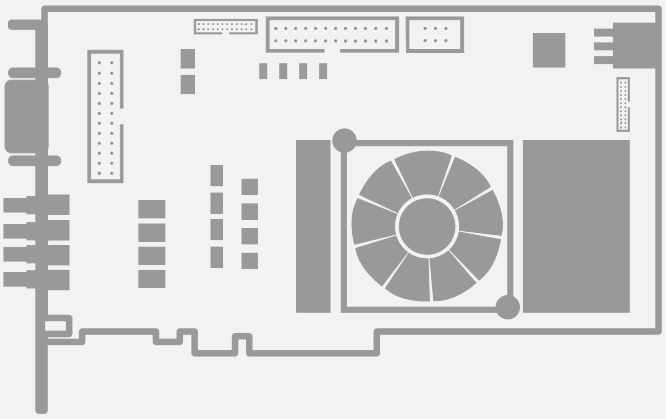
euresys
Empowering Computer Vision

프로그래머 가이드

Coaxlink

Coaxlink 12.2 Programmer Guide



사용 약관

이 문서는 Coaxlink 12.2.1 의 부속 자료입니다(문서 빌드 2100).
www.euresys.com

목차

1. 소개

Coaxlink 카드용 *응용 프로그램 프로그래밍 인터페이스* (API)는 GenICam을 기반으로 합니다.

GenICam의 목표는 서로 다른 물리적 인터페이스(CoaXPress, GigE Vision 등) 또는 다른 공급 업체를 기반으로 카메라 및 프레임 그래버를 사용하기 위한 표준화되고 통일된 프로그래밍 인터페이스를 제공하는 것입니다.

GenICam은 일련의 EMVA 표준 (GenApi 및 GenTL)뿐만 아니라 이름 지정 관련 규칙 (표준 기능의 경우 SFNC, 픽셀 형식의 경우 PFNC)입니다.

- GenApi는 설명입니다. GenApi의 핵심은 *레지스터 설명*의 개념입니다. 레지스터 설명은 XML 파일 형식으로 제공됩니다. 하위 수준 하드웨어 레지스터를 상위 수준 *기능*에 매핑합니다.

  GenApi를 사용하면 응용 프로그램에서 일관된 방식으로 카메라 및 프레임 그래버의 기능을 감지, 구성 및 사용할 수 있습니다.
- GenTL은 데이터 전송에 관한 것입니다. TL 접미사는 *전송 계층*을 나타냅니다.

  GenTL 표준은 카메라 및 프레임 그래버에서 이미지를 열거하고 구성하고 잡는 데 필요한 C 함수 및 데이터 유형 세트를 정의합니다. 이 API는 C 헤더 파일에 의해 정의됩니다.

  프레임 그래버 벤더는이 API를 구현하는 라이브러리 (즉, 표준 헤더 파일에 선언된 함수를 내보내는 라이브러리)를 제공합니다. 이러한 라이브러리는 *GenTL 생성자* 또는 *CTI (Common Transport Interfaces)*라고하며 `cti` file extension. The GenTL producer for Coaxlink cards is `coaxlink.cti`를 사용합니다.

이 문서는 처음부터 끝까지 읽어야 합니다. 각 장과 섹션은 앞의 내용을 토대로 합니다. 텍스트의 일부를 건너 뛰면 일부 설명과 예제가 어설픈 것처럼 보일 수 있습니다. 그런 일이 발생하면 건너뛴 부분을 다시 읽어야 합니다.

2. GenAPI

GenApi는 카메라 구성 문제를 해결합니다. 이것이 달성되는 방식은 일반적이며 프레임 그래버를 포함하여 여러 종류의 장치에 적용됩니다. 이 장에서는 카메라에 대해 말한 모든 내용이 프레임 그래버에도 적용됩니다.

GenApi는 *레지스터 설명*과 *GenApi 구현*이라는 두 가지 기능을 필요로 합니다.

2.1. 레지스터 설명

*레지스터 설명*은 카메라가 컴퓨터에서 읽을 수 있는 데이터 시트라고 생각할 수 있는 XML 파일입니다. 카메라 설정 (예 :`PixelFormat` 및 `TriggerSource`)과 이를 구성하는 방법에 대한 지침을 정의합니다 (예: `ExposureMode`를 `Timed`로 설정하고 `0x12`를 `0xE0140`에 등록). 카메라 문서화를 포함할 수도 있습니다.

2.2. GenApi 구현

*GenApi 구현*은 레지스터 설명 파일을 읽고 해석할 수있는 소프트웨어 모듈입니다.

EMVA는 참조 구현을 제공하지만 사용하기가 매우 어렵고 로깅이 매우 부족합니다. 대신 Coaxlink 소프트웨어 패키지와 함께 번들로 제공되는 Eureys 구현을 사용하는 것이 좋습니다. 또한 이 구현은 강력한 구성 스크립트를 작성할 수 있게 합니다.

2.3. 특징

사용자가 GenApi에서 얻는 것은 카테고리로 구성된 이름 지정된 *기능* 모음입니다.

기능 설정/가져 오기

설정/가져오기 기능은 간단한 설정 (MultiCam의 *매개 변수*라고 함)이며 다른 유형일 수 있습니다:

- 정수 (예: `Width`)
- 부동 (예: `AcquisitionFrameRate`)
- 열거 (예: `PixelFormat`)
- 불린 (예: `LUTEnable`)
- 문자열 (예: `DeviceVendorName`)

기능 값은 *가져오기/설정* 함수를 사용하여 검색/수정할 수 있습니다. 일부 기능은 읽기 전용이며 일부는 쓰기 전용이지만 대부분 읽기/쓰기 액세스가 가능합니다.

명령

또 다른 종류의 기능이 있습니다: *명령* (예: `AcquisitionStart`). 명령은 특별합니다. 관련 값이 없습니다. 그들은 부작용이 있습니다. 명령 기능은 *실행하기* 위한 것입니다. 명령이 실행되면 카메라에서 일부 동작이 발생합니다(예: 소프트웨어 트리거가 생성됨). 분명히 *가져오기/설정* 함수는 명령에 적합하지 않으므로 사용할 수 없습니다.

3. GenTL

GenTL은 부모/자식 관계로 구성된 5가지 유형의 객체를 정의합니다:

1. *시스템* 모듈
2. *인터페이스* 모듈
3. *장치* 모듈
4. *데이터 스트림* 모듈
5. *버퍼* 모듈

각 모듈:

- 시스템의 특정 요소에 해당합니다:
- 질의 할 수있는 관련 정보(*정보 명령*)를 정의합니다 (*정보 얻기* 기능 사용).
- (특정 기능을 사용하여) 해당 모듈의 기능을 수행할 수 있습니다.

또한 버퍼 모듈을 제외한 모든 모듈은 읽기/쓰기 작업을 허용하는 *포트*로 작동합니다. 이러한 포트 기능은 GenApi가 해당 모듈의 설명 파일을 로드하고 GenApi 기능을 사용하는 데 사용됩니다.

3.1. 시스템 모듈

시스템 모듈(*TLSystem*이라고도 함)은 GenTL 제작자 (`coaxlink.cti` 라이브러리)를 나타냅니다. 이 모듈은 상위/하위 트리의 맨 위에 있습니다.

시스템 모듈은 GenTL 생산자에 대한 기본 정보를 제공합니다: `coaxlink.cti`에 대한 전체 경로 및 벤더 이름 (Euresys)과 같습니다.

시스템 모듈의 실제 요점은 시스템에 있는 *인터페이스*(또는 프레임 그래버)를 나열하는 것입니다시스템 모듈의 가장 중요한 기능은 `TLGetNumInterfaces`(시스템의 프레임 그래버 수를 검색하기 위해) 및 `TLOpenInterface`(프레임 그래버 중 하나에 액세스하기 위해)입니다.

