Groget

a tour focused on camera vendors

introduction to

GEN<i>CAM Standards

About the Presentation

Latest version of the presentation available from https://www.groget.org/publications/genicam_introduction.pdf.

Presentation Overview

Before We Start

Please stop the presentation whenever something is unclear or further details are required

We should pause and demonstrate the topics on a real examples or with hands-on experience wherever useful

1

Introduction to GenICam

History and goals

Brief History

Started in 2004 as a fork from GigE Vision group with intention to define flexible and technology independent way to access camera features

First release in 2006 in ± sync with GigE Vision, started with GenApi and SFNC modules

GenTL added to standardize more than just feature control and spread GenICam to other technologies

Being well accepted through the GigE Vision use case, GenICam is reused for all new machine vision standards as well as independently

Goals of GenICam

Provide common framework for communication between (machine vision) devices and applications.

Covering more and more device interface technologies, delivering similar (thus familiar) user experience with wide variety of products.

Generic interfaces providing wide flexibility for product design, but with enough space for specific vendor features.

Easy integration of the products in the target system.

2

SFNC GenICam Module

Names and hierarchy of the (camera) features

Evolution of SFNC

Started before finishing the initial release of GenICam and GigE Vision standards

- To ensure at least the essential interoperability through minimal set of mandatory features
- To prevent unnecessary diversity in naming same things always different way

Over years extended to 500+ page document as standardizing more feature groups was considered useful by numerous contributors

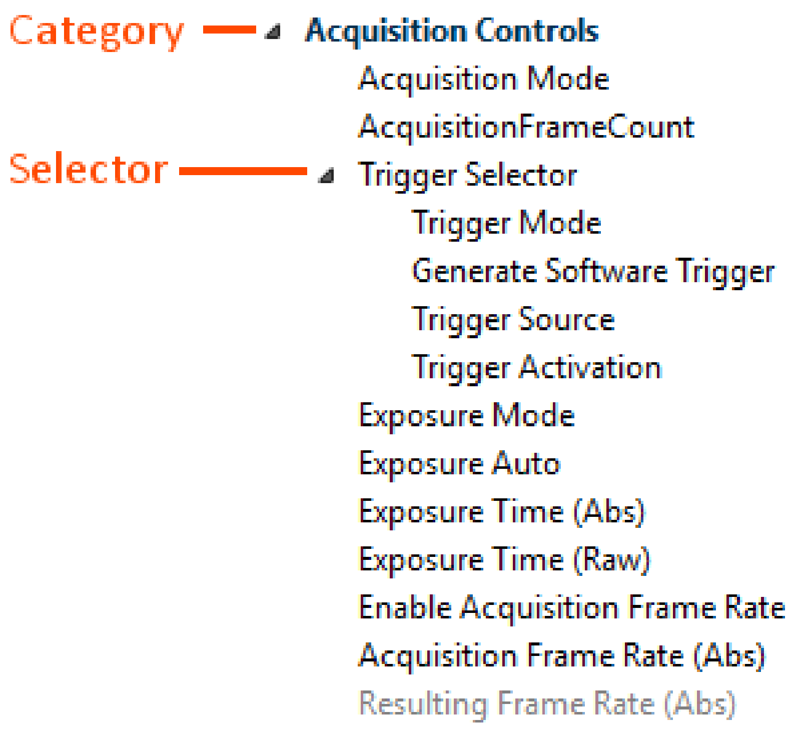
Features Hierarchy

Features organized in **categories**

- Category contains related features, controlling common subtopic of device functionality
- GUI only meaning

Repeated features ("arrays") implemented using **selectors**

- GenICam approach to eliminate unneeded duplicities



Acquisition Controls
Acquisition Mode
AcquisitionFrameCount
Trigger Selector
Trigger Mode
Generate Software Trigger
Trigger Source
Trigger Activation
Exposure Mode
Exposure Auto
Exposure Time (Abs)
Exposure Time (Raw)
Enable Acquisition Frame Rate
Acquisition Frame Rate (Abs)
Resulting Frame Rate (Abs)

Features Behaviour

Feature properties defined by SFNC include

Data type: integer, float, string, enumeration, command, boolean, register

Visibility: beginner, expert, guru, invisible

Access mode: R/W, RO, WO (GenICam also recognizes NI/NA)

Standard values: list or range of expected feature values

Unit: expected standard unit to be used for the feature

Feature Groups

Categories defined in SFNC 2.3

- Device info

- Image format
- Scan 3D

- Analog
- LUT
- Color Transform.

- Acquisition
- Transfer

- Digital I/O
- Counter and Timer
- Encoder
- Logic Block
- Action Commands
- Software Signal
- Lighting Control

- Chunk Data
- Event

- Source
- User Set
- Sequencer

- File Access

- GenICam
- Transport Layer

- Test

Feature Groups (cont.)

Topics being discussed for future SFNC versions:

- Complex pixel format description

Functional Model: Fixed vs. Flexible

Initially a fixed functional model was considered for SFNC

- Did not work, too hard to standardize so that everybody is happy
- Could be advantage for generic applications but limiting for flexible device design
- In many cases the SFNC feature description is loose, giving space for interpretation
- Generic applications need to be careful about making too strong assumptions about the model

Where Fixed Model is Important

Mandatory features

- Features shared as mandatory by most technologies, essential for elementary device control
- Width, Height, PixelFormat, PayloadSize, AcquisitionMode, AcquisitionStart, AcquisitionStop

3D support (refer to dedicated presentation)

- Features required for 3D interoperability are too complex to be embedded in a usual transfer protocol (GEV/U3V)
- The missing protocol support needs to be compensated by reliable SFNC based info exchange (incl. chunk data)

Other feature groups

- File access, action commands, source/transfer control, FW update
- LinePitch or similar features breaking stream self-description

Special Purpose Features

Root: top level category used to define set of published feature nodes

Device: port used to connect to the remote device port in GenApi

PayloadSize: important to allocate acquisition buffers (discuss also the variable payload size option and GenDCFlowMappingTable)

TLParamsLocked: used to protect sensitive transport layer related features during acquisition, discussions about its exact use still running

SourceSelector: special selector covering features from multiple categories, used to implement multi-source devices

Transport Layer Control Features

Defined by SFNC as well but most of them have limited use

- Given functionality typically accessed directly through the low level control protocol
- Danger of clash with the protocol engine
- Caching problems (if given feature is modified under the hood)

Some libraries/applications might hide even some typically public features and wrap them within own functionality

- AcquisitionStart, AcquisitionStop

Special Feature Groups

Chunk features

- Purpose
- Naming: ChunkXYZ where XYZ is the base feature
- Individual chunks need to be switched on as well as the chunk mode itself

Event features

- Purpose
- Naming for XYZ base event:
  - EventXYZData is category holding all data related to the event
  - EventXYZ is a feature to be used to register notification
  - Recommended EventXYZTimestamp, EventXYZFrameID
- Individual events need to be switched on

PFNC - Sub-module of SFNC

PFNC - generic convention for naming pixel formats

- Shared among technologies
- Now part of SFNC
- Provides also standard ID's for the most usual formats to provide interoperability also at protocol level
- Easy and straightforward mechanisms to standardize new formats
- Includes usual formats used by 2D and 3D devices
- More complex formats like polarized or hyperspectral under discussion (2019)

PFNC Formats Overview

Each PFNC format defines following properties of a pixel:

- Number, order and type of pixel components
- Pixel data type (unsigned, signed, float)
- Number of bits used by each component
- Bit packing, grouping, clustering or alignment style
- Planar formats

Examples:

- Mono8, Mono10p, Mono8s
- RGB12, BGR565p, RGB10p32, YUV422_8_UYVY
- BayerGR10, SCF1WBWG8, BiColorRGBG10p
- Coord3D_ABC32f_Planar, Coord3D_AC8, Confidence1p
- (GigE Vision uses also few legacy non-PFNC formats)

PFNC Advanced Topics

Camera interface specific options:

- Line vs. image padding conventions for non-byte-aligned pixels
- Default lsb vs. msb versions of pixel formats

Beware and distinguish between:

- Pixel format "endianness" - are the pixel data bits shifted towards the most or least significant **bit**? (CXP uses msb on the wire)
- Pixel data endianness - are the multi-**byte** pixel data stored in memory as LE or BE? (GEV 1.x allows to switch to BE)

Interoperability:

- Not all formats widely supported, same for above options
- If possible, prefer image padding, lsb, little endian pixels

Implementing SFNC

Reference SFNC compliant XML

- Generated automatically for each SFNC version and published on genicam.org
- Includes complete PFNC support
- Covers also GenTL SFNC
- Generated using tools available from SVN
- Includes all SFNC features with correct type, visibility, access modes and also default info texts
- Covers categories and selectors
- Includes even hints about feature limits or expected units
- Provides outputs specific for individual GenApi schema versions
- Can be easily used to start device XML development quickly

Implementing SFNC - PFNC

Reference PFNC header file

- Generated automatically for each update of the PFNC pixel format value list and published on genicam.org
- Includes symbolic names for all official PFNC format ID's

Validating SFNC

Schematron files available for all SFNC releases

- Generated by the same tools as reference XML
- Available for XML's based on any GenApi schema
- Can be used to validate within editors supporting Schematron (e.g. oXygen XML) or with other Schematron validators
- Outputs errors and optionally also warnings
- The checks include
  - Correct feature type
  - Standard namespace (incl. standard enumeration entries)
  - Correct visibility and category sorting
  - Warns about deprecated features

