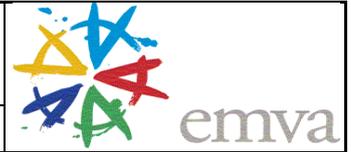


GEN<math>i</math>CAM

GenCP Standard



GenICam

**GenCP**

Generic Control Protocol

Version 1.0

03 December 2012

## Content

1.	Introduction.....	7
1.1.	Motivation.....	7
1.2.	Objective.....	7
1.3.	Abstract.....	8
1.4.	Acronyms.....	9
1.5.	References.....	10
1.6.	Requirement Terminology.....	10
2.	Definitions.....	11
2.1.	Device Description File.....	11
2.2.	String Encoding.....	11
2.3.	Byte and Bit Order.....	11
2.3.1.	Serial.....	11
2.3.2.	USB.....	11
2.3.3.	Other.....	12
2.4.	GenCP Version.....	12
2.5.	CRC.....	12
2.6.	Technologies.....	12
2.7.	Link.....	12
2.8.	Channel.....	13
2.8.1.	Default Channel.....	13
2.9.	Compression.....	13
3.	Operation.....	14
3.1.	Protocol.....	14
3.1.1.	Command & Acknowledge Mechanism.....	14
3.1.2.	Pending Acknowledge.....	16
3.1.3.	Message Channel.....	17
3.1.4.	Failure.....	18
3.2.	Heartbeat.....	22
3.3.	GenICam File.....	23
3.3.1.	Manifest Table.....	23



- 3.3.2. Retrieval .....23
- 3.4. Serial.....23
  - 3.4.1. Packet Size .....23
  - 3.4.2. Serial Parameters.....23
- 4. Packet Layout.....25
  - 4.1. General Packet Layout .....25
  - 4.2. Prefix .....27
    - 4.2.1. Serial .....27
  - 4.3. Common Command Data .....28
    - 4.3.1. Command Packet Layout .....28
    - 4.3.2. Acknowledge Packet Layout .....29
    - 4.3.3. Command IDs .....31
  - 4.4. Command Specific Data.....32
    - 4.4.1. ReadMem Command .....32
    - 4.4.2. ReadMem Acknowledge .....32
    - 4.4.3. WriteMem Command.....33
    - 4.4.4. WriteMem Acknowledge .....33
    - 4.4.5. Pending Acknowledge.....34
    - 4.4.6. Event Command.....35
    - 4.4.7. Event Acknowledge .....35
  - 4.5. Postfix.....35
    - 4.5.1. Serial .....36
- 5. Bootstrap Register Map .....37
  - 5.1. Technology Agnostic Bootstrap Register Map.....37
  - 5.2. String Registers .....37
  - 5.3. Conditional Mandatory Registers.....37
  - 5.4. Register Map .....38
    - 5.4.1. GenCP Version .....40
    - 5.4.2. Manufacturer Name .....40
    - 5.4.3. Model Name.....41
    - 5.4.4. Family Name.....41
    - 5.4.5. Device Version .....42

5.4.6.	Manufacturer Info .....	42
5.4.7.	Serial Number .....	43
5.4.8.	User Defined Name.....	43
5.4.9.	Device Capability.....	44
5.4.10.	Maximum Device Response Time (MDRT) .....	45
5.4.11.	Manifest Table Address .....	46
5.4.12.	SBRM Address .....	46
5.4.13.	Device Configuration .....	47
5.4.14.	Heartbeat Timeout .....	47
5.4.15.	Message Channel ID .....	48
5.4.16.	Timestamp .....	49
5.4.17.	Timestamp Latch .....	50
5.4.18.	Timestamp Increment.....	51
5.4.19.	Access Privilege .....	52
5.4.20.	Protocol Endianness .....	53
5.4.21.	Implementation Endianness .....	54
5.5.	Technology Specific Bootstrap Register Map.....	54
5.5.1.	Serial .....	54
5.5.2.	USB3 Vision.....	56
5.6.	Generic Tables .....	57
5.6.1.	Manifest .....	57