3.2. 인터페이스 모듈

GenTL 표준은 프레임 그래버 *인터페이스*를 호출합니다. 시스템 모듈에는 각 프레임 그래버에 대해 하나의 하위 인터페이스가 있습니다: 컴퓨터에 2개의 Coaxlink 카드가 있으면 시스템 모듈에 두 개의 하위 인터페이스가 있습니다.

각 인터페이스는 프레임 그래버를 나타냅니다. 디지털 I/O 라인과 같은 글로벌 프레임 그래버 기능은 인터페이스 모듈에 속합니다. 즉, I/O 라인을 제어하는 GenApi 기능이 인터페이스에 연결되어 있음을 의미합니다.

각 인터페이스는 하나 또는 여러 *장치*의 상위 역할도 합니다. 인터페이스 모듈의 가장 중요한 기능은 `IFGetNumDevices` (인터페이스에 연결할 수 있는 카메라 수를 검색) 및 `IFOpenDevice` (장치 중 하나에 액세스)입니다.

3.3. 장치 모듈

GenTL 표준은 관련이 있지만 서로 다른 두 가지 개념에 대해 *장치* 및 원격 장치라는 용어를 사용합니다. *원격 장치*는 실제로 프레임 그래버에 연결된 실제 카메라입니다. 이것은 여기에 설명된 장치 모듈과 다릅니다.

장치 모듈은 카메라와 관련된 프레임 그래버 설정을 포함하는 모듈입니다. 여기에는 트리거 및 스트로브 같은 것이 포함됩니다.

또한 장치 모듈은 하나의 *데이터 스트림*에 대한 상위 역할을 하며 *원격 장치*의 형제로 간주될 수 있습니다. 장치 모듈의 가장 중요한 기능은 `DevOpenDataStream`(데이터 스트림 중 하나에 액세스하기 위해) 및 `DevGetPort`(원격 장치에 액세스하기 위해)입니다.

3.4. 데이터 스트림 모듈

데이터 스트림 모듈은 *버퍼*를 처리합니다. 수집 실행 중에 카메라에서 프레임 그래버로 이미지가 보내지면 프레임 그래버가 호스트 컴퓨터에 할당된 메모리 버퍼로 전송됩니다. 데이터 스트림 모듈은 이미지 수집이 이루어지는 곳입니다. 대부분의 기능이 있는 곳입니다.

버퍼 처리는 매우 유연합니다. 임의의 수의 버퍼를 사용할 수 있습니다. 버퍼는 입력 대기열에 있거나 출력 대기열에 있거나 일시적으로 대기열에서 제외됩니다. 응용 프로그램은 빈 버퍼가(입력 FIFO로) 대기할 때와 채워진 버퍼가 (출력 FIFO에서) 팝되면 결정합니다.

3.5. 버퍼 모듈

버퍼 모듈은 단순히 부모 데이터 스트림에 주어진 메모리 버퍼를 나타냅니다. 유용한 메타데이터는 버퍼와 연결됩니다. 이것은 이미지 너비, 높이, 픽셀 형식, 타임 스탬프...를 포함하여 이들은 *정보 명령*을 통해 검색됩니다 (표준 GenTL 헤더 파일의 `BUFFER_INFO_CMD_LIST` 참조).

버퍼 모듈은 읽기/쓰기 포트 기능이 없는 유일한 모듈입니다. GenApi 기능이 없습니다.

3.6. GenTL API

GenTL을 사용하면 모든 카메라 및 프레임 그래버 기능을 감지, 제어 및 사용할 수 있지만 사용법은 지루합니다:

- `cti` 파일은 동적으로 로드되어야 하며, 내보내는 함수는 포인터를 통해 액세스되어야 합니다.
- 함수는 응용 프로그램에서 확인해야하는 오류 코드를 반환합니다.
- 형식화되지 않은 버퍼에서 읽거나 쓰는 기능 대부분: 응용 프로그램은 필요한 버퍼 크기를 결정하고, 임시 버퍼를 할당하고,이 버퍼와/에서 데이터를 변환하고 마지막으로 버퍼 메모리를 해제해야 합니다.

GenTL API를 직접 사용하는 대신 다음 중 하나를 사용하는 것이 좋습니다.

- `Euresys::EGenTL` 라이브러리는 이러한 병합을 다루기 때문에 사용자는 할 필요가 없습니다.
- 또는 `Euresys::EGrabber` 라이브러리는 높은 수준의 사용하기 쉬운 인터페이스를 제공합니다.

4. Euresys::EGenTL

Euresys::EGenTL은 표준 GenICam GenTL과 동일한 기능을 제공하지만 보다 사용자 친화적인 인터페이스를 제공하는 C++ 클래스 라이브러리입니다. 예를 들어, 원시 char 포인터 대신 std::string을 사용하고 오류 코드는 예외로 변환됩니다. 또한 Euresys::EGenTL은 GenTL 제작자를 찾아서 내보내는 기능을 로드합니다.

이 라이브러리는 전적으로 C++ 헤더 파일로 구현됩니다. 결과적으로 관련 헤더 파일[1]을 간단히 포함시킬 수 있습니다:

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

GenTL이 정의한 원시, 저수준 C 함수 대신 GenTL 생성자를 나타내는 Euresys::EGenTL 객체를 얻습니다:

- 각 GenTL 기능은Euresys::EGenTL의 멤버 메소드로 사용할 수 있습니다. GenTL 함수 이름은 대문자 접두사로 시작합니다. Euresys::EGenTL에서 메소드 이름은 동일한 접두어로 시작하지만 소문자로 작성됩니다. 예를 들어, GCReadPort함수는 gcReadPort메소드로 표시되고 TLOpenInterface는 tlOpenInterface 메소드로 표시됩니다.
- 모든 GenTL 함수는 성공 또는 실패를 나타내는 GC_ERROR 코드를 반환합니다. 함수가 GC_ERR_SUCCESS 이외의 코드를 반환하면 예외가 발생합니다. 이렇게 하면 각 함수 호출 후에 오류 코드를 수동으로 확인해야 하는 부담을 없앨 수 있습니다.
- ●GenTL 함수는 GC_ERROR를 반환하기 때문에 출력 값은 인수로 전달된 포인터를 통해 반환됩니다. Euresys::EGenTL 메소드는 보다 자연스러운 인터페이스를 제공합니다. 출력 값을 직접 반환합니다:

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

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

  (GC_API는 GC_IMPORT_EXPORT GC_ERROR GC_CALLTYPE으로 정의된다는 것을 참고). 호출 규칙과 DLL 가져 오기/내보내기 특성으로 장식된 GC_ERROR입니다.)
- 텍스트를 처리하는 GenTL 함수의 경우 해당 GC_ERROR 메서드가 char *를 std::string으로 변환하며 그 반대의 경우도 마찬가지입니다:

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

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

- 일부 GenTL 기능은 카메라 또는 프레임 그래버에 대한 정보를 검색합니다. 이 함수는 void * 버퍼에 값을 채우고 INFO_DATATYPE에 실제 값 유형을 표시합니다. Euresys::EGenTL은 C++ 템플릿을 사용하여 이러한 기능을 사용하기 쉽게 만듭니다.

  ```
  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. 첫 번째 예

이 프로그램은 시스템에 있는 Coaxlink 카드를 반복하고 ID를 표시하는 데 `Euresys::EGenTL`을 사용합니다:

```
#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. `Euresys::EGenTL` 클래스의 정의가 들어있는 `EGenTL.h`를 포함하십시오.
2. `Euresys::EGenTL` 객체를 만듭니다. 여기에는 다음 작동이 포함됩니다:
   - Coaxlink GenTL 프로듀서(`coaxlink.cti`)를 찾고 동적으로 로드하십시오.
   - `coaxlink.cti`에서 내보낸 함수에 대한 포인터를 검색하고 `Euresys::EGenTL` 메소드를 통해 사용할 수 있도록 합니다.
   - `coaxlink.cti` 초기화 (이것은 GenTL 초기화 함수 `GCInitLib`를 호출하여 수행됩니다).
3. GenTL 프로듀서를 엽니다. `GenTL::TL_HANDLE` 유형의 핸들을 리턴합니다. `GenTL`네임 스페이스는 1단계에서 `EGenTL.h`에 의해 자동으로 포함된 표준 GenTL 헤더 파일에 정의됩니다.
4. 시스템에 몇 개의 카드가 있는지 확인하십시오.
5. n번째 카드의 ID를 검색하십시오.
6. `Euresys::EGenTL`는 예외를 사용하여 오류를 보고하므로 코드를 `try ... catch` 블록 안에 넣습니다.

