emva
european machine vision association

GEN<i>CAM

Generic Transport Layer Interface

Rupert Stelz, STEMMER IMAGING GmbH

Group Manager Image Acquisition

Version 1.0

Content

Generic Transport Layer Interface

- Some wording
- The Modules
- Configuration
- Signaling
- The acquisition
- Buffer handling
- Feature Wrap Up

GenTL Overview

emva
european machine vision association

Generic Transport Layer Interface

Provides a technology agnostic API to enumerate and control devices (cameras) and acquire (image)data.

- **C-API**
- **No Device Functionality**
- **Uses GenApi to configure**
- **Interacts closely with GenApi**

GEN<i>CAM

STEMMER® IMAGING

GenTL Wording

emva
european machine vision association

GenTL Producer

A GenTL Producer is the implementation of a GenTL interface in form of a dynamic link library. It provides enumeration, control and image acquisition services.

GenTL Consumer

A GenTL Consumer is a library or application which is able to access / use the interface provided by a GenTL Producer.

GEN<i>CAM

STEMMER® IMAGING

GenTL Modules internal structure

emva
european machine vision association

GenTL Modules

- System — Abstraction of the Host
- Interface — Abstraction of a single interface board
- Device — Abstraction of a single device
- Stream — Abstraction of a data source on a device
- Buffer — Representing the buffer which receives the data

GEN<i>CAM

STEMMER® IMAGING

GenTL Module Enumeration & Instantiation

Open the GenTL Producer

```
HMODULE hDll = LoadLibrary(TLPath.c_str());
if (hDll == NULL)
{
  cerr << "Error loading TL Client: " << TLPath.c_str() << endl;
  return NULL;
}


TL_HANDLE hTl = NULL;
if (Client::TLOpen(&hTl) < 0)
{
  cerr << "Error loading TL\n";
  return hTl;
}
```

Instantiate Module

```
if (TLGetNumInterfaces(hTl, &iNumInterfaces) < 1)
  return NULL;


char szBuffer[1024];
size_t iSize = 1024;
// retrieve name of interface with index 0
status = TLGetInterfaceID(hTl, 0, szBuffer, &iSize);
if (status < 0)
{
  cerr << "Error retrieving interface name\n";
}

// Open th interface
status = TLOpenInterface(hTl, szBuffer, &hInterface);
if (status < 0)
{
  cerr << "Error opening interface name\n";
}
```

GenTL Module Configuration

emva
european machine vision association

Basic Module parameter inquiry through C API.

The C API provides functions in each module to inquire basic settings. This interface does not allow setting any of these parameters.

Advanced Module configuration through GenApi access.

The Module configuration (parameter setting) is done through a GenICam interface. Each module provides a “virtual” register map and a GenICam XML to describe that register map.

GENICAM

STEMMER® IMAGING

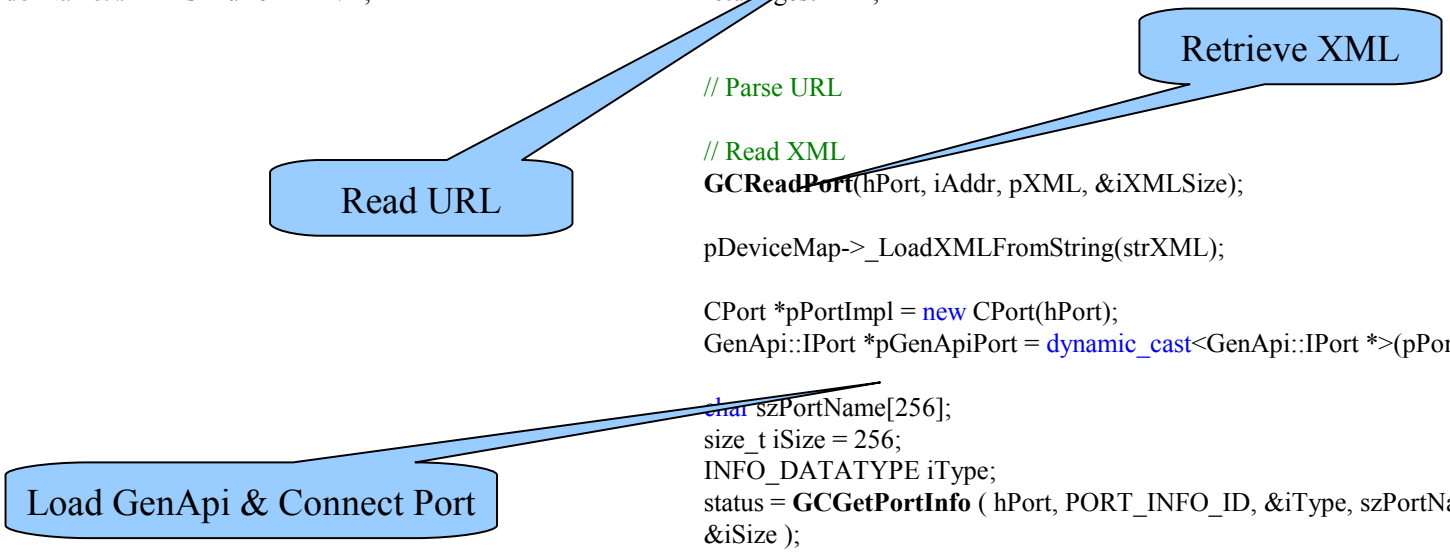
GenTL Module Configuration

Using Info Functions

```
status = TLGetInfo ( hTl, TL_INFO_VENDOR, &iType, szBuffer, &iSize );
if (status >= 0)
{
  cout << " VendorName:\t " << szBuffer << "\n";
}
```