**List of Figures**

Fig. 1 - Command Cycle ..... 15

Fig. 2 – Pending Ack Cycle ..... 16

Fig. 3 – Event Cycle..... 17

Fig. 4 – Command Failure ..... 20

Fig. 5 – Ack Failure..... 21

Fig. 6 - Serial Parameter Change ..... 24

Fig. 7 – General Packet Layout..... 25

List of Tables

Table 1 - Acronyms ..... 9

Table 2 - Event ID ..... 18

Table 3 – GenCP Event IDs ..... 18

Table 4 - Serial Prefix ..... 27

Table 5 - Common Command Data ..... 28

Table 6 - Acknowledge layout ..... 29

Table 7 – Status Codes ..... 30

Table 8 – Command Identifier ..... 31

Table 9 - ReadMem SCD-Fields ..... 32

Table 10 - ReadMem Ack SCD-Fields ..... 32

Table 11 - WriteMem Command SCD-Fields ..... 33

Table 12 - WriteMem Ack SCD-Fields ..... 33

Table 13 - Pending Ack SCD-Fields ..... 34

Table 14 - Event Command SCD-Fields ..... 35

Table 15 - Event Acknowledge SCD-Fields ..... 35

Table 16 - Technology agnostic BRM ..... 39

Table 17 - Register GenCP Version ..... 40

Table 18 - Register Device Capabilities ..... 44

Table 19 - Register Maximum Device Response Time ..... 45

Table 20 - Register Manifest Table Offset ..... 46

Table 21 - Register Technology Specific Bootstrap Register Map ..... 46

Table 22 - Register Device Configuration ..... 47

Table 23 - Register Heartbeat Timeout ..... 47

Table 24 - Register Message Channel ID ..... 48

Table 25 - Register Timestamp ..... 49

Table 26 - Register Timestamp Latch ..... 50

Table 27 - Register Timestamp Increment ..... 51

Table 28 - Register Access Privilege ..... 52

Table 29 - Register - Protocol Endianess ..... 53

Table 30 - Register - Implementation Endianess ..... 54

Table 31 - Serial BRM ..... 54

Table 32 - Register – Serial – Supported Baudrates ..... 55

Table 33 - Register – Serial – Current Baudrate ..... 56

Table 34 – Manifest Table Layout ..... 57

Table 35 - Manifest Entry Layout ..... 59

## 1. Introduction

### 1.1. Motivation

Today's Camera Link V 1.2 based products implement a wide variety of proprietary control protocols. Most of these protocols are based on ASCII command strings and ASCII responses. Proprietary protocols can be integrated into GenICam through the GenICam CLProtocol module, assuming the device manufacturer provides a dynamic link library (DLL) for all supported platforms/operating systems. This DLL does the translation between the camera-specific proprietary control protocol and a GenICam compliant register map, which allows the integration of a device into GenICam.

Providing a manufacturer-specific and platform-specific DLL adds cost and effort:

- It has to be maintained for various platforms and OS versions.
- Device features must be added and updated
- The integration of embedded platforms must be taken into account

A more straightforward way would be to provide a read/write register protocol on the serial link and do the register map integration in the camera. There would be only one place to change, the camera firmware, in order to introduce new features. There would be *no* platform-specific software needed, which would allow the use of embedded devices as the controlling host.

Some devices on the market implement serial protocols in a similar way already. The idea is to propose a common approach for implementing a protocol to give new implementers a hint and maybe to allow a de facto standard in the future.

The original idea was to ease the CLProtocol implementation by providing a protocol description. But because a protocol can potentially be used on other technologies as well, the definition is kept more generic. It can be adjusted to other technologies even though Camera Link is the first approach.

### 1.2. Objective

The objective of this document is to describe

- a packet-based protocol to read and write registers in a register-based device
- a Bootstrap Register Map (BRM) to provide basic device information
- access to the device's GenICam file
- the technology specific communication configuration

For example, for Camera Link this protocol could be used in the generic CLProtocol module to communicate with a manufacturer’s device over the Camera Link’s serial link. At boot up, the generic CLProtocol module would allow the configuration of the serial link. A “generic” software could download the GenICam file by accessing the camera’s registers. The software can then provide native GenICam (like GigE Vision) access to the device without the need for the camera vendor to provide a platform/operating system-specific software running on the host, implementing the translation between GenICam register access and manufacturer proprietary protocols.