프로그램 출력 예:

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

4.2. 관련 파일

`include/EGenTL.h`	기본 헤더. 다른 모든 헤더를 포함합니다. `Euresys::EGenTL`를 정의합니다.
`include/GenTL_v1_5.h`	표준 GenTL 헤더. 표준 유형, 함수 및 상수를 정의합니다.
`include/GenTL_v1_5_EuresysCustom.h`	Coaxlink 관련 상수를 정의합니다

5. Euresys::EGrabber

`Euresys::EGrabber`는 고급 인터페이스를 제공하는 C++ 클래스의 라이브러리입니다. `Euresys::EGenTL` 라이브러리 위에 구축되어 대부분의 사용자에게 권장됩니다.

`Euresys::EGrabber` C++ 클래스 위에 구축된 .NET 어셈블리도 제공됩니다. 이 문서에서는 주로 C++ API에 중점을 둡니다. C++과 .NET 인터페이스 간의 사소한 차이점이 전용 장에 나열되어 있습니다.

여기에 설명된 클래스를 사용하려면 `Euresys::EGrabber` 메인 파일을 포함시켜야 합니다.

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

`Euresys::EGrabber`는 헤더 전용 라이브러리입니다 (`lib` 또는 `dll`파일[2]로 제공되지 않음). 이 클래스는 몇 개의 클래스로 구성되며, 가장 중요한 클래스는 `Euresys::EGrabber`입니다:

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

이 텍스트에서는이 클래스를 *그래버*라고 부릅니다. 그래버는 일련의 관련 GenTL 모듈을 캡슐화합니다:

- 인터페이스: 전역 (공유) 프레임 그래버 설정 및 기능을 나타내는 모듈입니다. 여기에는 디지털 I/O 제어, PCIe 및 펌웨어 상태...가 포함됩니다.
- 장치 (또는 *원격* 장치가 아닌 *로컬* 장치): 프레임 그래버 설정과 카메라 관련 기능이 포함된 모듈입니다. 이것은 주로 카메라 및 조명 제어 기능으로 구성됩니다: 스트로브, 트리거 ...
- 데이터 스트림: 이미지 버퍼를 처리하는 모듈.
- 원격 장치: CoaXPress 카메라.
- 여러 버퍼.

이러한 개념이 명확하지 않은 경우 GenTL 모듈에 대한 장으로 돌아가십시오.

5.1. 첫 번째 예

이 예는 그래버를 생성하고 인터페이스, 장치 및 포함된 원격 장치 모듈에 대한 기본 정보를 표시합니다.

```
#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. `Euresys::EGrabber`클래스의 정의를 포함하는 `EGrabber.h`를 포함하고 필요한 다른 헤더 파일 (예: `EGenTL.h` 및 표준 GenTL 헤더 파일)을 포함합니다.
2. `Euresys::EGenTL` 객체를 만듭니다. 여기에는 다음 작동이 포함됩니다:
   - Coaxlink GenTL 프로듀서(`coaxlink.cti`)를 찾고 동적으로 로드하십시오.
   - `coaxlink.cti`에서 내보낸 함수에 대한 포인터를 검색하고 `Euresys::EGenTL` 메소드를 통해 사용할 수 있도록 합니다.
   - `coaxlink.cti` 초기화 (이것은 GenTL 초기화 함수 `GCInitLib`를 호출하여 수행됩니다).
3. `Euresys::EGrabber` 객체를 만듭니다. 생성자는 2단계에서 만든 `gentl` 객체가 필요합니다. 또한 선택적 인수로 사용할 인터페이스와 장치의 색인을 가져옵니다.

   `EGrabber` 뒤에 오는 꺾쇠 괄호(<>)의 목적은 나중에 분명해질 것입니다. 현재로서는 무시할 수 있습니다.
4. Coaxlink 카드의 ID를 찾는 데 GenApi을 사용하십시오. 우리가 인터페이스 모듈로부터 로부터 답을 원한다는 것을 `Euresys::InterfaceModule`은 나타냅니다.
5. 장치의 ID를 찾으십시오. 이번에는 장치 모듈을 대상으로 `Euresys::DeviceModule`을 사용합니다.
6. 마지막으로 카메라 해상도를 읽습니다. `Euresys::DeviceModule`는 값이 카메라에서 검색되어야함을 나타냅니다.
7. `Euresys::EGrabber`는 예외를 사용하여 오류를 보고하므로 코드를 `try ... catch` 블록 안에 넣습니다.

프로그램 출력 예:

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

5.2. 이미지 수집

이 프로그램은 `Euresys::EGrabber`를 사용하여 Coaxlink 카드에 연결된 카메라에서 이미지를 수집합니다:

```
#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. `Euresys::EGrabber` 객체를 만듭니다. 생성자의 두 번째 및 세 번째 인수는 여기에서 생략됩니다. 그래버는 시스템에 있는 첫 번째 인터페이스의 첫 번째 장치를 사용합니다.
2. 3개의 버퍼를 할당하십시오. 그래버는 필요한 버퍼 크기를 자동으로 결정합니다.
3. 그래버를 시작하십시오. 여기서 우리는 그래버가 10개의 버퍼를 채우도록 요청합니다. 특정 수의 버퍼 이후에 그래버를 멈추게 하지 않으려면, `grabber.start(GenTL::GENTL_INFINITE)` 또는 간단히 `grabber.start()`를 할 수 있습니다.

   그래버를 시작하려면 다음 작업이 필요합니다:

   - `AcquisitionStart` 명령이 카메라에서 실행됩니다.
   - `DSStartAcquisition` 함수가 호출되어 데이터 스트림을 시작합니다.

   이 예에서는 카메라와 프레임 그래버가 올바르게 구성되었다고 가정합니다. 실제 응용 프로그램의 경우 수집을 시작하기 전에 (그리고 해당 문제에 대한 버퍼를 할당하기 전에) 구성 스크립트를 실행하는 것이 더 안전합니다. 이것은 다른 예에서 보여줄 것입니다.
4. 그래버로 가득 찬 버퍼를 기다리십시오. 결과는 `AcquisitionStart`입니다. *범위 지정* 용어는 버퍼의 수명이 현재 범위(즉, 현재 블록)임을 나타 내기 위해 사용됩니다.

