression re]
//  $features([re]) | strict* variant of features([re])
//  ee(f,[re])      | get list of enum entries [matching regular expression 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
//  selectors(f)    | get list of features that act as selectors of f
//  attributes(...) | extract information from the XML file describing the port
//
// * by strict we mean that the returned list contains only nodes/values
//   that are available (as dictated by 'pIsAvailable' GenICam node elements)

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));
        }
    }
}

// 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 brRevision = 0x00000004;
        var standard = port.brRead(brStandard);
        var revision = port.brRead(brRevision);
        if (0xc0a79ae5 === standard) {
            console.log('Bootstrap register "Standard" is OK (0xc0a79ae5)');
        } else {
            console.log('Bootstrap register "Standard" is ' + standard);
        }
        console.log('Bootstrap register "Revision" is ' + revision);
    }
}
```

6.4. DOC/MODEL1.JS

```
// This file describes the special 'module' variable of Euresys GenApi Script.
// It can be executed by running 'gentl script coaxlink://doc/module1.js'. It
// is also dynamically loaded by the coaxlink://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. 适用于MultiCam用户的EGrabber

概念

MultiCam	EGrabber
板	界面
信道	设备+数据流
表面	缓冲区
表面簇(MC_Cluster)	向数据流通告的缓冲区
-	远程设备(相机)
MultiCam参数	GenApi设置/获取功能
-	GenApi命令
CAM文件	Euresys GenApi 脚本
-	CallbackOnDemand
回调函数	CallbackSingleThread
-	CallbackMultiThread

初始化

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

```
//EGrabber
Euresys::EGenTL gentl;
```

信道创建

```
//MultiCam
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) {
    ...
}
```

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

曲面创建(自动)

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

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

曲面创建(手动)

```
//MultiCam
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) {
        ...
    }
}
```

```
//EGrabber
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]));
}
```

表面群集重置

```
//MultiCam
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) {
    ...
}
```

```
//EGrabber
grabber.resetBufferQueue();
```

帧捕获器配置

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

其中H是一个MC_HANDLE(全局MC_CONFIGURATION句柄、板句柄或信道句柄)，并且M指定目标模块(SystemModule，InterfaceModule，DeviceModule或StreamModule)。

相机配置

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

脚本文件

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

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

采集开始/停止

```
//MultiCam
// 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);
```

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

同步(阻塞)缓冲接收

```
//MultiCam
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) {
    ...
}
```

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

```
//EGrabber
{
    // wait for a buffer
    ScopedBuffer buffer(grabber, timeout);
```

```
    // process buffer
    ...
    // ScopedBuffer destructor takes care of making buffer available for new images
}
```

回调

```
//MultiCam
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);
    }
};
```

```
//EGrabber
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
            ...
        }
};
```

同步(阻塞)事件处理

```
//MultiCam
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:
        ...
};
```

```
//EGrabber
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
            ...
        }
};
```

8. .NET程序集

EGrabber 可以通过名为.NET的程序集在.NET语言(C、VB.NET等)中使用 Coaxlink_NetApi.dll

8.1. 第一个例子

此示例创建一个捕获器，并显示其包含的接口、设备和远程设备模块的基本信息。这是第一个C++EGrabber示例的C#版本。

```
using System;
namespace FirstExample {
    class ShowInfo {
        const int CARD_IX = 0;
        const int DEVICE_IX = 0;

        static void showInfo() {
            using (Euresys.EGenTL gentl = new Euresys.EGenTL()) {                          // 1
                using (Euresys.EGrabberCallbackOnDemand grabber =
                       new Euresys.EGrabberCallbackOnDemand(gentl, CARD_IX, DEVICE_IX)) {  // 2
                    String card = grabber.getStringInterfaceModule("InterfaceID");         // 3
                    String dev = grabber.getStringDeviceModule("DeviceID");                // 4
                    long width = grabber.getIntegerRemoteModule("Width");                  // 5
                    long height = grabber.getIntegerRemoteModule("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. 创建一个Euresys.EGenTL对象。这包含以下操作：
   - 定位并动态加载Coaxlink GenTL 生成器(coaxlink.cti)：
   - 检索指向coaxlink.cti导出的函数的指针；
   - 初始化coaxlink.cti。
2. 创建一个Euresys.EGrabberCallbackOnDemand对象。构造函数需要我们刚刚创建的gentl对象。并把要使用的接口和设备的索引，作为可选参数。

3. 使用"GenApi" 于页面6来查找Coaxlink卡的ID。注意`InterfaceModule`后缀(在`getString`**`InterfaceModule`**中)。这指示我们需要来自接口模块的答复。
4. 同样地，找出设备的ID。这一次，我们使用`getString`**`DeviceModule`**瞄准设备模块。
5. 最后，读取相机分辨率。这次，我们使用`getInteger`**`RemoteModule`**，因为必须从相机读取值。
6. `EGrabber` 使用异常来报告错误，并将代码封装在一个`try ... catch`块中。

程序输出示例

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

8.2. C++与.NET EGrabber的区别

*ITALIC*中的术语是占位符：

- *MODULE* 可替换为`InterfaceModule`，`DeviceModule`...
- *EVENT_DATA* 可替换为`NewBufferData`，`CicData`...

Egraber类

C++	.NET
`EGrabber<>`	-
`EGrabber<CallbackOnDemand>`	`EGrabberCallbackOnDemand`
`EGrabber<CallbackSingleThread>`	`EGrabberCallbackSingleThread`
`EGrabber<CallbackMultiThread>`	`EGrabberCallbackMultiThread`

Egraber方法

C++	.NET
`getInfo<`*MODULE*`,`*TYPE*`>(cmd)`	`getInfo`*MODULE*`(cmd, out ...)`
`getInteger<`*MODULE*`>(f)`	`getInteger`*MODULE*`(f)`
`getFloat<`*MODULE*`>(f)`	`getFloat`*MODULE*`(f)`
`getString<`*MODULE*`>(f)`	`getString`*MODULE*`(f)`
`getStringList<`*MODULE*`>(f)`	`getStringList`*MODULE*`(f)`
`setInteger<`*MODULE*`>(f, v)`	`setInteger`*MODULE*`(f, v)`
`setFloat<`*MODULE*`>(f, v)`	`setFloat`*MODULE*`(f, v)`

C++	.NET
setString<*MODULE*>(f, v)	setString*MODULE*(f, v)
execute<*MODULE*>(f)	execute*MODULE*(f)
enableEvent<*EVENT_DATA*>()	enable*EVENT_DATA*Event(f)
disableEvent<*EVENT_DATA*>()	disable*EVENT_DATA*Event(f)

回调

在.NET中，回调定义为委托：