### **1.3. Abstract**

The protocol is packet based. It follows a simple command/acknowledge scheme to provide resend and timeout capabilities, adding minimum overhead.

The Bootstrap Register Map (BRM) resides in a 64-bit register space. The 64 Kbytes starting on address zero contain technology agnostic information like manufacturer name, model name, etc., and provide a directory for technology specific settings.

In order to locate the GenICam file for a device, software would need to retrieve a list of available GenICam files, called the manifest, from the device’s register map. The software would then pick the best fitting GenICam file from the list and access via the device’s register map.

### 1.4. Acronyms

Name	Description
BRM	Bootstrap Register Map
ABRM	Technology Agnostic Bootstrap Register Map
SBRM	Technology Specific Bootstrap Register Map
Device	Device to be controlled, can be any entity, may not be a camera
Host	Controlling Master, can be any entity, may not be a PC
Link	Connection between a device and a host.
Channel	Logic communication channel between two entities. A Channel is always unidirectional.
Datagram	A single GenCP packet.
Entity	Either the Device or the host
DRT	Device Response Time The time a device needs to process a command not including the transfer time for the packet containing the command.
PTT	Packet Transfer Time Time to transfer a message/command over a link at a given link speed.
URL	Uniform Resource Locator
CCD	Common Command Data Section within a GenCP command packet which is common to all commands.
SCD	Specific Command Data Section within a GenCP command packet which is specific to a given command.

Table 1 - Acronyms

## **1.5. References**

Camera Link	AIA Camera Link Spec 1.2
GigE Vision	AIA GigE Vision Spec 1.2
GenICam	EMVA GenICam
RFC3986	URL
RFC791	Internet Protocol

## **1.6. Requirement Terminology**

Version 1.0 of the standard does not yet define a requirement scheme even though it is planned to apply that in future.

## 2. Definitions

### 2.1. *Device Description File*

Device Description File means a GenICam file describing the register space of a device.

### 2.2. *String Encoding*

All strings are encoded in ASCII, UTF8 or UTF 16 depending on the BRM setting. The endianness of the characters in an encoded string must match the endianness of the containing register map. So strings defined in the bootstrap register map must follow the endianness of the GenCP Protocol. Strings in the device's register map must follow the implementation endianness.

### 2.3. *Byte and Bit Order*

The order and size of fields within packets is **not** depending on the endianness used. Fields are always listed with its byte offset relative to the start of the section within a packet. All fields are byte aligned.

The endianness of all fields in GenCP protocol packets is technology specific and it must match the endianness of the bootstrap registers of the device. If the technology is not listed in this document it is assumed to be big-endian.

This document does not define or use explicit bit numbers but identifies bits by its offset to the least significant bit. This notation is supposed to be endian agnostic even though the offset matches the bit numbers of little-endian notations.

The endianness of the non-bootstrap registers is device implementation specific.

For reference the byte order is as described in Appendix B of RFC791.

#### 2.3.1. *Serial*

For Serial devices the byte order of bootstrap registers and protocol fields is big-endian.

#### 2.3.2. *USB*

For USB devices the byte order of bootstrap registers and protocol fields is little-endian.

### 2.3.3. *Other*

Unless explicitly stated for a given technology the endianness for GenCP-Implementations is big-endian.

## 2.4. *GenCP Version*

The GenCP version this document describes is

Major Version Number	1
Minor Version Number	0

A change in the Major Version Number indicates a significant feature change and a potential break in backward compatibility.

A change in the Minor Version Number indicates minor feature changes, bug fixes, text clarifications and assures backward compatibility.

## 2.5. *CRC*

The CRC checksum used on the packets depends on the underlying technology. If the underlying technology already provides a CRC, that service is used. If the underlying technology does not provide a CRC, the checksum is defined in the Prefix chapter for this technology.

## 2.6. *Technologies*

This document covers the following technologies:

- Serial (with the intention to be used on the serial connection of Camera Link)
- USB

Other technologies may be added over time.