5. 버퍼 주소를 얻습니다. 이것은 버퍼의 `getInfo` 메소드를 호출하여 수행됩니다. 이 메소드는 인수로 `BUFFER_INFO_CMD`를 취합니다. 이 경우 표준 GenTL 헤더 파일에 정의된 `BUFFER_INFO_BASE`를 요청합니다.

```
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;
```

`getInfo`는 템플릿 메소드이며, 호출할 때 예상되는 값 유형을 지정해야 합니다. `BUFFER_INFO_BASE`는 포인터를 반환합니다. 이것이 `getInfo<void *>`를 사용하는 이유입니다.

6. 동일한 작업을 수행하여 버퍼의 타임 스탬프를 검색하십시오. 이번에는 `getInfo`의 `uint64_t`버전과 일치하는 `BUFFER_INFO_TIMESTAMP` 버전을 사용합니다.

   참고로, Coaxlink의 경우 타임 스탬프는 항상 64비트이며 컴퓨터를 시작한 후 경과된 마이크로 초 수로 표시됩니다.

7. 우리는 `for` 블록의 끝에 도달합니다. 로컬 변수 `buf`가 범위를 벗어나 파괴됩니다: `ScopedBuffer` 소멸자가 호출됩니다. 이로 인해 `buf`에 포함된 GenTL 버퍼가 그래버의 데이터 스트림에 대기열에 다시 대기하게 됩니다.

프로그램 출력 예:

```
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
```

할당 된 세 개의 버퍼 (`0x7f3c32c54010`의 A, `0x7f3c2c4bf010`의 B 및`0x7f3c2c37e010`의 C)는 라운드 로빈 방식으로 사용됩니다. A → B → C → A → B → C → ... 이것은 다음과 같은 결과입니다:

- 입력 및 출력 버퍼 대기열의 FIFO 특성 :
  - Coaxlink 드라이버는 입력 대기열의 앞쪽에서 버퍼를 팝하고 DMA 전송을 위해 Coaxlink 카드에 제공합니다.
  - 전송이 완료되면 버퍼가 출력 대기열의 뒤쪽으로 푸시됩니다.
- `ScopedBuffer`의 사용:
  - `ScopedBuffer` 생성자는 출력 대기열의 앞쪽에서 버퍼를 팝합니다 (즉, 가장 오래된 버퍼를 사용합니다).
  - `ScopedBuffer` 소멸자는 해당 버퍼를 입력 대기열의 뒤쪽으로 푸시합니다 (따라서 이 버퍼는 이미 입력 대기열에 있는 모든 버퍼 이후에 새로운 전송에 사용됩니다).

5.3. 그래버 구성하기

구성은 모든 이미지 수집 프로그램에서 매우 중요한 부분입니다.

- 카메라와 프레임 그래버는 모두 응용 프로그램 요구 사항에 따라 구성해야 합니다.
- 카메라 구성은 프레임 그래버 구성과 호환되어야 하며 반대의 경우도 마찬가지입니다.

구성은 기본적으로 원격 장치 (즉, 카메라), 인터페이스, 장치 또는 데이터 스트림 모듈과 같은 그래버 모듈에서 수행되는 일련의 설정/가져 오기 작업으로 이어집니다.

이 프로그램은 소위 *RG* 제어 모드 (비동기식 리셋 카메라 제어, 프레임 그래버 제어 노출)를 위해 그래버를 구성합니다.

```
#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>("CycleTargetPeriod", 1e6 / FPS);            // 6
}

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

1. 카메라에서 트리거를 사용합니다.
2. CoaXPress 링크에서 트리거를 찾도록 카메라에 지시합니다.
3. `TriggerWidth` 노출 모드를 사용하도록 카메라를 구성하십시오.
4. 프레임 그래버의 카메라 제어 방법을 `RG`로 설정하십시오. 이 모드에서 카메라 사이클은 프레임 그래버에 의해 시작되고 노출 지속 시간은 프레임 그래버에 의해 제어됩니다.
5. 하드웨어 또는 소프트웨어 트리거를 기다리지 않고 프레임 그래버에게 (`CycleTargetPeriod`에 의해 정의된 속도로) 카메라 사이클을 시작하도록 알려줍니다.
6. 프레임 속도를 구성하십시오.

그러나 그래버를 구성하는 더 좋은 방법이 있습니다. 스크립트 파일을 사용하면 프로그램이 다음과 같이 됩니다.

```
#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;
    }
}
```

구성 스크립트는 다음과 같습니다:

```
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("CycleTargetPeriod",  1e6 / FPS);
```

스크립트 파일을 사용하면 몇 가지 이점이 있습니다:

- 구성은 응용 프로그램을 다시 컴파일하지 않고 변경할 수 있습니다. 이를 통해 개발주기가 단축되고 랩 또는 현장에서 구성을 업데이트할 수 있습니다.
- 구성 스크립트는GenICam 브라우저 및 커맨드 라인 `gentl` 도구로 로드할 수 있습니다. 이렇게 하면 사용자 응용 프로그램 외부의 구성을 확인할 수 있습니다.
- 구성 스크립트는 다른 프로그래밍 언어로 작성된 여러 응용 프로그램에서 쉽게 공유할 수 있습니다: C++, C#, VB.NET...
- Euresys GenApi스크립트의 모든 기능을 사용할 수 있습니다.

5.4. 이벤트

배경

Coaxlink 카드는 다양한 종류의 이벤트를 생성합니다.

- *새로운 버퍼* 이벤트: 버퍼가 데이터 스트림으로 채워졌음을 나타내는 이벤트입니다.
- *데이터 스트림* 이벤트 : 데이터 스트림 및 해당 프레임 저장소와 관련된 이벤트입니다.
- *카메라 및 조명 컨트롤러* 이벤트: 카메라 및 조명 장치의 실시간 제어(장치에서 수행)와 관련된 이벤트입니다.

- *I/O 도구 상자* 이벤트: 디지털 I/O 회선 및 기타 I/O 도구와 관련된 이벤트 (인터페이스에서 오는 이벤트).
- *CoaXPress 인터페이스* 이벤트 : CoaXPress 인터페이스와 관련된 이벤트(인터페이스에서 발생하는 이벤트)입니다.

새로운 버퍼 이벤트는 GenTL에서 표준입니다. 버퍼가 프레임 그래버에 의해 채워지는 경우에 발생합니다. *새로운 버퍼* 이벤트에 첨부된 정보에는 버퍼의 핸들과 타임 스탬프가 포함됩니다.

다른 유형의 이벤트는 Coaxlink로 제한되며 특정 이벤트의 범주로 볼 수 있습니다. 예를 들어, *CIC* 범주의 이벤트에서 우리는 다음을 수행합니다:

- `CameraTriggerRisingEdge` (카메라 트리거 시작)
- `CameraTriggerFallingEdge` (카메라 트리거 끝)
- `StrobeRisingEdge` (라이트 스트로브 시작)
- `StrobeFallingEdge` (라이트 스트로브 끝)
- `AllowNextCycle` (CIC는 다음 카메라 사이클을 위해 준비되었습니다)
- ...

그리고 *I/O 도구 상자* 범주의 이벤트에는 다음이 포함됩니다:

- `LIN1` (라인 입력 툴 1)
- `LIN2` (라인 입력 툴 2)
- `MDV1` (멀티플라이어/디바이더 툴 1)
- ...

카운터

Coaxlink 펌웨어는 각 이벤트 (*새로운 버퍼* 이벤트 제외)의 발생 횟수를 계산하고 이 카운터를 `EventCount`라는 GenApi 기능에서 사용할 수 있도록 합니다. 각 이벤트에는 고유 한 카운터가 있으며 `EventCount` 값은 선택한 이벤트에 따라 다릅니다:

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

또는 *선택한 기능* 표기법을 사용합니다:

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

알림

지금까지 살펴본 것처럼 이벤트가 발생하면 전용 카운터가 증가합니다. Coaxlink는 Euresys :: `Euresys::EGrabber`가 사용자 정의 콜백 함수를 실행하도록함으로써 이 이벤트를 응용 프로그램에 알릴 수도 있습니다. 하지만 먼저 하나 이상의 이벤트에 대한 알림을 사용하도록 설정해야 합니다.

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

또는:

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

구성 스크립트를 사용하면 모든 이벤트에 대한 알림을 쉽게 사용할 수 있습니다:

```
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. `enableAllEvents`라는 헬퍼 함수를 정의하고 모듈(또는 포트) `p`를 인수로 취하십시오.
2. `$ee` 함수를 사용하여 `EventSelector`가 취할 수 있는 값 목록을 검색하십시오. 모듈 `p`에 의해 생성된 이벤트 목록입니다. (`ee`는 *enum 항목*을 의미합니다.)
3. 각 이벤트에 대해 알림을 사용하도록 설정합니다. (`+` 연산자는 문자열을 연결하므로 `e`가 `'LIN1'`인 경우 `'EventNotification[' + e + ']'`식이 `'EventNotification[LIN1]'`으로 평가됩니다.)
4. 인터페이스 모듈에 대해 1단계에서 정의한 `enableAllEvents` 함수를 호출하십시오. 이렇게 하면 *I/O 도구 상자* 및 *CoaXPress 인터페이스* 범주의 모든 이벤트에 대한 알림을 사용할 수 있습니다.
5. 마찬가지로 장치 모듈에서 오는 모든 이벤트(*CIC* 이벤트)에 대한 알림을 활성화합니다.
6. 마지막으로 모든 *데이터 스트림* 이벤트에 대한 알림을 사용하도록 설정합니다.

콜백 함수

이벤트가 발생하고 이벤트 알림이 해당 이벤트에 대해 활성화되면 `Euresys::EGrabber`는 여러 콜백 함수 중 하나를 실행합니다.

이러한 콜백 함수는 재정의된 가상 메소드에서 정의됩니다.

```
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) {
            ...
        }
};
```

보시다시피 이벤트의 각 카테고리에 대해 다른 콜백 함수를 정의할 수 있습니다.

.NET에서 콜백 함수는 가상 메서드를 재정의하는 대신 대리자를 만드는 방법으로 정의됩니다. 예제는 .NET 어셈블리에 대한 장에서 제공됩니다.

이벤트 식별

이벤트가 응용 프로그램에 통보되면 실행되는 콜백 함수는 해당 이벤트의 범주를 나타냅니다. 발생한 실제 이벤트는 numid라는 숫자로 식별되며 include/GenTL_v1_5_EuresysCustom.h에 정의됩니다:

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

참고로 다음 표는 다음과 같은 관계를 보여줍니다:

- 이벤트를 생성하는 모듈
- 이벤트 카테고리
- 콜백 함수의 이름
- 콜백 함수에 전달된 데이터 유형
- 일반적인 `numid`접두사

모듈	카테고리	콜백 함수	데이터 유형	`numid` 접두사
데이터 스트림	새로운 버퍼	`onNewBufferEvent`	`NewBufferData`	-
데이터 스트림	데이터 스트림	`onDataStreamEvent`	`DataStreamData`	`EVENT_DATA_NUMID_DATASTREAM_`
장치	CIC	`onCicEvent`	`CicData`	`EVENT_DATA_NUMID_CIC_`
인터페이스	I/O 도구 상자	`onIoToolboxEvent`	`IoToolboxData`	`EVENT_DATA_NUMID_IO_TOOLBOX_`
인터페이스	CXP 인터페이스	`onCxpInterfaceEvent`	`CxpInterfaceData`	`EVENT_DATA_NUMID_CXP_INTERFACE_`

참고:

- *새로운 버퍼* 이벤트 범주에는 하나의 이벤트만 있으므로 `numid`가 필요하지 않습니다.
- 간단한 명명 체계가 뒤따릅니다: 일부 범주의 이벤트 범주에는 `on`***`SomeCategory`***`Event`라는 콜백 함수가 있으며,이 콜백 함수는 ***`SomeCategory`***`Data` 구조체를 인수로 사용하고 ***`SOME_CATEGORY`***`_`를 일반적인 `numid`접두사로 사용합니다.

예제

우리는 곧 이벤트와 콜백을 보여주는 몇 가지 완전한 예제 프로그램을 보여줄 것입니다.하지만 콜백 함수가 실행되는 상황에 대해 설명하기 전에 설명해야 할 사항이 하나 더 있습니다. 이것은 다음 섹션의 주제입니다.

5.5. EGrabber 선호

언제 콜백 함수를 호출해야 합니까? 어떤 컨텍스트(즉, 어떤 스레드)에서? 이러한 질문에 대해 생각해 보면 여러 *콜백 모델*을 정의할 수 있습니다:

- 응용 프로그램에서 그래버에게 묻습니다: *"버퍼 또는 이벤트가 있습니까? 그렇다면 지금 콜백 기능을 실행하십시오."* 이것은 응용 프로그램 스레드에서 콜백이 요구될 때 콜백이 실행되는 폴링, 동기식 작업 모드입니다.

  이 콜백 모델은 *요청시* 사용할 수 있습니다.
- 응용 프로그램은 그래버에게 단일 전용 스레드를 만들고 이 스레드에서 이벤트를 기다리라는 요청을 합니다. 이벤트가 발생하면 그래버는 해당 콜백 함수를 이 단일 콜백 스레드에서도 실행합니다.

  이 콜백 모델은 *단일 스레드*라고 합니다.
- 응용 프로그램은 그래버에게 각 콜백 함수에 대한 전용 스레드를 생성하도록 요청합니다. 이러한 각 스레드에서 그래버는 특정 이벤트 범주를 기다립니다. 이벤트가 발생하면 해당 콜백 함수가 해당 스레드에서 실행됩니다.

  이 콜백 모델은 *멀티 스레드*라고 부를 것입니다.

이 세 가지 콜백 모델은 모두 의미가 있으며 각 모델은 일부 응용 프로그램에 가장 적합합니다.

- *온 디맨드* 모델이 가장 간단합니다. 구현시 작업자 스레드를 사용할 수 있지만 사용자 관점에서는 스레드를 추가하지 않습니다. 이것은 응용 프로그램이 스레드 동기화, 뮤텍스 등과 같은 것에 대해 걱정할 필요가 없다는 것을 의미합니다.
- 사용자가 전용 콜백 스레드를 원한다면 *단일 스레드*모델이 그를 위해 생성합니다. 스레드가 하나만 있으면 사물이 단순합니다.

  스레드가 하나만 있으면 간단합니다. 이 모델에서 그래버는 (최소한) 두 개의 스레드에서 사용되므로 동기화와 공유 데이터에 대해 걱정할 필요가 있습니다.
- *다중 스레드* 모델에서 이벤트의 각 카테고리는 자체 스레드를 가져옵니다. 이것의 이점은 한 가지 유형의 이벤트(및 콜백 함수의 실행)가 다른 유형의 이벤트에 대한 알림을 지연시키지 않는다는 것입니다. 예를 들어 `onNewBufferEvent`에서 과도한 이미지 처리를 수행하는 스레드는 `onCicEvent` (예: 노출이 완료되었음을 나타내는 이벤트 및 객체 또는 카메라 이동 가능 이벤트)에 의한 CIC 이벤트 알림을 지연시키지 않습니다.

  이 모델에서 그래버는 여러 스레드에서 사용되므로 *단일 스레드* 모델 에서처럼 동기화가 필요합니다.