3

GenApi GenICam Module

How the features work under the hood

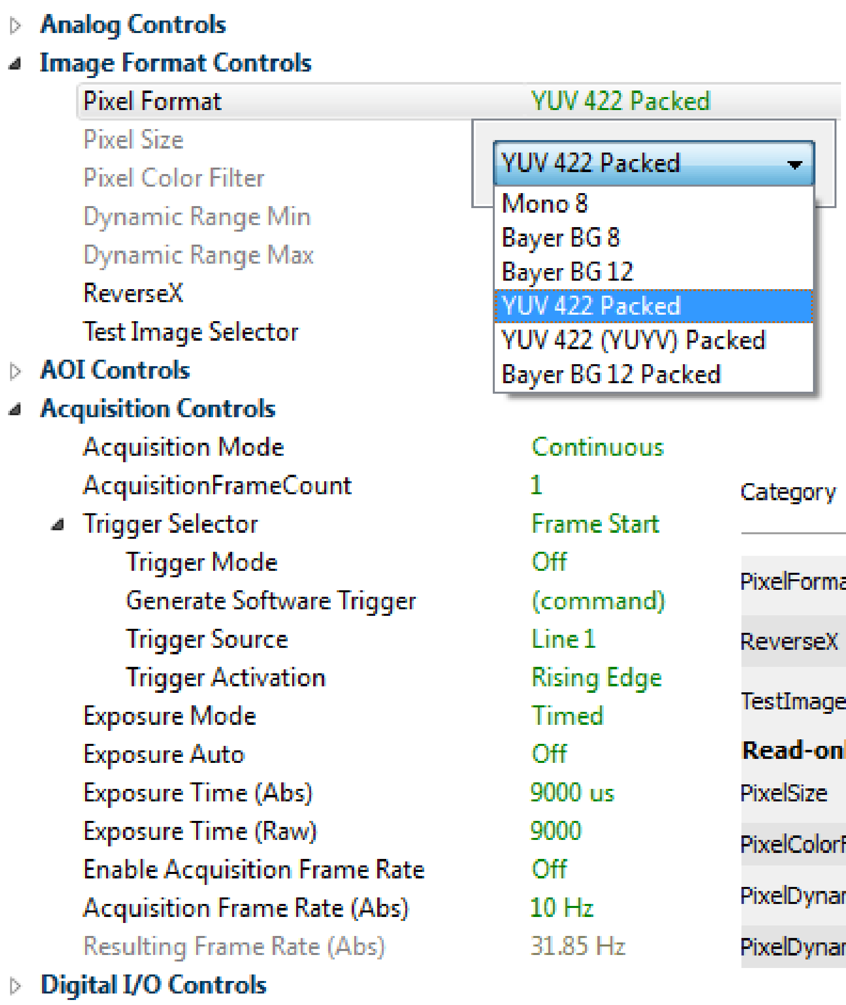
GenApi - GenICam Basic Building Block

Core of GenICam

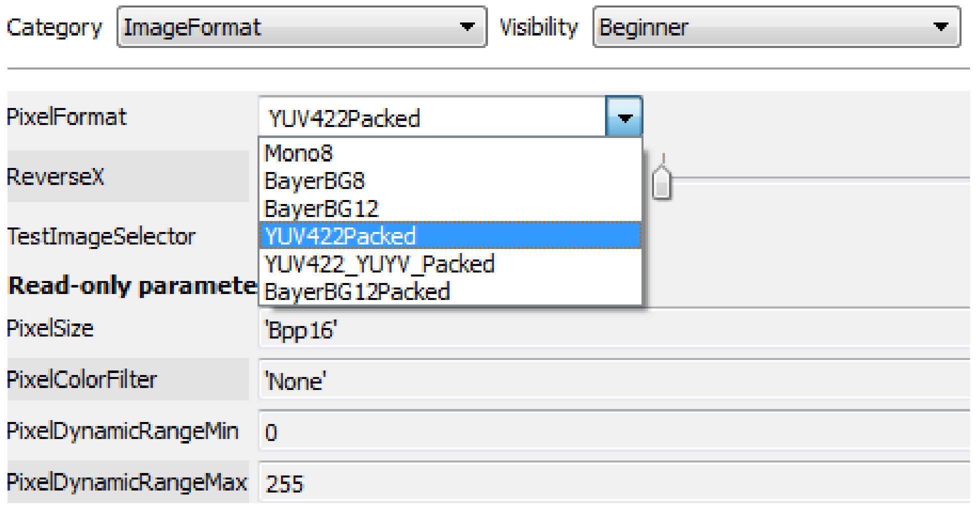
Some highlights

- Generic access to cameras
- Flexible (device specific) register access
- Full list of feature interfaces, flexible relationships
- Transport layer agnostic
- Based on well documented XML specification
- Production quality reference implementation

User Experience



Analog Controls	
Image Format Controls	
Pixel Format	YUV 422 Packed
Pixel Size	
Pixel Color Filter	
Dynamic Range Min	
Dynamic Range Max	
ReverseX	
Test Image Selector	
AOI Controls	
Acquisition Controls	
Acquisition Mode	Continuous
AcquisitionFrameCount	1
Trigger Selector	Frame Start
Trigger Mode	Off
Generate Software Trigger	(command)
Trigger Source	Line 1
Trigger Activation	Rising Edge
Exposure Mode	Timed
Exposure Auto	Off
Exposure Time (Abs)	9000 us
Exposure Time (Raw)	9000
Enable Acquisition Frame Rate	Off
Acquisition Frame Rate (Abs)	10 Hz
Resulting Frame Rate (Abs)	31.85 Hz
Digital I/O Controls	



Category: ImageFormat — Visibility: Beginner

PixelFormat	YUV422Packed
ReverseX	
TestImageSelector	
Read-only paramete	
PixelSize	'Bpp16'
PixelColorFilter	'None'
PixelDynamicRangeMin	0
PixelDynamicRangeMax	255

XML Format

```xml
<Integer Name="Width" NameSpace="Standard">
  <ToolTip>Image width provided by the device</ToolTip>
  <Description>Image width provided by the device in pixels.</Description>
  <DisplayName>Width</DisplayName>
  <Visibility>Beginner</Visibility>
  <pIsLocked>AOISize_LockCalc</pIsLocked>
  <Streamable>Yes</Streamable>
  <pValue>Width_Val</pValue>
  <pMin>Width_MinCalc</pMin>
  <pMax>Width_MaxCalc</pMax>
  <pInc>Width_IncCalc</pInc>
  <Unit>px</Unit>
  <Representation>Linear</Representation>
</Integer>

<IntSwissKnife Name="Width_MinCalc" NameSpace="Custom">
  <Visibility>Invisible</Visibility>
  <pVariable Name="MINPRESENT">AOIMin_Inq</pVariable>
  <pVariable Name="MIN">Width_Min</pVariable>
  <pVariable Name="AOIPRESENT">AOI_Inq</pVariable>
  <pVariable Name="MAXWIDTH">WidthMax_Val</pVariable>
  <Formula><![CDATA[ (MINPRESENT = 1) ? MIN : ((AOIPRESENT = 1) ? 1 : MAXWIDTH) ]]></Formula>
</IntSwissKnife>

<IntReg Name="Width_Val" NameSpace="Custom">
  <Visibility>Invisible</Visibility>
  <Address>0xA0008</Address>
  <Length>4</Length>
  <AccessMode>RW</AccessMode>
  <pPort>Device</pPort>
  <pInvalidator>UserSetLoad_Val</pInvalidator>
  <pInvalidator>WidthMax_Val</pInvalidator>
  <Sign>Unsigned</Sign>
  <Endianess>BigEndian</Endianess>
</IntReg>
```

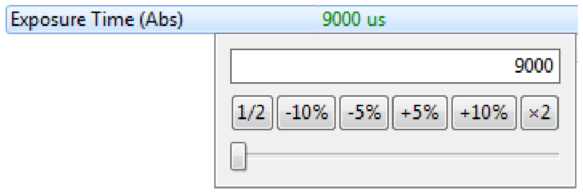
XML Features

Basic properties of a GenApi feature

- **Name** (used to access the feature programmatically)
- **Namespace** (standard, custom, both), possible approaches for custom feature naming (prefix?)
- **Data type** (integer, float, string, ...)
- **Actual meaning** (abstract, for standard features defined in SFNC)
- Programming feature access helpers (access mode, limits, ...)
- GUI helpers (display name, info texts, ...)

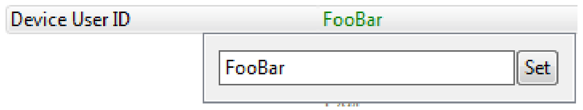
Feature Interface Types

Integer: incl. limits, increment or even value set (slider)

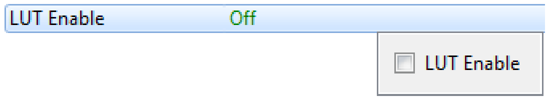


Float: possibly with hints for display such as precision/notation

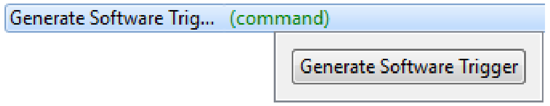
String: ASCII/UTF8 (edit box)



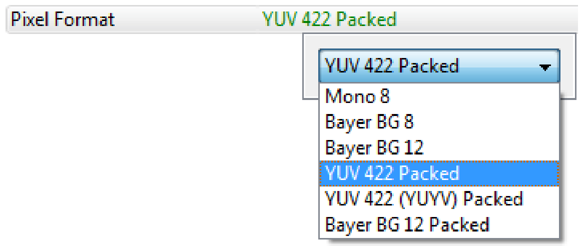
Boolean: true/false (checkbox)



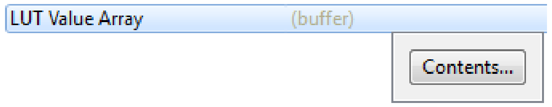
Command: triggers action on the device (button), completion feedback



Enumeration: selection from set of values (combo box)



Register: blob of memory



XML File Structure

Typical GenApi XML file consist of

- **Feature nodes** (those visible to users in GUI, accessible from Root)
- **Implementation nodes** (registers, computation helpers)
- **Auxiliary nodes** (structure, ports, etc.)