## 2.7. *Link*

A link is the physical end to end connection between a device and a host used for control communication. For example, for Camera Link Medium, even though there are two cables carrying data there is only one serial link for the RS232 communication. Each link can carry multiple logic communication channels. GenCP assumes a single link between a host and a device.

## **2.8. Channel**

A channel is a logical communication path between two entities communicating over a link. There may be multiple logical channels on a single link. Each channel is identified by a unique id number. This number is used in the communication between two entities to identify the channel a given packet belongs to. This is either part of the protocol layers below the protocol described here or in the PacketPrefix (see chapter 4.2), depending on the technology. This number is called “channel\_id”. A channel’s communication is unidirectional, meaning that on a single channel, the sender and receiver side for commands and the sender and receiver side for acknowledges are fixed. Different logical channels may have different directions. The protocol also defines packet layouts and the communication scheme between a device and a host. This document assumes that the host is the command sender and the device is the command receiver even though the roles may change in real live.

### **2.8.1. Default Channel**

#### **2.8.1.1. Serial**

The default channel\_id for the control channel is channel\_id = 0.

## **2.9. Compression**

The compression methods used are DEFLATE and STORE of the .zip file format. File extension for compressed files is zip.

## 3. Operation

### 3.1. Protocol

#### 3.1.1. Command & Acknowledge Mechanism

The protocol uses a command/acknowledge pattern. On each channel each entity has a defined role of being either a “command sender and acknowledge receiver” or a “command receiver and acknowledge sender”. It is defined in the BRM which channel acts as a command channel from the host to the device, and which channel is used for the opposite direction from the device to the host. The command sender sends a command and waits for the acknowledge packet. The command receiver receives the command, acts according to the command, and sends the acknowledge packet with the result.

The communication on the default communication channel defines the role of an entity. The sender of a command on the default communication channel is called the host. The command receiver on the default communication channel is called the (remote) device.

A command packet contains a number called `command_id`, which specifies the action to be executed by the receiver and some additional data to be used when executing the command. The command receiver is expected to process the command and return the result to the sender of the command using an acknowledge packet.

All commands on a channel are sent sequentially. This means that after a command has been sent, the command sender must wait for an acknowledge or wait for a timeout and process the failure before the next command may be sent.

Each command is sent with a sequentially incremented request id. This id allows resending a command in case of a failure. A successful communication would follow this schema:

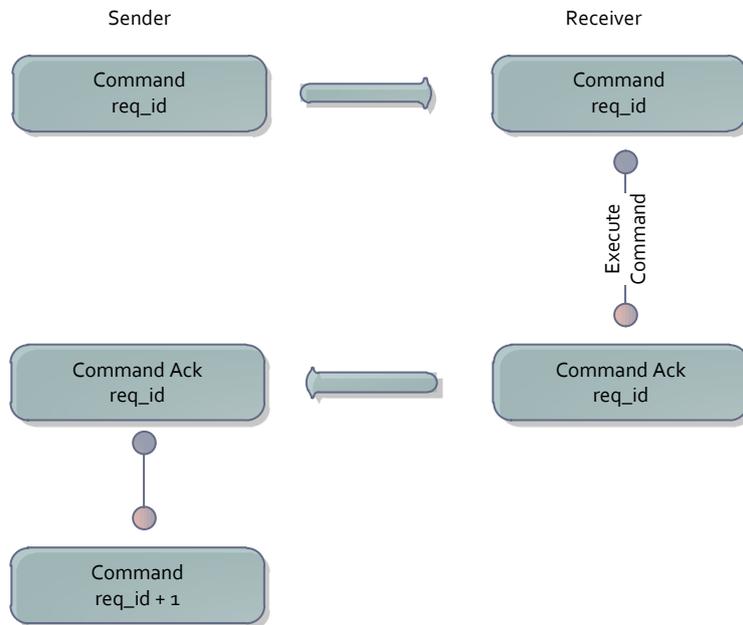


Fig. 1 - Command Cycle

One entity, such as the host, sends a command with a given request\_id to the other entity, such as the device, on a channel. The device processes the command, forms an acknowledge packet and sends that back to the host. Command and acknowledge must have the same request\_id. After the completion of a cycle, a different request\_id for the next cycle must be used. It is up to the implementation to pick its request\_id. By default an increment of 1 is recommended.