Using GenApi Module access

```
iUrlLength = 2048;
status =  GCGetPortURL( hPort, sURL, &iUrlLength );
if (strlen(sURL) > 2047)
return gcstrXml;

// Parse URL

// Read XML
GCReadPort(hPort, iAddr, pXML, &iXMLSize);

pDeviceMap->_LoadXMLFromString(strXML);

CPort *pPortImpl = new CPort(hPort);
GenApi::IPort *pGenApiPort = dynamic_cast<GenApi::IPort *>(pPortImpl);

char szPortName[256];
size_t iSize = 256;
INFO_DATATYPE iType;
status = GCGetPortInfo ( hPort, PORT_INFO_ID, &iType, szPortName,
&iSize );

gcstring gcstrPortName = "Device"/*szPortName*/;
bool bResult = pDeviceMap->_Connect(pGenApiPort, gcstrPortName);
```



GenTL Signaling

emva
european machine vision association

Each Module provides an Event Signaling Mechanism.

This allows the GenTL Consumer to wait for defined event types from within the thread context of the calling application. Such an event can carry arbitrary data.

For Example after a buffer is filled in the acquisition engine a “NewBuffer” event is signaled to the GenTL Consumer. The GenTL Consumer can now fetch the data associated with the event to know which buffer has been filled and process the data.

GEN<i>CAM

STEMMER® IMAGING

GenTL Acquisition Interface

emva
european machine vision association

Generic Acquisition Interface

The GenTL Producer does not need to interpret the buffer.
Therefor ANY data can be acquired.

But

It can interpret the buffer to
do some preprocessing

GENICAM

STEMMER® IMAGING

GenTL Signaling

```
// Register New Buffer Event
void *NewImageEventData[2];
EVENT_HANDLE pEventNewBuffer = NULL;
status = GCRegisterEvent       ( hDatastream, EVENT_NEW_BUFFER, &pEventNewBuffer);

.....

status = EventGetData(pEventNewBuffer, &NewImageEventData, &iSize, 500);

if ( status  == GenICam::Client::GC_ERR_TIMEOUT)
{
  cout << "Timeout" << endl;
}
else if ( status  < 0)
{
  cout << "Error" << endl;
  ......
}
else
{
  cout << "NewImage: " << NewImageEventData[1] << endl;
  ....
}
```

Register Event

Wait for event

GenTL Acquisition Interface

- Announce Buffer
- Queue Buffer for Acquisition
- StartAcquisition
- Wait for Buffer
  - …
- Queue Buffer for Acquisition
- StopAcquisition
- RevokeBuffer

GenTL Buffer Handling

Allocating , Announcing and Queuing

```
BUFFER_HANDLE pB = NULL;
for (int i = 0; i < 2; i++)
{
        pImageBuffer[i] = malloc(iImageSize);

        status = DSAnnounceBuffer       ( hDatastream, pImageBuffer[i], iImageSize, (void *)i, &pB);
        if (status < 0) { HandleError( "Error in DSAnnounceBuffer: "); return;}

        status = DSQueueBuffer          ( hDatastream, pB);
        if (status < 0) { HandleError( "Error in DSQueueBuffer: "); return;}
}
```

GenTL Start Acquisition

emva
european machine vision association

Starting the Acquisition

```
// Start Acquisition
status = DSStartAcquisition(hDatastream, ACQ_START_FLAGS_DEFAULT, INFINITE);
if (status < 0) { HandleError( "DSStartAcquisition failed: " ); return;}

CCommandPtr ptrStartAcq= pDeviceMap->_GetNode("AcquisitionStart");
(*ptrStartAcq).Execute();
```

GEN<i>CAM

STEMMER® IMAGING

GenTL Acquisition Loop

```
while (bRun)
{
  size_t iSize = sizeof(NewImageEventData);
  status = EventGetData(pEventNewBuffer, &NewImageEventData, &iSize, 500);

  if ( status  == GenICam::Client::GC_ERR_TIMEOUT)
  {
        // Timeout
  }
  else if ( status  < 0)
  {
        // Error
  }
  else
  {
    // Process Image Data
    status = DSQueueBuffer          ( hDatastream, NewImageEventData[0]);
  }
}
```

GenTL Acquisition Shutdown

emva
european machine vision association

```
// Stop Acquisition
status = DSStopAcquisition(hDatastream, ACQ_STOP_FLAGS_DEFAULT);
if (status < 0) { HandleError( "DSStopAcquisition failed: " ); return;}

// Stop Remote Device
CCommandPtr ptrStopAcq= pDeviceMap->_GetNode("AcquisitionStop");
(*ptrStopAcq).Execute();

// Cleanup
status = DSFlushQueue          ( hDatastream,
ACQ_QUEUE_INPUT_TO_OUTPUT);
if (status < 0) { HandleError( "DSFlushQueue failed: " ); return;}

status = DSFlushQueue          ( hDatastream, ACQ_QUEUE_OUTPUT_DISCARD);
if (status < 0) { HandleError( "DSFlushQueue failed: " ); return;}
```

GEN<i>CAM

STEMMER® IMAGING

GenTL Compliance

- GenICam Trac
  - Discussions
  - Bugs
- Test Framework
  - In SVN
- Simple Demo Implementation
- Standard Text
  - V 1.1, RC for 1.2 is out
- Plugfest

GenTL Features

emva
european machine vision association

- Technology Agnostic
- Any number of devices
- Any number of data streams per device
- Data streams of any data type
- Using GenTL Consumer Thread environment
- Allows multithreaded processing
- Flexible Cofiguration Mechanism

GEN<i>CAM

STEMMER® IMAGING

Thank you for your attention!

Contact me ➔ r.stelz@stemmer-imaging.de

Get information ➔ www.genicam.org