  물론 응용 프로그램은 이미 수신되어 처리된 *y* 유형의 이벤트 알림보다 오래된 *x* 유형의 이벤트에 대한 통지를 수신할 수도 있음을 알아야 합니다. 결국 이것은 *단일 스레드* 모델과 *다중 스레드* 모델 간의 차별화 요소입니다.

사용자에게 최대한의 유연성을 제공하기 위해 세 가지 콜백 모델을 모두 지원합니다. 이것이 `Euresys::EGrabber`가 다른 플레이버로 존재하는 이유입니다. 지금까지 우리는 `EGrabber<>`에서 꺾쇠 괄호의 의미를 알 수 없었습니다. `EGrabber` 클래스는 실제로 *템플릿 클래스*, 즉 다른 유형에 의해 매개 변수화된 클래스입니다.

- 이 경우 템플릿 매개 변수는 `CallbackOnDemand`, `CallbackSingleThread` 또는 `CallbackMultiThread` 중 하나인 사용할 콜백 모델입니다.
- 빈 <>은 `CallbackOnDemand`라는 기본 템플릿 매개 변수를 선택하는 데 사용됩니다.
- 인스턴스화할 수 있는 그래버 유형은 다음과 같습니다:
  - `EGrabber<CallbackOnDemand>`
  - `EGrabber<CallbackSingleThread>`
  - `EGrabber<CallbackMultiThread>`
  - `EGrabber<>`은 `EGrabber<CallbackOnDemand>` 동일합니다
- 또한 `EGrabber` 헤더 파일은 다음과 같은 타입 별칭(동의어)을 정의합니다:

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

- .NET 제네릭은 C ++ 템플릿과 거의 일치하지 않으므로 .NET에서 템플릿 `EGrabber` 클래스는 존재하지 않으므로 `EGrabberCallbackOnDemand`, `EGrabberCallbackSingleThread` 또는 `EGrabberCallbackMultiThread` 중 하나를 사용해야 합니다.

5.6. 이벤트 및 콜백 예제

주문형 콜백

이 프로그램은 그래버가 생성한 CIC 이벤트에 대한 기본 정보를 표시합니다.

```
#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. 이 *using* 지시어는 `Euresys::Xyz` 대신에 `Xyz` 쓰기를 허용합니다. 이렇게 하면 라인을 비교적 짧게 유지할 수 있습니다.
2. `EGrabber<CallbackOnDemand>`에서 파생된 새로운 클래스 `MyGrabber`를 정의하십시오.
3. `MyGrabber`의 생성자는 `EGrabber<CallbackOnDemand>`의 생성자를 호출하여 기본 클래스를 초기화합니다.
4. 다음을 수행해야 하는 `config.js` 스크립트를 실행하십시오:
   - 카메라와 프레임 그래버를 올바르게 구성하십시오;
   - CIC 이벤트에 대한 알림을 활성화하십시오.
5. `onCicEvent` 콜백을 활성화합니다.
6. `onCicEvent` 콜백 함수는 `const CicData &`를 수신합니다. 이 구조는 `include/EGrabberTypes.h`에서 정의됩니다. 발생한 이벤트에 대한 몇 가지 정보가 들어 있습니다. 여기서는 각 이벤트의 `timestamp` 및 `numid`를 표시합니다. `numid`는 어떤 CIC 이벤트가 발생했는지 나타냅니다.
7. `processEvent<CicData>(1000)`를 호출합니다:
   - 그래버는 CIC 이벤트를 기다리기 시작합니다;
   - 이벤트가 `1000`밀리초 이내에 발생하면 그래버는 `onCicEvent` 콜백 함수를 실행합니다;
   - 그렇지 않은 경우, *타임 아웃* 예외는 버려집니다.

프로그램 출력 예:

```
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)
...
```

단일 스레드 및 다중 스레드 콜백

이 프로그램은 이번에는 `CallbackSingleThread` 모델을 사용하여 그래버가 생성한 CIC 이벤트에 대한 기본 정보를 표시합니다.

```
#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;
    }
}
```

이 프로그램과 `CallbackOnDemand`버전 간에는 거의 차이가 없습니다:

1. `EGrabber<CallbackOnDemand>` 대신에 `EGrabber<CallbackSingleThread>`에서 파생됩니다.
2. 결과적으로 `MyGrabber`의 생성자는 `EGrabber<CallbackSingleThread>`의 생성자를 호출하여 기본 클래스를 초기화합니다.
3. `EGrabber`는 `processEvent`을 호출하는 콜백 스레드를 작성하므로 필요하지 않습니다. 여기서 무한 루프를 입력하기만 하면 됩니다.

보시다시피, `CallbackOnDemand`에서 `CallbackSingleThread`로 이동하는 것은 매우 간단합니다. 대신 `CallbackMultiThread` 변형을 원하면 간단히 기본 클래스를 `MyGrabber`에서 `EGrabber<CallbackMultiThread>`로 변경합니다 (그리고 적절한 생성자를 호출하십시오).

새로운 버퍼 콜백

이 프로그램은 *새로운버퍼 이벤트*와 관련된 정보에 액세스하는 방법을 보여줍니다. `CallbackMultiThread`를 사용하지만 다른 콜백 방법도 사용할 수 있습니다.

```
#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. `onNewBufferEvent`에서 임시 `ScopedBuffer` 객체 `buf`를 만듭니다. `ScopedBuffer` 생성자는 두 개의 인수를 취합니다:
   - 버퍼를 소유한 그래버: 우리가 `EGrabber`에서 파생된 클래스에 있기 때문에 우리는 단순히 `*this`을 전달합니다;
   - 버퍼에 대한 정보: 이것은 `data`에 제공됩니다.
2. *카메라가 프레임 그래버에 데이터를 보내기 시작한* 시간으로 정의된 버퍼의 타임 스탬프를 검색합니다.
3. 이벤트 식별에 대한 섹션에서 설명한 것처럼 *새로운 버퍼* 이벤트는 다른 종류의 이벤트와 약간 다릅니다. 표준(GenTL 기준)이며 관련 `numid`가 없습니다.

   따라서 `onNewBufferEvent`로 전달된 `NewBufferData` 구조에는 `numid` 필드가 없습니다. 그러나 *버퍼에 대한 데이터 전송이 완료되었음을 드라이버에 알린* 시간을 나타내는 `timestamp` 필드가 있습니다이 *이벤트 타임스탬프*는 2단계에서 검색한 *버퍼 타임스탬프*보다 불가피하게 깁니다.

4. 우리는 지역 변수buf가 생성된 블록의 끝 부분에 도달합니다. 범위를 벗어나 파괴됩니다: ScopedBuffer 소멸자가 호출됩니다. 이로 인해 buf에 포함된 GenTL 버퍼가 그래버의 데이터 스트림에 대기열에 다시 대기하게 됩니다.