The round trip time for a command and the according acknowledge is

$$\text{Command Transfer Time} + \text{Processing Time} + \text{Acknowledge Transfer time}$$

When calculating the timeout time for the command cycle, a host must therefore consider:

- the transfer time of the maximum packet size on a given link speed
- the Maximum Device Response Time, which is provided via a bootstrap register
- some margin for technology-dependent delays, which may occur on the link

Reading the Maximum Device Response Time (MDRT) register should not exceed 50 ms in order to guarantee a responsive device. The maximum device response time for any other read or write operation should not exceed 300 ms. This plus the maximum packet transfer time allows the host to calculate a timeout value.

### 3.1.2. Pending Acknowledge

In case the processing of a command takes longer than specified in the Maximum Device Response Time register, the command receiver must send a pending acknowledge. This pending acknowledge response uses the same request\_id as the command which triggered it and provides a temporary timeout in ms to be used only with the command currently executed. The command sender can then temporarily adjust its acknowledge timeout for the current cycle. In case the command receiver has the heartbeat enabled it has to suspend its heartbeat mechanism so that the device does not lose connection. In case the execution of the command takes longer than signaled through an already sent pending acknowledge, the command receiver may issue another pending acknowledge indicating a new, longer timeout.

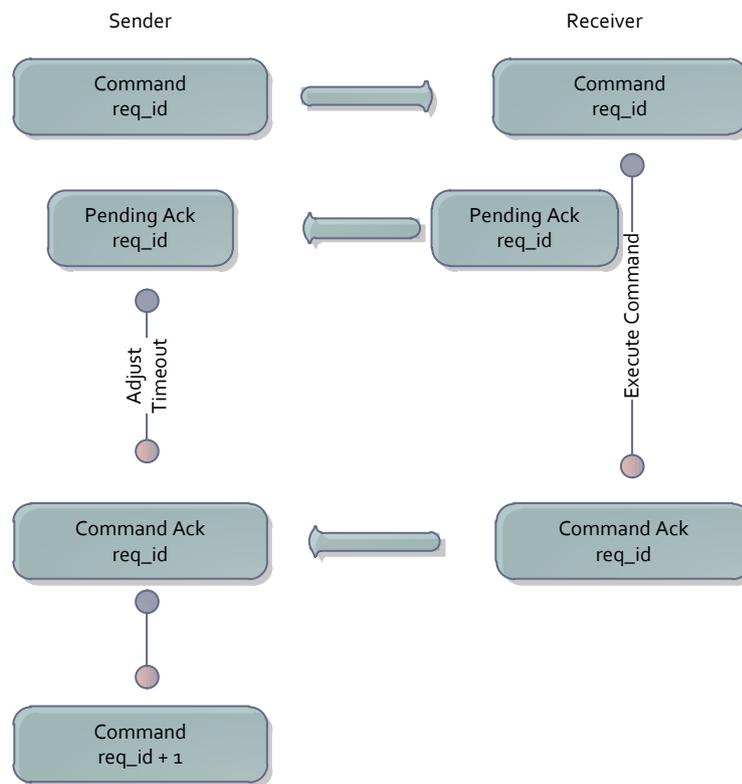


Fig. 2 – Pending Ack Cycle

In case the device receives a further command packet while processing a command, it reacts as follows:

- If the new command has the same request\_id as the command currently processed, another pending acknowledge packet is sent. In this case the pending ack timeout from the original command is used.
- If the new command has a different request\_id the device responds with a GENCP\_BUSY status code.

The Processing Time for the inquiry of the Maximum Device Response Time register must not take longer than 50ms.

After the cycle finishes, the host timeout resets to the previously calculated timeout using Maximum Device Response Time and the heartbeat mechanism in the device works as configured before.