Any of the nodes can be accessed programmatically

If needed (embedded) this layering approach can be simplified

Feature Nodes

Accessible from Root category

Feature nodes allow to add documentation and GUI hits

- info texts (plus DocuURL, pError)
- limits, increments (plus value set)
- unit
- representation (linear, logarithmic, pure number)
- display notation/precision (fixed, scientific)
- visibility control

Also the access mode can be further limited

- ImposedAccessMode, pIsImplemented/Available/Locked
- Helps to avoid problem with dependencies if low level nodes would have limited access mode

Computation/Conversion Nodes

Integer or float "swiss knives" and converters

- Value conversion between raw and interface units
- Complex calculations
- Access mode computation and limiting
- Register address computation (incl. selector handling)
- Register/value selection based on register availability

Additional helpers

- Replicator (pValueCopy)
- Multiplexer (pValueDefault/pValueIndexed)

Register Nodes

Describe the register space of the underlying medium

- Device read/write, chunk data, event data
- The medium itself described through port

Different register types

- IntReg, FloatReg, StringReg, Register, MaskedIntReg, StructReg

Important register properties

- Address (incl. pAddress, pIndex[Offset], IntSwissKnife), length
- Sign, Endianess (refer to endianess problems in GEV)
- Caching, polling time, invalidators
- Connection to port

(some properties might overlap between feature nodes and registers)

Auxiliary Nodes

Category: feature structuring for GUI

Port: connection to the register space

- Port name used to connect ("Device", GenTL ports)
- Chunk and event ports (ChunkID, EventID)

Group: only internal XML structuring helper

Selector Nodes

Selector is GenApi's idea of array (or other iterable container)

- In XML applied through pSelected
- Can be used with Integer, Enumeration and Boolean
- Can be cascaded (multidimensional array)
- By convention acts only as array index, no other side effects
- In GUI typically handled similar as categories

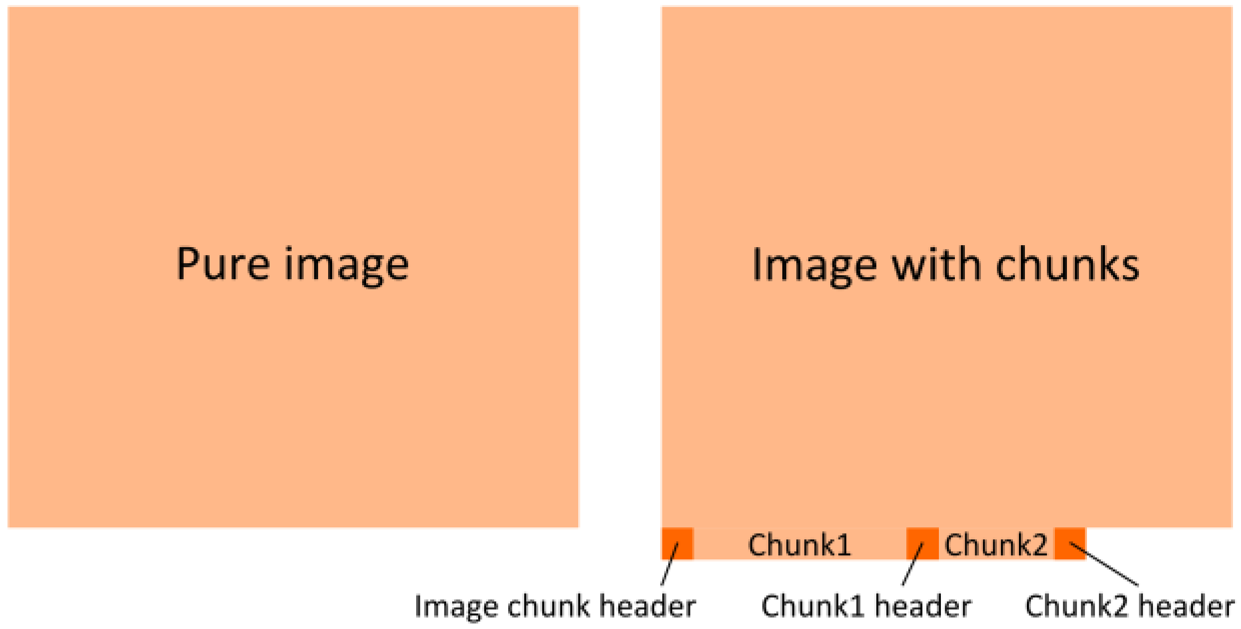
The indexing can be performed in

- The device (selector value written to a register)
- XML (selector value used in selected register address calculation)

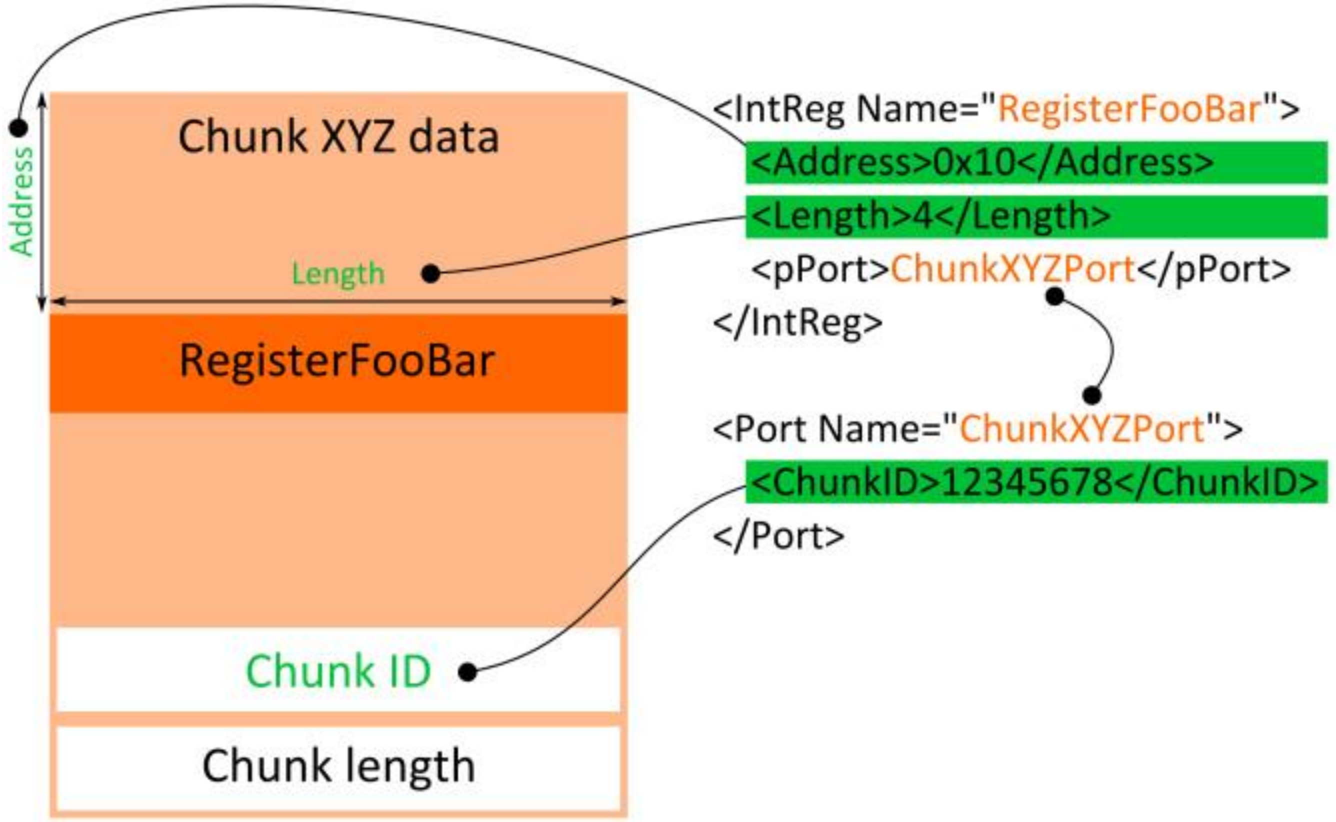
Examples:

- TriggerSource[TriggerSelector] - enumerated selector
- DeviceID[DeviceSelector] - integer selector

Chunk Data


Pure image
Image with chunks
Chunk1
Chunk2
Image chunk header
Chunk1 header
Chunk2 header

Chunk Data (cont.)



```
<IntReg Name="RegisterFooBar">
  <Address>0x10</Address>
  <Length>4</Length>
  <pPort>ChunkXYZPort</pPort>
</IntReg>

<Port Name="ChunkXYZPort">
  <ChunkID>12345678</ChunkID>
</Port>
```

Chunk Data (cont.)

Chunk data have fixed relation to the buffer (not affected by device nodemap changes)

Preferably the chunk data do not depend on Device port

- Planned possibility to extract chunk sub-nodemap

Chunk features/registers are just like regular features/registers

- Chunk registers can implement selectors/arrays
- Chunk-depending features can use any computations
- Chunk registers can carry access mode inquiry bits
- etc.

GenDC

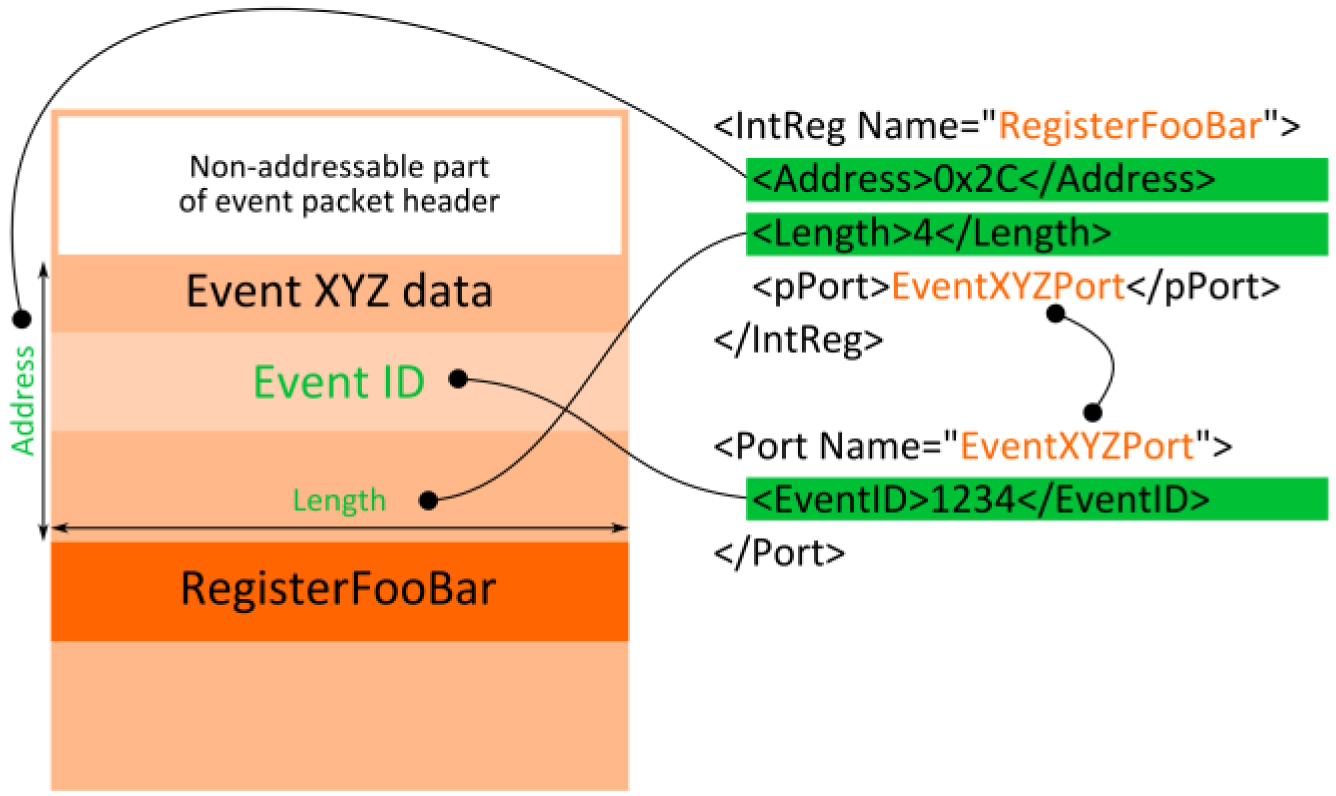
- Possibly multiple chunk sections in one buffer

Chunk Data (cont.)