프로그램 출력 예:

```
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. 관련 파일

include/EGrabber.h	기본 헤더. include/RGBConverter.h외에 다른 모든 헤더를 포함합니다. Euresys::EGrabber, Euresys::Buffer, Euresys::ScopedBuffer를 정의합니다,.
include/EGrabberTypes.h	Euresys::EGrabber와 관련된 데이터 유형을 정의합니다..
include/EGenTL.h	Euresys::EGenTL를 정의합니다.
include/GenTL_v1_5.h	표준 GenTL 헤더. 표준 유형, 함수 및 상수를 정의합니다.
include/GenTL_v1_5_EuresysCustom.h	Coaxlink 관련 상수를 정의합니다
include/RGBConverter.h	Euresys::RGBConverter 헬퍼 클래스를 정의합니다..

6. Euresys GenApi 스크립트

Euresys GenApi 스크립트 언어는 몇 가지 GenApi 스크립트에 설명되어 있습니다. 편의를 위해 여기에 포함됩니다.

6.1. 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 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/builtins.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 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"));        // 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('Math.floor(3.14) = ' + Math.floor(3.14));
console.log('Math.ceil(3.14) = ' + Math.ceil(3.14));
console.log('Math.abs(-1.5) = ' + Math.abs(1.5));
console.log('Math.pow(2, 5) = ' + Math.pow(2, 5));
console.log('Math.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
//  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));
        }
    }
}

// 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/module1.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. MultiCam 사용자를위한 EGrabber

개념

Multicam	EGrabber
보드	인터페이스
채널	장치 + 데이터 스트림
서페이스	버퍼
서페이스 클러스터(MC_Cluster)	데이터 스트림에 발표된 버퍼
-	원격 장치 (카메라)
MultiCam 매개 변수	GenApi 설정/가져오기 기능
-	GenApi 명령
CAM 파일	Euresys GenApi 스크립트
-	CallbackOnDemand
콜백 함수	CallbackSingleThread
-	CallbackMultiThread

초기화

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

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

초기화 채널 생성

```
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);
```

서페이스 생성 (자동)

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

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

서페이스 생성 (수동)

```
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]));
}
```

서페이스 클러스터 재설정

```
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();
```

프레임 그래버 구성

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

여기서 H는 MC_HANDLE (전역 MC_CONFIGURATION 핸들, 보드 핸들 또는 채널 핸들)이고 M은 대상 모듈 (SystemModule, InterfaceModule, DeviceModule 또는 StreamModule)을 지정합니다.

카메라 구성

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

스크립트 파일

```
; 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);
```

수집 시작/정지

```
// 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);
```

동기식(블로킹) 버퍼 수신

```
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
}
```

콜백

```
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
            ...
        }
};
```

동기식 (블로킹) 이벤트 핸들링

```
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
            ...
        }
};
```

8. NET 어셈블리

EGrabber는 Coaxlink_NetApi.dll .NET 어셈블리를 통해 .NET 언어(C #, VB.NET 등)로 사용할 수 있습니다.

8.1. 첫 번째 예

이 예는 그래버를 생성하고 인터페이스, 장치 및 포함된 원격 장치 모듈에 대한 기본 정보를 표시합니다. 이것은 첫 번째 C++ EGrabber 예제의 C# 버전입니다:

```
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. Euresys.EGenTL 객체를 만듭니다. 여기에는 다음 작동이 포함됩니다:
   - Coaxlink GenTL 프로듀서(coaxlink.cti)를 찾고 동적으로 로드하십시오.
   - coaxlink.cti에서 내보낸 함수에 대한 포인터를 검색합니다.
   - coaxlink.cti를 초기화하십시오.

2. `Euresys.EGrabberCallbackOnDemand` 객체를 만듭니다. 생성자에는 방금 만든 `gentl` 객체가 필요합니다. 또한 선택적 인수로 사용할 인터페이스와 장치의 색인을 가져옵니다.
3. Coaxlink 카드의 ID를 찾는 데 GenApi을 사용하십시오. `getString`**인터페이스 모듈**에서 접미사 `InterfaceModule`를 확인하십시오. 이것은 우리가 인터페이스 모듈로부터 답을 원한다는 것을 나타냅니다.
4. 장치의 ID를 찾으십시오. 이번에는 **장치 모듈**을 대상으로 `getString`**장치 모듈** 을 사용합니다.
5. 마지막으로 카메라 해상도를 읽습니다. 이번에는 카메라에서 값을 읽어야 하기 때문에 `getInteger`**리모트모듈**을 사용합니다.
6. `EGrabber`는 예외를 사용하여 오류를 보고하므로 코드를 `try ... catch` 블록 안에 넣습니다.

프로그램 출력 예:

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

8.2. C++과 .NET EGrabber의 차이점

이탤릭 용어는 자리 표시자입니다:

- *모듈*은 , `InterfaceModule`, `DeviceModule`에 의해 대체될 수 있습니다...
- *EVENT_DATA*는 `NewBufferData`, `CicData`에 의해 대체될 수 있습니다...

`EGrabber` 클래스

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

`EGrabber` 방법

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)`

C++	.NET
setInteger<*MODULE*>(f, v)	setInteger*MODULE*(f, v)
setFloat<*MODULE*>(f, v)	setFloat*MODULE*(f, v)
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)

콜백

.NET에서 콜백은 대리자로 정의됩니다.

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

완전한 예가 다음 섹션에 나와 있습니다.

8.3. 단일 스레드 콜백

이 프로그램은 `CallbackSingleThread` 모델을 사용하여 그래버가 생성한 CIC 이벤트에 대한 기본 정보를 표시합니다. 이것은 the C++ `CallbackSingleThread` 예제의 C # 버전입니다:

```
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. 다음을 수행해야 하는 `config.js` 스크립트를 실행하십시오:
   - 카메라와 프레임 그래버를 올바르게 구성하십시오;
   - CIC 이벤트에 대한 알림을 활성화하십시오.
2. *CIC* 이벤트에 대한 콜백 함수를 등록하십시오:
   - *CIC* 이벤트가 발생할 때 `EGrabber`에 의해 호출될 `delegate`을 작성하십시오. 이 델리게이트는 이벤트에 대한 정보를 포함하는 그래버와 `CicData`;
   - 그래버의 `onDataStreamEvent`를 이 `delegate` 함수로 설정하십시오.
3. 콜백 함수의 본문에는 이벤트에 대한 기본 정보만 표시됩니다.
4. `onCicEvent` 콜백 함수의 정의가 끝납니다.
5. `onCicEvent` 콜백을 활성화합니다.
6. 그래버를 시작하고 무한 루프를 입력하십시오. *CIC* 이벤트는 전용 스레드에서 알림을 받을 수 있습니다.

프로그램 출력 예:

```
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. 샘플 프로그램

Coaxlink의 샘플 프로그램은 `coaxlink-<OS>-sample-programs-<MA.MI.RE.BU>.<EXT>`라는 전용 패키지로 제공됩니다. 여기서 `<OS>`는 운영 체제 (`linux`, `macos` 또는 `win`)이고 `<MA.MI.RE.BU>`는 패키지의 버전 번호입니다.

샘플 프로그램	설명	언어	OS
`cpp/egrabber`	`EGrabber`에 대한 코드 스니펫 수집	C++	Windows, Linux, macOS
`cpp/live`	이미지 캡처 및 디스플레이를 보여주는 Win32 응용 프로그램	C++	Windows
`cpp/egrabber-mfc`	이미지 캡처 및 디스플레이를 보여주는 MFC 응용 프로그램	C++	Windows
`cpp/amdDirectGMA`	이미지 캡처, AMD GPU 메모리로 직접 전송 및 디스플레이를 보여주는 OpenGL 응용 프로그램	C++	Windows
`cpp/nvidia/egrabber-cuda`	`EGrabber`로 이미지 캡처 및 CUDA로 처리하는 OpenGL 콘솔 응용 프로그램 (Nvidia GPU에서)	C++	Windows, Linux
`cpp/ffcWizard`	Coaxlink FFC의 계수 계산 방법(플랫 필드 보정)을 보여주는 콘솔 응용 프로그램	C++	Windows, Linux, macOS
`cpp/exif`	*Coaxlink Quad CXP-12 JPEG* 사용 방법 및 EXIF 파일에 메타 데이터 포함 방법을 보여주는 샘플 프로그램 모음	C++	Windows, Linux, macOS
`cs/egrabber`	C#에서 `EGrabber` 및 콜백을 사용하는 방법을 보여주는 콘솔 응용 프로그램	C#	Windows
`cs/grabn`	이미지 캡처를 보여주는 콘솔 응용 프로그램	C#	Windows
`cs/live`	이미지 캡처 및 디스플레이를 보여주는 Windows Forms 응용 프로그램	C#	Windows
`cs/egrabber-wpf`	이미지 캡처 및 디스플레이를 보여주는 WPF 응용 프로그램	C#	Windows

샘플 프로그램	설명	언어	OS
vb/grabn	이미지 캡처를 보여주는 콘솔 응용 프로그램	VB.NET	Windows
vb/live	이미지 캡처 및 디스플레이를 보여주는 Windows Forms 응용 프로그램	VB.NET	Windows

9.1. EGrabber C++ 코드 스니핏

cpp/egrabber는 다음 코드 스니핏을 포함합니다.