```
grabber.onNewBufferEvent = delegate ...
grabber.onDataStreamEvent = delegate ...
grabber.onCicEvent = delegate ...
grabber.onIoToolboxEvent = delegate ...
grabber.onCxpInterfaceEvent = delegate ...
```

下一节将给出一个完整的示例。

8.3. 单线程回调

该程序显示捕获器生成的CIC事件的基本信息，使用CallbackSingleThread模型。这是C++CallbackSingleThread示例的C++版本：

```
using System;

namespace Callbacks {
    class CallbackExample {
        static void showEvents(Euresys.EGrabberCallbackSingleThread grabber) {
            grabber.runScript("config.js");                                          // 1

            grabber.onCicEvent = delegate(Euresys.EGrabberCallbackSingleThread g,    // 2
                                          Euresys.CicData data) {
                System.Console.WriteLine("timestamp: {0} us, {1}",                   // 3
                                         data.timestamp, data.numid);
            };                                                                       // 4

            grabber.enableCicDataEvent();                                            // 5

            grabber.reallocBuffers(3);                                               // 6
            grabber.start();                                                         // 6
            while (true) {                                                           // 6
            }
        }

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

1. 运行一个config.js脚本，该脚本应：
   - 正确配置相机和帧捕获器；
   - 为CIC事件启用通知。
2. 为*CIC*事件注册回调函数：
   - 创建一个delegate(可被EGrabber调用-在*CIC*事件发生时)；使用两个参数调用此委托：捕获器和包含此事件信息的CicData；
   - 将捕获器的onDataStreamEvent设置为delegate功能。
3. 在回调函数的主体中，只需显示事件的基本信息。
4. 这将结束onCicEvent回调函数的定义。
5. 启用onCicEvent回调。
6. 启动捕获器，进入无限循环。*CIC*事件将在专用线程中通知。

程序输出示例

```
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
```

9. 定义

首字母缩略词

CIC

相机和照明控制器

CTI

公共传输接口

CXP

CoaXPress

EMVA

欧洲机器视觉协会

PFNC

像素格式命名约定

SFNC

标准功能命名约定

术语表

缓冲模块

表示内存缓冲区的GenTL模块。必须向数据流通告缓冲区，以便用图像数据填充它们。

回调模型

定义何时何地(在哪个线程中)执行回调函数。

其中之一：`CallbackOnDemand`，`CallbackSingleThread`，`CallbackMultiThread`。

相机和照明控制器

Coaxlink卡的一部分，用于控制相机及其相关照明设备。

托管在设备模块中。

CoaXPress

高速数字接口标准，允许通过一条或多条Coaxial电缆将数据从设备(如照相机)传输到主机(如计算机内的帧捕获器)。

公共传输接口

GenTL 发生器

数据流模块

处理缓冲区的GenTL模块。

设备模块

包含与相机相关的帧抓取器设置的Gentl模块。

数据流模块的父级。

*远程设备*的同级。

GenApi

处理相机和帧捕获器配置的GenICam标准。

GenApi功能

相机或帧捕获器功能，在寄存器描述中定义。

*设置/获取*参数，或带有副作用的命令。

GenICam

EMVA标准集。它由Genapi、Gentl、SFNC和PFNC组成。

GenTL

处理数据传输的GenICam标准。TL代表传输层。

GenTL发生器

实现GenTL API的软件库。

扩展名为`cti`的文件(例如`coaxlink.cti`)。

信息命令

用于从GenTL模块查询特定信息的数字标识符。信息命令既可以在标准GenTL头文件中定义，也可以在特定于供应商的头文件中定义(例如，在`include/GenTL_v1_5_EuresysCustom.h`中定义了特定于Coaxlink的信息命令)。

接口模块

表示帧捕获器的GenTL模块。

设备模块的父级。

I/O工具箱

Coaxlink卡的一部分，用于控制数字I/O线，并实现诸如速率转换器、延迟线等工具。

托管在接口模块中。

寄存器描述

将低级硬件寄存器映射到相机或帧捕获器功能的XML文件。

远程设备

连接到帧捕获器的照相机。

术语*远程*用于将其与GenTL设备模块区分开来。

系统

代表GenTL发生器的GenTL模块。

也称为TLSystem。

接口模块的父级。

时间戳

事件发生的时间。

对于Coaxlink，时间戳始终是64位整数，并以计算机启动后经过的微秒数表示。