GenApi uses chunk adapters to apply chunk data from a buffer to the nodemap. The adapter:

1. Parses chunk structure in the buffer.
2. For each chunk checks if XML has port with matching chunk ID.
3. If found, it attaches the chunk's buffer to that port.
4. Invalidates all nodemap features depending on that port to force them read new values from the attached buffer.

Asynchronous Events



```
<IntReg Name="RegisterFooBar">
  <Address>0x2C</Address>
  <Length>4</Length>
  <pPort>EventXYZPort</pPort>
</IntReg>

<Port Name="EventXYZPort">
  <EventID>1234</EventID>
</Port>
```

Asynchronous Events (cont.)

Event data handling is in many ways similar to chunk data handling with some specifics:

- For each transport technology it's important to know where in the packet start by convention the addressable data
- Application should read event data within the notification callback to avoid feature overriding by next event
- Application should process the data within the callback quickly to avoid blocking the event handling engine

Preferably the event data do not depend on Device port

Event features/registers are just like regular features/registers

- Event registers can implement selectors/arrays
- Event-depending features can use any computations
- Event registers can carry access mode inquiry bits
- etc.

Asynchronous Events (cont.)

GenApi uses event adapters to apply event data from a "packet" to the nodemap. The adapter:

1. Parses event structure in the packet (typically just single event).
2. Finds event ID, checks if XML has port with matching event ID.
3. If found, it attaches the event data from the packet to that port.
4. Invalidates all nodemap features depending on that port to force them read new values from the attached buffer (and deliver notifications to the application).

Dependencies & Notifications

GenApi reference implementation allows to register for callbacks notifying about changes of individual features:

- When new value is written to the node or a node it depends on
- Upon change of a <pInvalidator> node
- As a result of polling thread activity for polling nodes (registers, commands, self-clearing enum entries)
- When a new buffer is attached (for chunk features)
- When a new event is delivered (for event features)
- For features depending on command also when the command completes

Beware of cycle dependencies (AOI...)

Feature Persistence

Current camera status can be persistent on the device

- User set features
- Device knows all dependencies and required persistence order, can even survive FW-update
- The stored status is bound to the device, but cannot be re-applied to another device

... or by means of software with help of GenApi persistence

- Streamable features (optionally including contents of user sets and sequencer sets)
- Was subject to discussions and fixing, might be hard to implement right for devices (temporarily NA nodes etc.), but improving
- Can be applied to set of same devices
- Maybe publish single hidden "streamable" register

More on Caching

Register caching modes

- WriteThrough, WriteAround, NoCache
- Invalidation, polling
- "ValueCache" - register space caching → beware of overlapping registers that would share the value cache

Autogain-like features

- Self-clearing enum entries, pBlockPolling
- Implementation relies on polling, best refer to EnumerationTestSuite::TestAutoGain()
- Many SW libs might ignore polling anyway

More on Endianness

GenApi assumes the data are read/written to/from camera always in the native byte order

- Conversion handled within GenApi based on <Endianess>
- Not the case with GEV's READREG/WRITEREG → confusion
- "Endianess of GigE Vision Cameras" appendix to GenICam standard

Camera Family Support

When implementing a camera family, the developer can

- Use separate XML specific for each device
- Use common XML and parameterize it with help of dedicated camera registers

With such design the camera can report

- Different values and limits of individual features
- Different access mode (incl. NI) of selected features using inquiry bits
- Applies also for chunk/event data

XML Syntax

The XML syntax is defined by the standard and enforced by the schema

- Additional verifications performed by the reference implementation (→ runtime only)
- Some checks are impossible or missing in the schema
- Extended checks can be performed by custom made Schematron rules during development
- Multiple schema versions (so far 1.0 and 1.1)

Creating GenICam XML

Creating XML using a generator

- Requires investment in the generator development
- Will probably limit the possibly used XML vocabulary
- Might help to limit errors
- Might help to output different XML flavors (optimization...)
- Might help to create XML for multiple schema versions

Creating XML manually

- Allows to start quickly
- Does not impose any limitations on the XML design
- Part of the advantages of the generator approach can be achieved with help of XSLT processing

Manual XML Creation

Discuss options for the manual XML creation

- oXygen XML editor
- Extended Schematron checks, applied as-you-type in oXygen or "offline" using a validator tool
- SFNC compatibility Schematron checks
- Possible XSLT adaptions of the manually created XML (strip off unneeded stuff, versioning, post processing)

Deploying GenICam XML

The XML needs to be versioned with each new release

- XML version, GUID's
- File name
- Manifest table
- Should be performed automatically to avoid mistakes
- Failing to update version info might cause that some SW libraries will not use the updated XML after FW update

Reuse of the GenApi Idea

GenApi is used to

- Control GenICam compliant devices (original use case)
- Reused to control individual GenTL modules (discussed later)
- Some vendors reuse it to control even other entities within their software package to provide similar user experience everywhere

GenApi Reference Implementation

The committee provides reference implementation of GenApi

- Not official part of the standard but almost universally used
- Simplifies device testing
- Details in SVN

Programming GenApi

Not essential for camera development as long as suitable regression tests can be established → currently not covered by this presentation

Refer to GenApi test suite in SVN

4

GenTL GenICam Module

Discover, configure, acquire

GenTL - Bridging Transport Technologies

GenTL allows extensive software-device interoperability, fully wrapping specifics of individual standard or custom transport technologies

- Device enumeration and information
- Device control (builds on GenApi)
- Streaming (buffering and acquisition engine interface), including chunk data support
- Asynchronous events delivery
- Allows to build generic interfaces fully agnostic to the technology running under the hood

GenTL User Base

Extensive and proven GenTL support from major library vendors

- MVTec Software, STEMMER IMAGING, MathWorks, Matrox

Implemented by ever growing number of device vendors

- Common interface for all product families
- Many use it as standard internal layer (under proprietary SDK)
- Essential for smart cameras and special cameras requiring additional software-side handling (3D)

Mandatory for CoaXPress (not yet CameraLink-HS)

- Could be turn point for some still resisting key players

GenTL vs. On-the-wire

With on-the-wire standardized technologies direct or GenTL based connection is possible

- Some vendors prefer more direct grip of the device (no 3rd party driver in the system)
- GenTL allows device vendor extensions and optimizations
- GenTL is much easier to fully implement (only big SW vendors can afford to implement and maintain all the direct interfaces)
- GenTL is a must outside GEV/U3V to ensure interoperability with major library products
- Direct on-the-wire interface is useful anyway, allows to simplify the system where on-the-wire implementation is available

Software Architecture

GenTL standard defines interaction between

- **GenTL Producer** - software knowing all aspects of the transport technology used by given device (or device class) and implements that technology by means of defined GenTL interface
- **GenTL Consumer** - generic software aiming to use devices accessible through GenTL interface without specific knowledge about their transport technology

GenTL Producer is a shared library (dll/so/...)

- Filename has to use extension .cti
- Its installation directory must be appended to environment variable GENICAM_GENTL{32/64}_PATH

Software Architecture (cont.)

Additional important notes:

- Producers are expected to be always dynamically loaded (LoadLibrary&GetProcAddress/dlopen&dlsym), exported functions must be undecorated
- On x86 stdcall is used regardless of the operating system
- Beware that producers implementing older GenTL version might be missing some function exports and other functionality
- GenTL Producer is responsible for thread and inter process safety

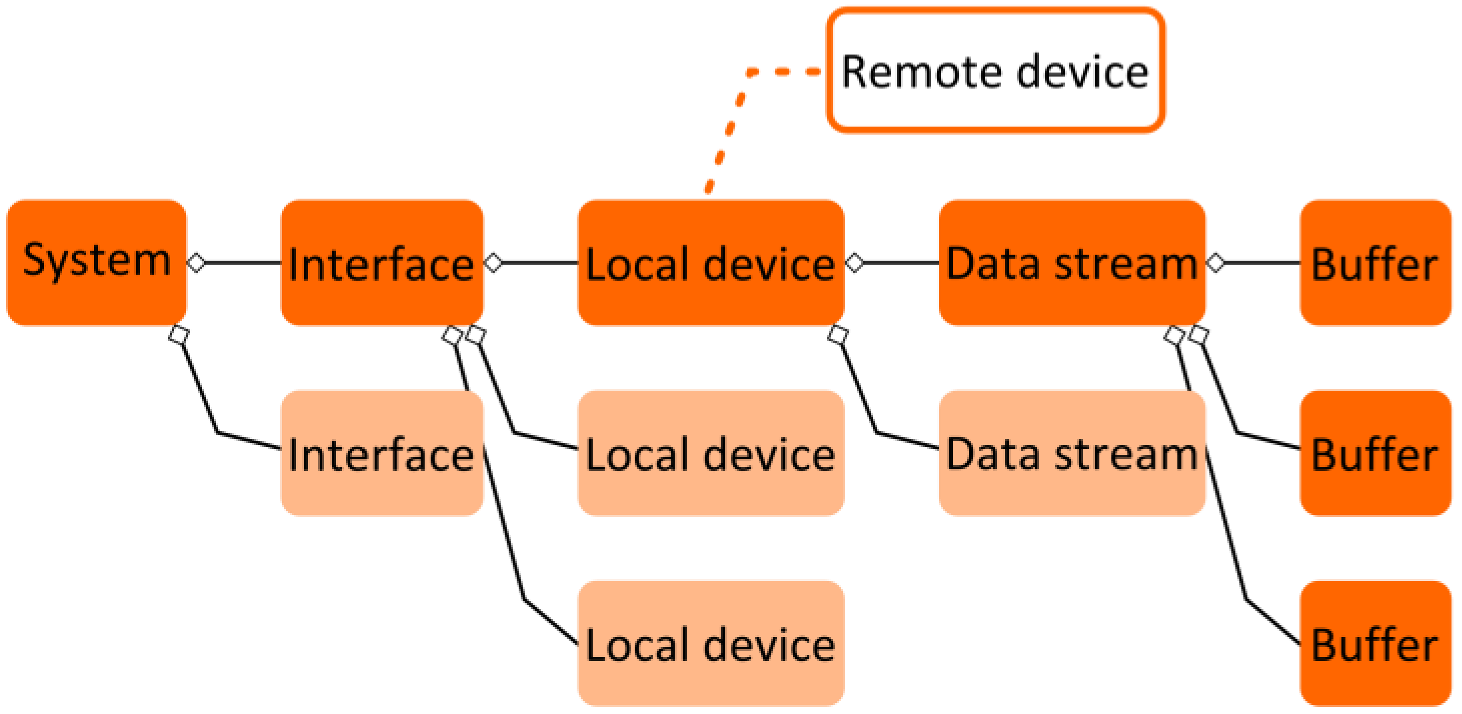
Programming Interface

Properties of the GenTL programming interface:

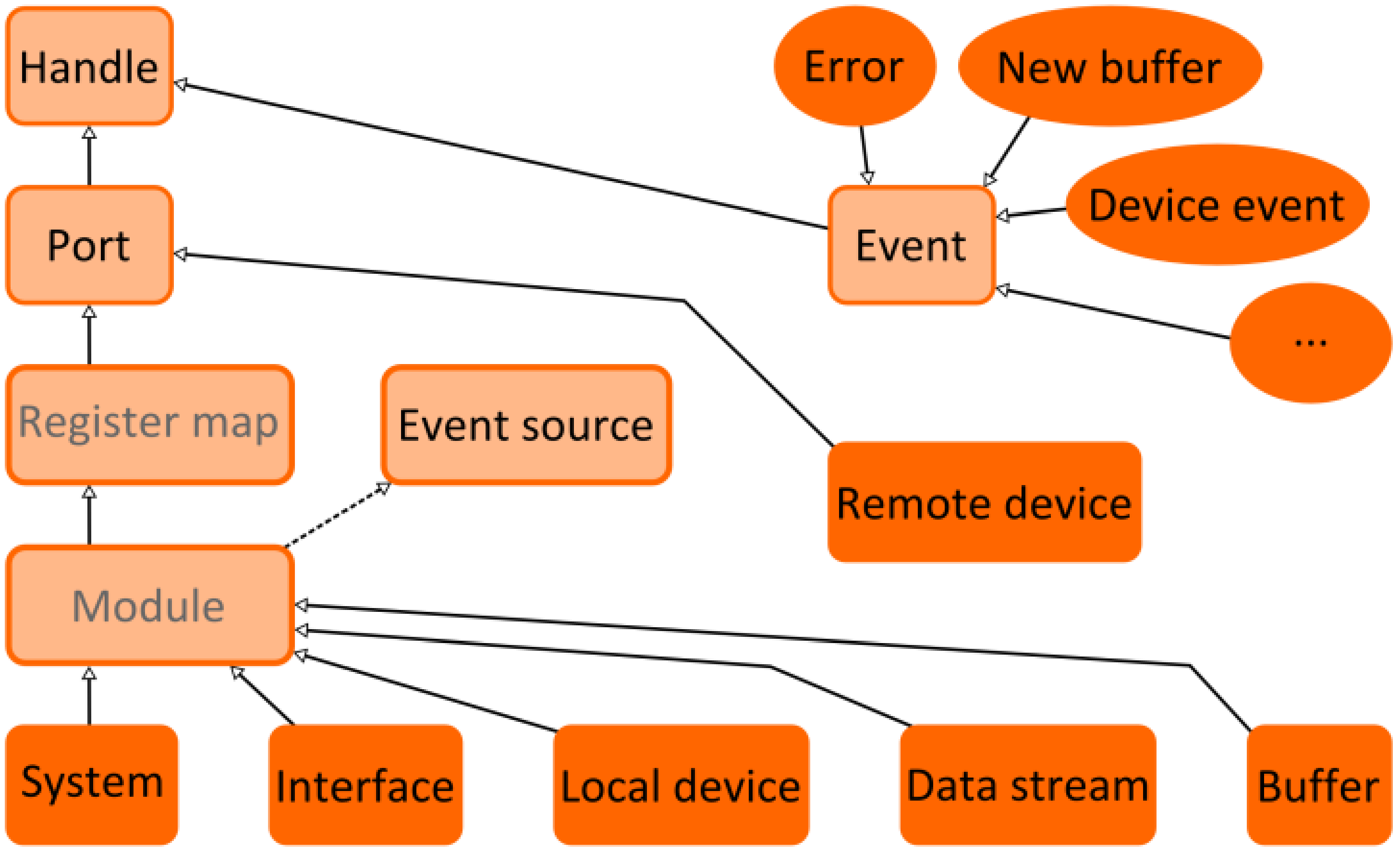
- Designed as ANSI C interface
- OS and platform independent
- All interaction initiated from GenTL Consumer, no callbacks, rather polling
- The C interface can only query information about individual modules, their control is performed always by means of GenApi nodemap

The reference header file implementing the specification is provided by the committee for convenience

GenTL Modules - Ownership Hierarchy



GenTL Modules - Programming Hierarchy



GenTL Modules

System - represents the entire GenTL Producer and its global properties

Interface - HW/SW entity used to discover and communicate with devices (it can represent e.g. a network card or a USB bus)

Local device - GenTL Producer's proxy to the actual "remote" device and is used to describe and establish communication with the real device

Data stream - stream of the (typically image) data acquired from the device, can represent a real data stream of the device or data stream generated by the GenTL Producer

Buffer - encapsulates the actual memory buffer used to acquire image or other data

Logical Module Interfaces

Port - provides register access (read/write), allows to establish GenApi based access to given entity (via "manifest table")

Event - allows polling for all kinds of asynchronous events

Event source - registers/unregisters generation of individual event types by given module

Remote device - represents the actual physical device, in particular its port interface

Library - initialization/cleanup of the GenTL producer, basic information

Module Enumeration

Common principles

- Modules identified through unique and persistent ID's
- Instances referred to by handles
- Handles stay valid until closed (directly or recursively from parent)
- Option to direct-open using the known ID
- Each instance opened just once (no reference counting)
- Eventual inter-process issues are responsibility of the producer

Module Enumeration (cont.)

Always a single system module

Interfaces & devices

- Dynamic consumer-controlled enumeration (semi hotplug)
- List stays intact until next update request
- Dual view & list control (C interface, nodemap) - beware of sync and nodemap caching problems
- Actually plugged devices, enumerated devices and devices with valid handle might be three different sets

Data streams

- Currently lifetime fixed list of streams expected
- Dynamic list control considered for future

Module Enumeration (cont.)

Buffers

- (Allocated and) announced by the consumer
- Same buffer can be announced to multiple streams (if the producer supports it)
- Not while streaming (optionally supported since GenTL 1.5)
- Composite buffer option (since GenTL 1.6)

Events

- Registered/unregistered based on the event type rather than ID

Remote device

- Lifetime equals Open-Close period of the local device
- Handle can be obtained ("opened") but not explicitly closed

Module Information

Information about individual entities published by GenTL Producer is queried using common info-command infrastructure

- Always the same mechanism
- Set of standard data types (integer types, string, string list, buffer)
- For each info command the data type is specified
- Producer can publish custom info-commands, Consumer can use them only if knowing what they mean
- GC_ERR_NOT_IMPLEMENTED, GC_ERR_NOT_AVAILABLE error codes
- Since GenTL 1.5 selected commands are marked as mandatory

Module Control

GenTL modules are controlled based on GenApi analogically as devices

- XML file(s), manifest table (URL retrieval, typically local)
- Port read/write support, virtual register map
- Optional for buffer module
- GenTL SFNC

Grabber based GenTL Producers

- Problem with sync between the grabber (local device) and the actual remote device - no reliable way to know actual status of the remote device
- Provisional workaround in CXP spec
- Full solution might need to be proposed and standardized

Module Signaling

Asynchronous information delivered to consumer through events

- Polled - no callbacks, all operations consumer-triggered
- Obtain data, query additional info (ID/value)
- Cleanup (event kill): 1:1 kill-abort mapping, recharge after re-registration
- Different modules support different events

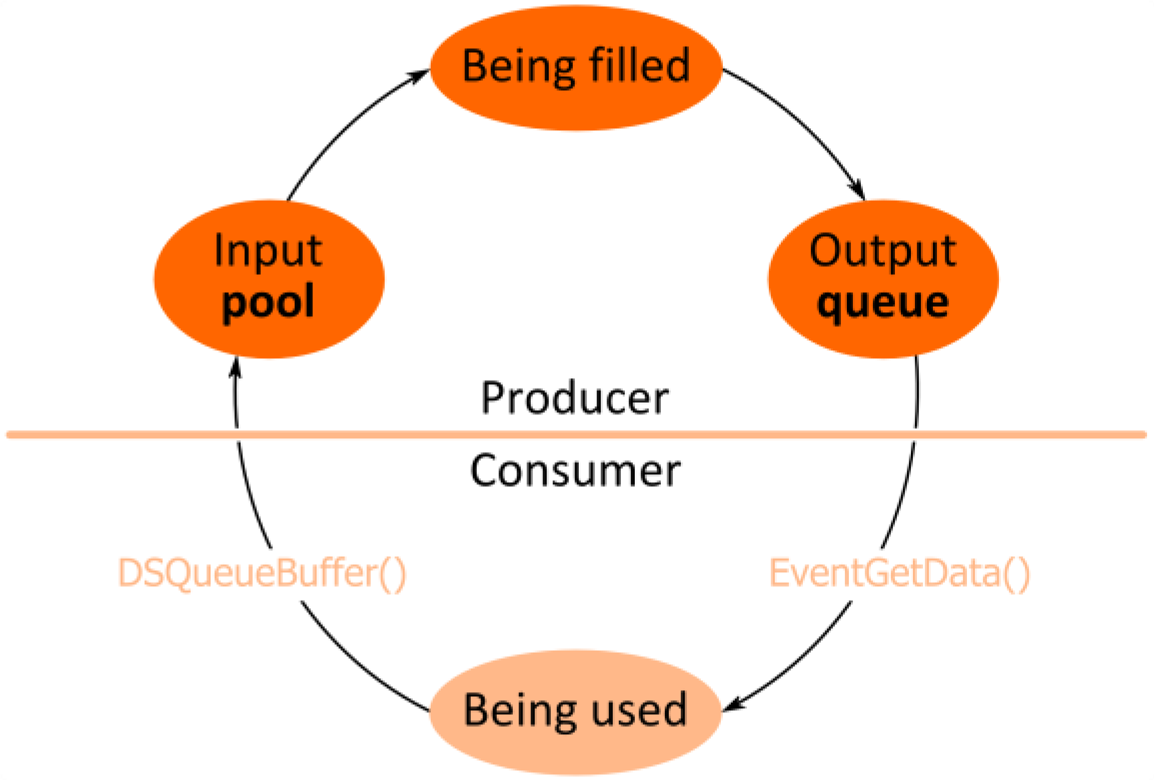
Event types

- Error (universal)
- New buffer (streaming)
- Remote device & module events (nodemap-mappable)
- Feature invalidate & change (limited use, SFNC-bound)

Acquisition Control

1. Allocate and announce acquisition buffers
   - consumer/producer allocated
   - payload size (device/modified), variable payload size, flow table
2. Start acquisition
   - on producer (possible overhead) and device
   - TLParamsLocked (still not quite standardized, hopefully soon)
   - register new buffer event if not already registered
3. Acquisition loop (next slides)
4. Stop acquisition
   - on device (and producer)
   - (TLParamsLocked)
5. Cleanup (if finished)
   - revoke buffers, memory released by entity that allocated it
   - announce/revoke only when acquisition is not active

Acquisition Loop



Acquisition Loop (cont.)