스니핏	설명
100-grabn	ScopedBuffer 클래스를 사용하는 간단한 그랩 N 프레임
110-get-string-list	EGrabber 메서드 getStringList의 기본 사용법
120-converter	FormatConverter 속도 측정
130-using-buffer	Buffer 클래스를 사용하는 간단한 그랩 N 프레임
140-genapi-command	GenApi 명령에 대한 쿼리
200-grabn-callbacks	콜백으로 N 프레임을 잡고 DataStream 이벤트 가져오기
210-show-all-grabbers	사용 가능한 그래버 보기
211-show-all-grabbers-ro	사용 가능한 그래버 보기(DEVICE_ACCESS_READONLY로 장치가 열림)
212-create-all-grabbers	사용 가능한 그래버 생성
213-egrabbers	EGrabber와 함께 사용 가능한 그래버 사용
220-get-announced-handles	공지된 버퍼에 대한 정보 및 핸들 가져오기
221-queue-buffer-ranges	다르게 구성된 2개의 버퍼 세트 작성 및 사용
230-script-vars	네이티브 코드와 Euresys 스크립트간에 데이터 전달
231-script-var	네이티브 코드 및 Euresys 스크립트에서 가상 기능 생성 및 사용
240-user-memory	사용자 할당 버퍼로 이동

스니핏	설명
250-using-lut	LUT 프로세서 구성 및 활성화
300-events-mt-cic	EGrabber 다중 스레드 구성에 대한 CIC 이벤트
301-events-st-all	EGrabber 단일 스레드 구성의 모든 이벤트
302-cxp-connector-detection	연결 및 장치 검색과 관련된 CoaXPress 이벤트 표시
500-grabn-cuda-process	N 프레임을 잡고 cuda 연산으로 처리하십시오.
501-all-grabbers-cuda-process	사용 가능한 모든 그래버를 사용하여 N 프레임을 잡고 cuda 작업으로 처리
610-line-scan-array	EGrabber 단일 스레드를 사용한 라인 스캔의 (연속)버퍼 배열
620-multiple-camera	모든 카메라에서 데이터 수집
630-sublink	동일한 PC에서 2개의 서브 링크 그래버에서 버퍼 병합
640-mitsubishi-kd6r807cx	하나의 Mitsubishi KD6R807CX에 연결된 2개의 그래버에서 병합 (memcpy 작업 포함) 버퍼
641-mitsubishi-kd6r807cx	하나의 Mitsubishi KD6R807CX에 연결된 2개의 그래버에서 병합 (DMA 포함) 버퍼
650-multistream	동일한 장치 (Coaxlink Quad G3의 "1-카메라, 4-데이터 스트림" 펌웨어 버전)에서 4개의 데이터 스트림에서 데이터를 수집합니다.
660-phantom	팬텀 스트리머 16 CXP6에 연결된 사용 가능한 그래버에서 버퍼 병합

9.2. EXIF 샘플 프로그램

cpp/exif는 다음 샘플을 포함합니다:

샘플	설명
100-jpeg-exif	4개의 JPEG 인코딩 데이터 스트림에서 데이터를 수집하고 EXIF 파일 생성
200-jpeg-preview-exif	4개의 미리보기 및 4개의 JPEG 인코딩 데이터 스트림에서 데이터를 획득하고 썸네일로 EXIF 파일 생성

10. 정의

10.1. 약어

CIC
: 카메라 및 조명 컨트롤러

CTI
: 공통 전송 인터페이스

CXP
: CoaXPress

EMVA
: 유럽 머신 비젼 협회

PFNC
: 픽셀 형식 명명 규칙

SFNC
: 표준 기능 명명 규칙

10.2. 용어 설명

버퍼 모듈
: 메모리 버퍼를 나타내는 GenTL 모듈. 버퍼는 이미지 데이터로 채울 데이터 스트림에 발표되어야 합니다.

콜백 모델
: 언제 어디서 (어떤 스레드에서) 콜백 함수가 실행되는지 정의합니다.

  `CallbackOnDemand`, `CallbackSingleThread`, `CallbackMultiThread` 중 하나입니다.

카메라 및 조명 컨트롤러
: 카메라 및 관련 조명 장치를 제어하는 Coaxlink 카드의 일부입니다.

  장치 모듈에서 호스팅됩니다.

CoaXPress
: 하나 이상의 동축 케이블을 통해 장치(예: 카메라)에서 호스트(예: 컴퓨터 내부의 프레임 그래버)로 데이터를 전송할 수 있는 고속 디지털 인터페이스 표준.

공통 전송 인터페이스
: GenTL 프로듀서.

데이터 스트림 모듈

버퍼를 처리하는 GenTL 모듈

장치 모듈

GenTL 모듈은 카메라와 관련된 프레임 그래버 설정을 포함하는 모듈입니다.

데이터 스트림 모듈의 부모.

*원격 장치*의 형제.

GenApi

카메라 및 프레임 그래버 구성을 다루는 GenICam 표준.

GenApi 기능

레지스터 설명에 정의된 카메라 또는 프레임 그래버 기능.

설정/가져오기 매개 변수 또는 부작용이 있는 명령.

GenICam

EMVA 표준 세트. GenApi, GenTL, SFNC 및 PFNC로 구성됩니다.

GenTL

데이터 전송을 다루는 GenICam 표준. TL은 전송 계층을 나타냅니다.

GenTL 프로듀서

GenTL API를 구현하는 소프트웨어 라이브러리.

`cti` extension (e.g., `coaxlink.cti` 파일).

정보 명령

GenTL 모듈에서 특정 정보를 쿼리하는 데 사용되는 숫자 식별자입니다. Info 명령은 표준 GenTL 헤더 파일 또는 공급 업체별 헤더 파일 (예: Coaxlink 관련 info 명령은 `include/GenTL_v1_5_EuresysCustom.h`에 정의되어 있음)에서 정의됩니다.

인터페이스 모듈

프레임 그래버에 나타내는 GenTL 모듈.

장치 모듈의 부모.

I/O 도구 상자

디지털 I/O 라인을 제어하고 레이트 컨버터, 지연 라인 등과 같은 툴을 구현하는 Coaxlink 카드의 일부

인터페이스 모듈에서 호스팅됩니다.

레지스터 설명

저수준 하드웨어 레지스터를 카메라 또는 프레임 그래버 기능에 매핑하는 XML 파일.

원격 장치

카메라가 프레임 그래버에 연결되었습니다.

*원격*이라는 용어는 GenTL 장치 모듈과 구별하기 위해 사용됩니다.

시스템

GenTL 프로듀서를 나타내는 GenTL 모듈.

TLSystem이라고도 합니다.

인터페이스 모듈의 부모.

타임스탬프

이벤트가 발생한 시간입니다.

Coaxlink의 경우 타임스탬프는 항상 64비트 정수이며 컴퓨터를 시작한 후 경과된 마이크로 초 수로 표시됩니다.