Additional notes about acquisition loop:

- Buffer state also affected by data-stream and new-buffer-event flush operations
- Presence of new data and their validity can be (especially after a flush) queried through corresponding buffer infos
- Internal status of the buffer is undefined while owned by the producer

Insufficient acquisition buffer:

- Setup time: refuse to start acquisition
- Runtime: fire error event (and optionally deliver part of the data)

Acquired Buffer Properties

When processing a new buffer, the order of querying its parameters is

1. DSGetBufferInfo (would cover also producer modifications)
2. Chunk data (is clearly bound to the buffer)
3. (Producer nodemaps → but not yet well standardized)
4. Device nodemap (might not be in sync with actual buffers)

Important info's include:

- Basic buffer info (address, size, eventually segments)
- Payload type, new data flag, data completeness
- Image properties (pixel format, AOI, not in GenDC case)

In some cases (3D) the set of essential infos is so wide that it's not practical to standardize them through buffer info → chunk data.

Acquired Buffer Properties (cont.)

Option to query multiple properties through single call

- DSGetBufferInfoStacked (since GenTL 1.6)
- Reduces per-buffer overhead associated with numerous consumer-producer calls
- Separate per-info error handling
- Available also for multi-part buffers (see later)

Multi-part Buffers

Single buffer can optionally contain multiple sets of data belonging together ("same timestamp") - available since GenTL 1.5

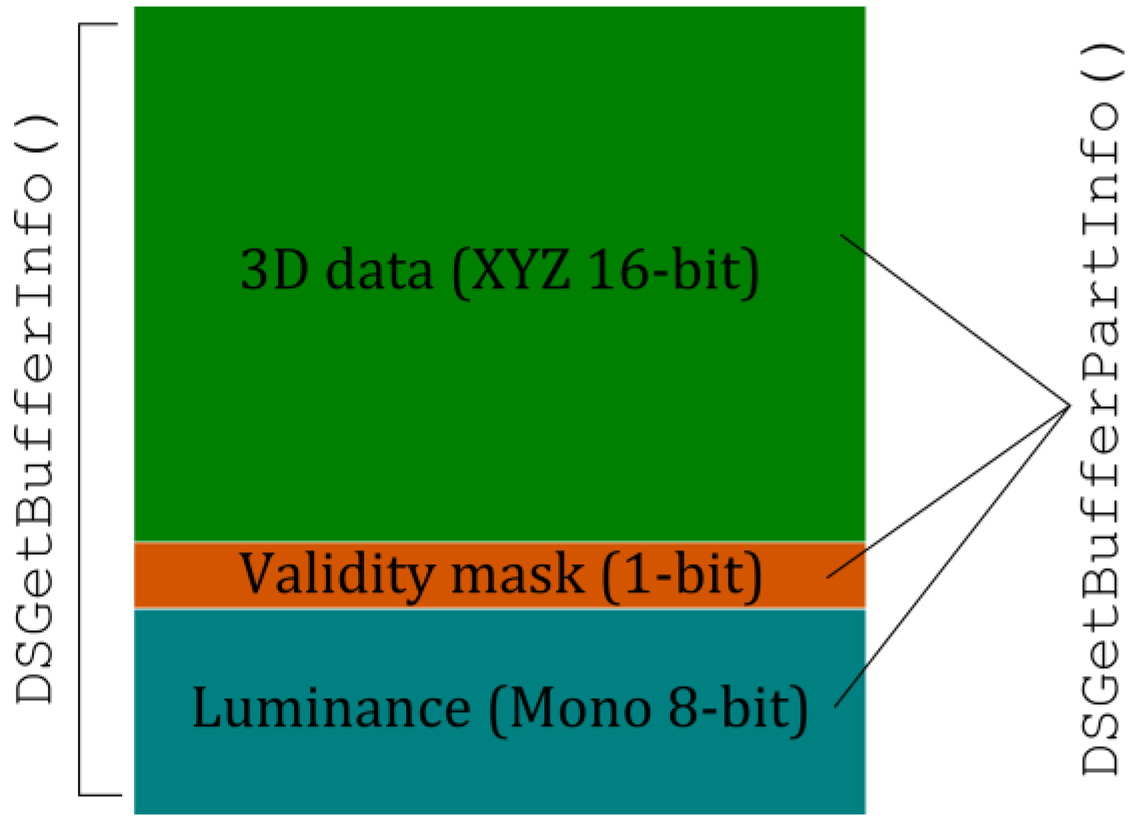
Example use cases:

- Planar pixel formats
- Multiple AOI's
- 3D data exchange (see dedicated presentation)
- Multi-source devices
- Non-rectangular images (data + mask)

Implementation:

- Dedicated multi-part payload type
- Functions to report number of buffer parts and their properties

Multi-part Buffers (cont.)



Composite Buffers

Option to acquire data in structured manner - available since GenTL 1.6

- Multiple separate segments instead of contiguous buffer
- Separate processing chains, GPU (?), …
- DSAnnounceCompositeBuffer to announce composite buffer
- Revoking and acquisition flow same as with regular buffers
- Mapping data to segments use case specific (currently explicitly defined for GenDC case)

Usage notes

- Consumer must know the intended structure (e.g. flows…)
- Care needed when querying data properties incl. chunks
- Might be wise to prefer contiguous buffers by default (also for backward compatibility)

Data Stream Flows

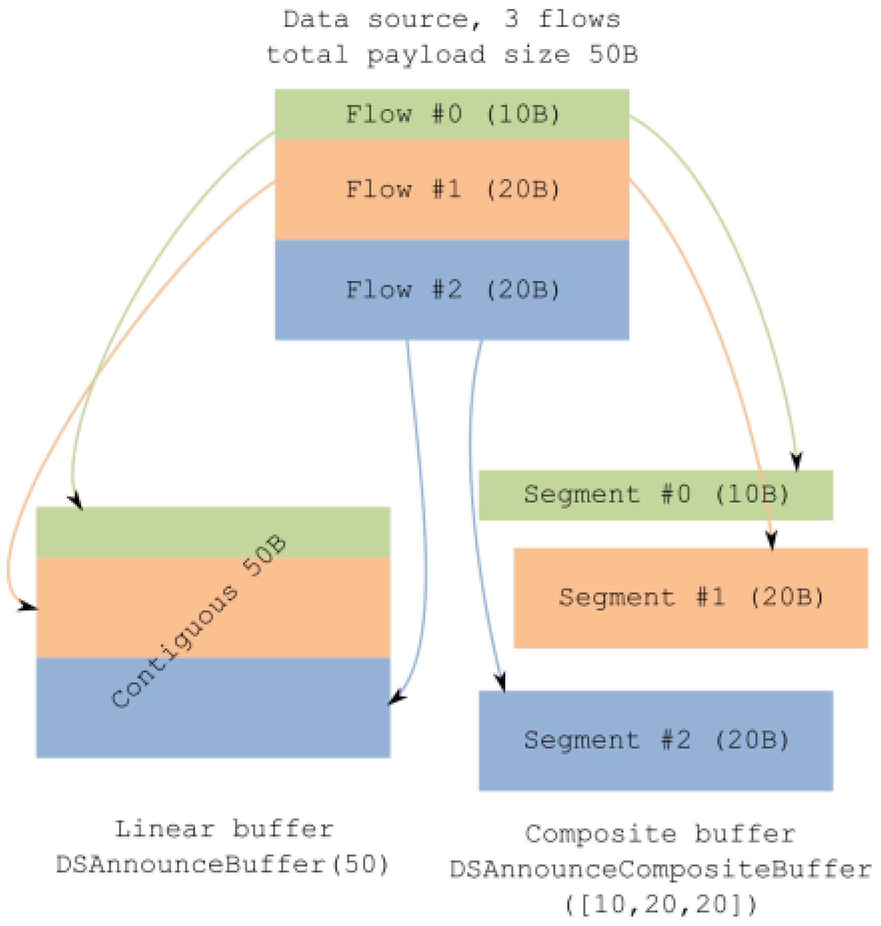
Independent “channels” within data stream - available since GenTL 1.6

- First introduced in GenDC, in GenTL generalized for any use case
- Allows to minimize latency/buffering (each flow data transferred as they become available)
- Allows to transfer data to discrete memory locations (when used with composite buffers)
- Depends on flow related capabilities of given TL

Usage notes

- Composite buffers – flows acquired to matching segments
- Contiguous buffers – flow structure linearized in the buffer by GenTL Producer (transparent for GenTL Consumer)
- Flow mapping table vs. PayloadSize (configuration dependent)

Data Stream Flows (cont.)



GenDC Containers

The buffer can be delivered as standard GenDC container - available since GenTL 1.6

- Principles and structure very similar to multi-part
- Starts with standard header
- Payload details (incl. chunk data) queried through the GenDC header instead of GenTL calls
- Refer to section about GenDC standard module

GenDC in different buffer types

- Contiguous – linear container, component/part data located using linear DataOffset
- Composite – per-flow structured container, component/part data located using FlowOffset

Payload Modification

When GenTL Producer modifies payload, it's even more important to report all the (possibly modified) buffer properties:

- Payload type
- **Payload size**
- Pixel format
- AOI (W/H, offsets, paddings)
- (repeat problem of producer not knowing the device config)

GenTL Consumer should always prefer buffer infos to device nodemap
Important properties not included in buffer info's must be reported through chunk data or GenTL SFNC

- 3D is an example use case relying on additional information transferred through the chunk data

Buffer Handling Modes

Alternative ways how to order delivery of acquired buffers and specify which (if any) buffers are discarded by the acquisition engine

Oldest first (default, mandatory): FIFO delivery, no buffers discarded from the output queue, incoming data dropped on overflow

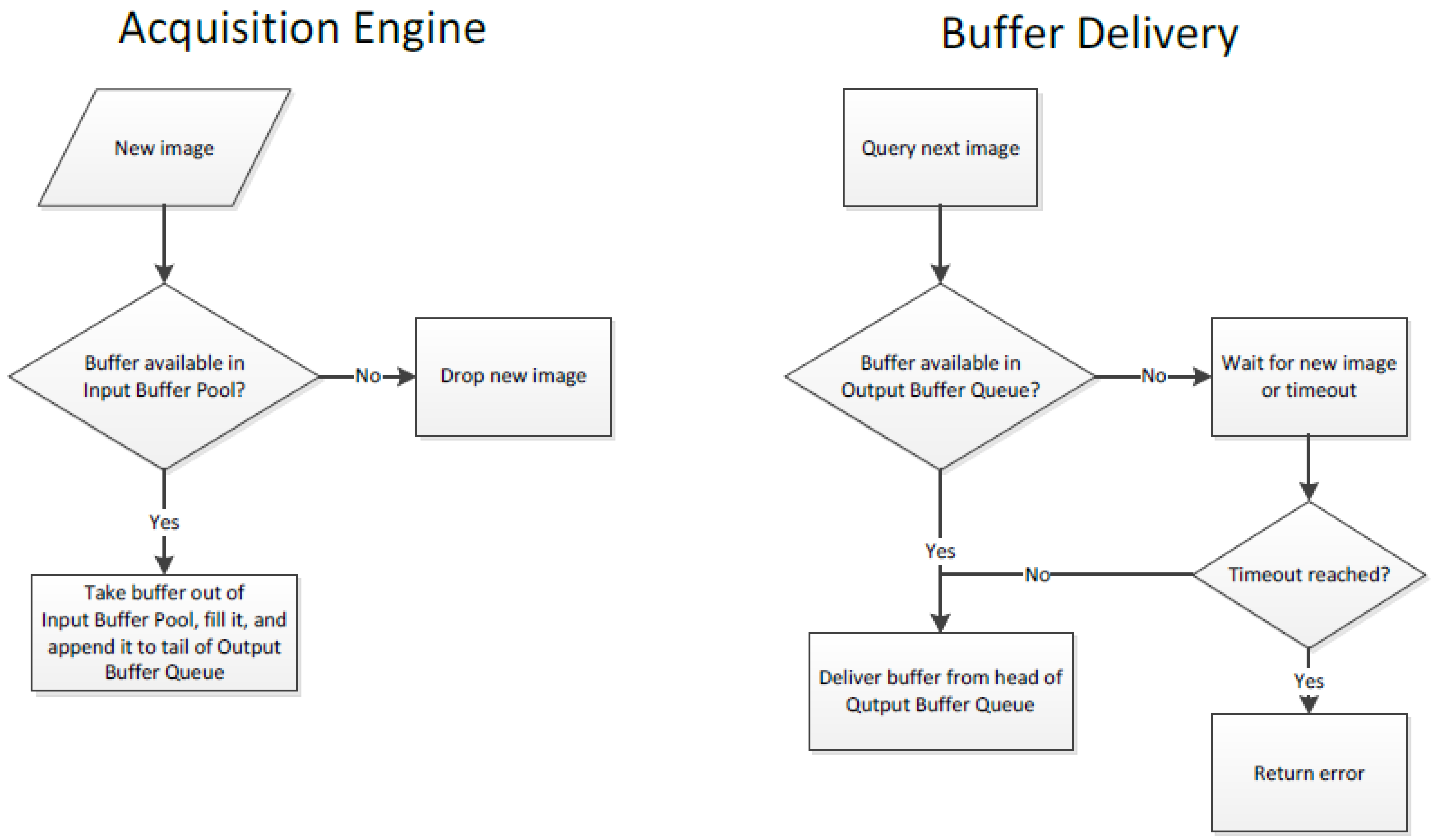
Oldest first overwrite: FIFO delivery, in case of overflow the oldest buffer in the output queue is reused

Newest only: Deliver only the newest buffer, reuse older buffers immediately for new acquisitions - this mode was added in GenTL SFNC 1.1

Other modes are yet to be standardized

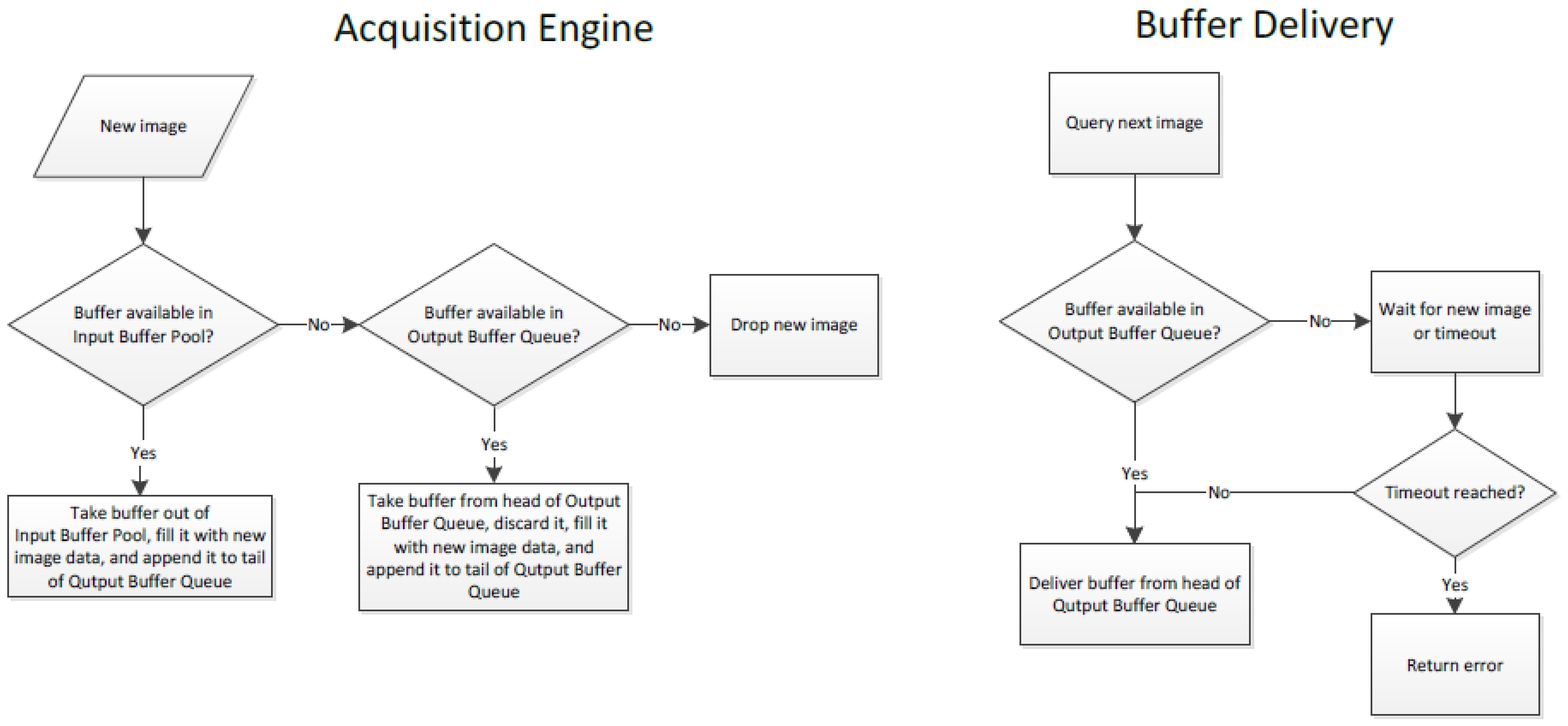
- Deliver only buffer arrived after the request (no output queue)?
- Other?

Buffer Handling Mode Oldest First



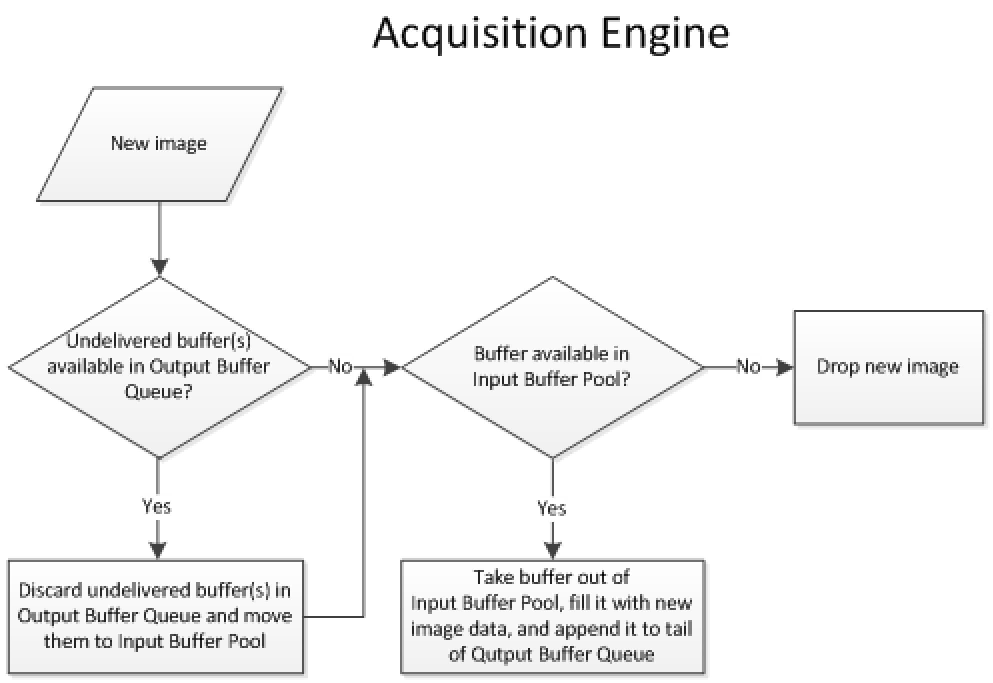
Acquisition Engine
New image
Buffer available in Input Buffer Pool?
No
Drop new image
Yes
Take buffer out of Input Buffer Pool, fill it, and append it to tail of Output Buffer Queue
Buffer Delivery
Query next image
Buffer available in Output Buffer Queue?
No
Wait for new image or timeout
Timeout reached?
No
Yes
Deliver buffer from head of Output Buffer Queue
Yes
Return error

Buffer Handling Mode Oldest First Overwrite

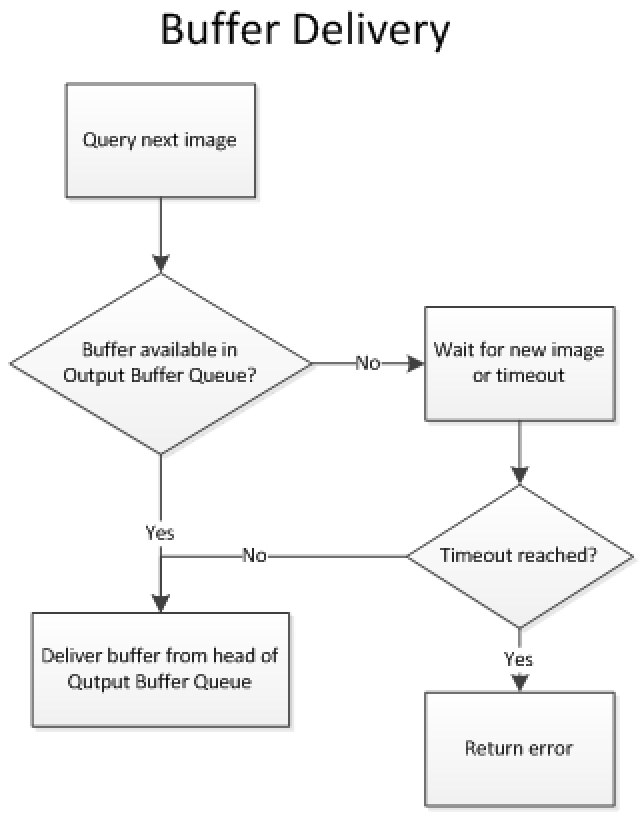


Acquisition Engine
New image
Buffer available in Input Buffer Pool?
No
Yes
Buffer available in Output Buffer Queue?
No
Yes
Drop new image
Take buffer out of Input Buffer Pool, fill it with new image data, and append it to tail of Qutput Buffer Queue
Take buffer from head of Output Buffer Queue, discard it, fill it with new image data, and append it to tail of Qutput Buffer Queue
Buffer Delivery
Query next image
Buffer available in Output Buffer Queue?
No
Yes
Wait for new image or timeout
Timeout reached?
No
Yes
Deliver buffer from head of Qutput Buffer Queue
Return error

Buffer Handling Mode Newest Only



Acquisition Engine
New image
Undelivered buffer(s) available in Output Buffer Queue?
No
Yes
Discard undelivered buffer(s) in Output Buffer Queue and move them to Input Buffer Pool
Buffer available in Input Buffer Pool?
No
Yes
Drop new image
Take buffer out of Input Buffer Pool, fill it with new image data, and append it to tail of Output Buffer Queue



Buffer Delivery
Query next image
Buffer available in Output Buffer Queue?
No
Yes
Wait for new image or timeout
Timeout reached?
No
Yes
Deliver buffer from head of Output Buffer Queue
Return error

Non-streamable Devices

GenTL supports devices without any streaming support

- Allows to include auxiliary devices such as strobes etc. within the GenTL framework
- Such devices provide zero data streams and any streaming related functionality is reported as not-implemented
- Main goal is to allow GenApi control of the device
- The device still can support asynchronous events

Chunk Data

The GenTL Producer can use one of the established chunk formats (GEV/U3V) or invent custom way to transfer chunk data.

If the format is known (DEVICE_INFO_TLTYPE/DeviceChunkDataFormat), the acquired buffer can be parsed with corresponding native adapter.

For GenDC payloads, the chunk data should be parsed based on GenDC rules (without help of GenTL Producer).

For unknown format, the buffer must be parsed by producer itself (DSGetBufferChunkData) and applied through the generic adapter.

Extra care with composite buffers.

Event Data

The GenTL Producer can use one of the established event formats (GEV/U3V) or invent custom way to transfer chunk data.
To reuse an established format, the producer might need to create fake EVENT_CMD header for given technology as well.

If the format is known (DEVICE_INFO_TLTYPE/DeviceEventDataFormat), the acquired buffer can be parsed with corresponding native adapter. The entire event (including all headers) must be passed through EventGetData.

For unknown format, the port addressable part of the event must be returned through EventGetDataInfo(EVENT_DATA_VALUE) and applied through the generic adapter.

Error Handling

Error codes for most typical failures standardized since GenTL 1.4

- Still new and thus not reliable
- Custom error codes used where standard ones do not suffice

Use of selected important error codes

- GC_ERR_SUCCESS
- GC_ERR_NOT_IMPLEMENTED/_NOT_AVAILABLE (versioning)
- GC_ERR_RESOURCE_IN_USE (module enumeration)
- GC_ERR_ABORT (event kill)

Last error handling

- Thread local
- Not affected by any successful operations following the error

GenTL Validation

GenTL Validator

- The official version available from SVN
- Validating and developer modes
- Very useful for GenTL Producer testing
- Can be further extended for in-house unit tests
- Tests just some aspects of GenTL interface, extensions desirable

GenTL certification

- Optional at plug-fests, to be mandatory since 1.6 (?)
- Self-certification, producer-consumer cross testing
- Similar approach as GigE Vision, USB3 Vision, CoaXPress
- GenICam Transport Layer logo

Future Standardization Topics

Selected important topics waiting for better standardization:

- Support for grabber based technologies (in particular how the grabber knows the current configuration of the remote device)
- Identification and synchronization of streams belonging together, matching GenTL streams and physical device streams (SFNC control) - some use cases covered by multi-part and GenDC
- Standardization of module specific error events (GenTL SFNC)
- GenTL Producer capability/configuration handling similar to GEV

GenTL SFNC Status

Still young module, recently revived

- Initial version built mostly around GenTL module infos
- New functionality starting to be added

Should reuse "device" SFNC feature names wherever possible

- Add GenTL specific features
- Perhaps define which feature might apply for which module
- Waterfall vs. concrete module approach

Important for GenTL producers modifying the payload (grabbers, smart producers, 3D)

- Might require further standardization

GenTL Producer Framework

Framework for implementation of GenTL Producers

- Official GenICam module based on code donated to the community by MVTec
- Quick start, less errors, focus only on TL specifics
- GenTL Core – static library wrapping most of common GenTL functionality, providing hooks for TL implementation
- Viky – toy GenTL Producer based on GenTL Core to demonstrate the framework usage and to allow direct experiments
- Retrieve from website or svn

More details in dedicated presentation (delivered together with GPF)

- Directory “doc” within GPF

5

GenDC GenICam Module

Generic data container

GenDC Purpose and History

GenICam module defining standard container to store/carry versatile image and other stream data

- Evolved from multi-part, almost identical structure
- Contains standard container header with same format across TL’s and allowing to store the container in a file

Current status

- Version 1.0 released in December 2018
- Some aspects still (2019) under discussion, to be clarified in next version
- To be adopted by GenTL (1.6), GEV (2.2) and other standards

GenDC in GenTL Context

Typical GenTL Producer does not need to care about contents of the GenTL payload

- GenDC is orthogonal, GenTL Producer only transports the container from remote device to GenTL Consumer
- GenTL Consumer is responsible to parse the container

Exceptions

- If GenTL Producer modifies the GenDC payload or transforms non-GenDC payload to GenDC
- If the remote device is virtualized within GenTL Producer
- If forced by GenDC related requirements of the underlying transport layer technology

6

Other GenICam Modules

Beyond GenApi, GenTL and SFNC

Modules not Covered in Detail

GenCP

- Communication protocol for new and future standards
- Designed to be similar to GigE Vision's GVCP
- Used in USB3 Vision, CameraLink-HS, sometimes CameraLink
- Covers control and message channel

CLProtocol

- Bridge between GenICam and Camera Link
- Allows to discover and configure CL cameras, still using vendor specific communication protocols and the clserial.dll mechanism
- Could be eventually replaced by GenCP?

7

Testing & Debugging

What if something goes wrong?

GenICam Testing & Debugging Hints

GenTL Validator

- Possibly extended with custom checks

Own test program

- Could be trivial consumer with possible ad-hoc changes

GenTL proxy

- But prefer good and extensive own logging

XML debugging

- Show status of internal features (modify XML)
- GenApi logging (GENICAM_LOG_CONFIG_V2_4=PathTo/Logging.propeties)

HALCON Debugging Support

Information in HDevelop's Output Console

Environment variables

- HALCON_GENTL_LOGGING, HALCON_LOG_LEVEL, HALCON_OWN_GENICAM_LOG_CONFIG
- HALCON_GENTL_WRITE_XMLFILE, HALCON_GENTL_WRITE_RAW_XMLFILE (GENTL will be replaced with GENICAM)

Command 'do_write_configuraiton'

- Stores all involved XML's in an ini file
- Allows to re-open next time, loading (possibly modified) XML's specified in the ini file

Get familiar with HALCON's GenTL interface documentation

8

Information Sources

How to stay in sync

Where to Get Information

Official website www.genicam.org

- Useful in particular to download all official releases and minutes from technical meetings

Mailing list genicam@list.stemmer-imaging.com

- Official place to ask for help
- Many of technical discussions (even historical - archive)

SVN-Trac-Forum-Wiki genicam.mvtec.com/trac/genicam

- Documents, source code, work on future versions
- Issue & homework tracking
- Technical discussions, various info (wiki)

FTP GenICam-rw@ftp.baslerweb.com

- Mainly presentations from technical meetings and releases
- Access details in Trac Wiki (above)

Groget ☺ (www.groget.org)

SVN: What to Look For?

Most useful data available from GenICam SVN repository:

- GenTL Producer Framework and its Viky example producer
- GenApi reference implementation and its test suite
- GenTL validator
- SFNC reference XML and Schematron validation

Beware of possibly loosely maintained stuff:

- In particular GenTL producer/consumer sample code
- Possible incomplete/obsolete howto's in the wiki

GenICam Group Membership

GenICam group membership

- Is free (for companies/organizations)
- Provides access to the group's resources and knowledge base

How to join:

- On www.genicam.org navigate to page named "GenICam Group Members" (the actual link might change)
- The page contains section "How to become an associated member?" with corresponding instructions

What else?

Further discussion...

Time to start with actual implementation?

Groget

GEN<i>CAM Standards