### 3.1.3. Message Channel

A Message Channel allows the asynchronous transfer of event commands from the device to the host. For each Message Channel a different channel\_id from the default channel must be used. The

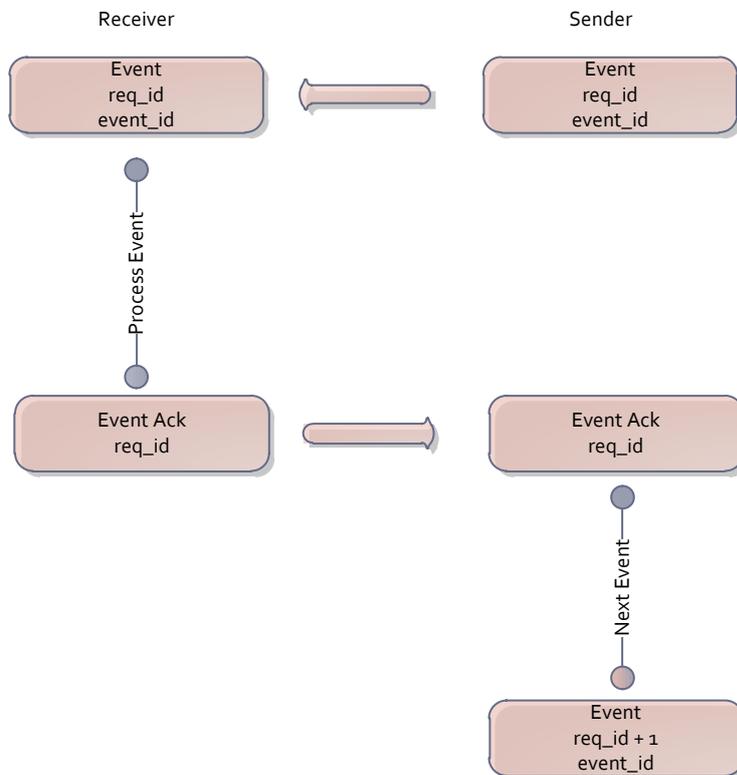


Fig. 3 – Event Cycle

channel\_id to be used by the Message Channel is set by the host in the according register in the device's BRM. An Event is identified by an event\_id. Subsequently sent events are identified by request\_ids. One entity, such as the device, sends an event with a given request\_id to the other entity, such as the host, on a channel. The host acknowledges the event by sending an EventAck packet back to the device. The event and the corresponding acknowledge must have the same request\_id. After the completion of a cycle, a different request\_id for the next cycle must be used. It is up to the implementation to pick its request\_id. By default an increment of 1 is recommended.

### 3.1.3.1. Event ID

The source of an event on the Message Channel is identified by an event\_id. An event\_id is a 16-bit value. The bits in this value have the following meaning:

Bit offset (lsb << x)	Width (bits)	Description
0	12	<b>Event ID</b>
12	2	<b>Reserved</b> Set to 0
14	2	<b>Namespace</b> 0 = GenCP Event ID 1 = Technology specific Event ID 2 = Device specific Event ID

Table 2 - Event ID

### 3.1.3.2. GenCP Event ID Codes

Event ID (Hex)	Name	Description
0x0000	Error	Generic Error Event

Table 3 – GenCP Event IDs

### 3.1.4. Failure

A failure on the Command Channel or the Message Channel is discovered through

- a corrupt CRC of either the command or the acknowledge packet
- a timeout waiting for an acknowledge
- an invalid (too short) packet (timeout waiting for the complete arrival)

- an incorrect packet header

### **3.1.4.1. Corrupt Packet**

In case of a corrupt CRC or in case of a wrong packet length the received data is discarded. The receive buffer should be flushed until no data is received within a maximum packet transfer time or longer.

- The sender must wait after a communication error until all corrupt data is removed and then it sends its command again.
- The receiver discards all corrupt data after a communication error and waits for the sender to resend its command.
- If the underlying technology controls packet handling, it is not necessary to wait for a packet transfer time on failure.

When there are errors on either side, the original command packet is resent from the sender as described in chapter 3.1.4.3.

In case of failure the sender should retry 3 times to transmit the packet.

### **3.1.4.2. Timeout**

A packet is considered “too short” if the data for a packet has not completely been received within the Packet Transfer Time (PTT) after the first byte of the packet has arrived. The PTT is depending on

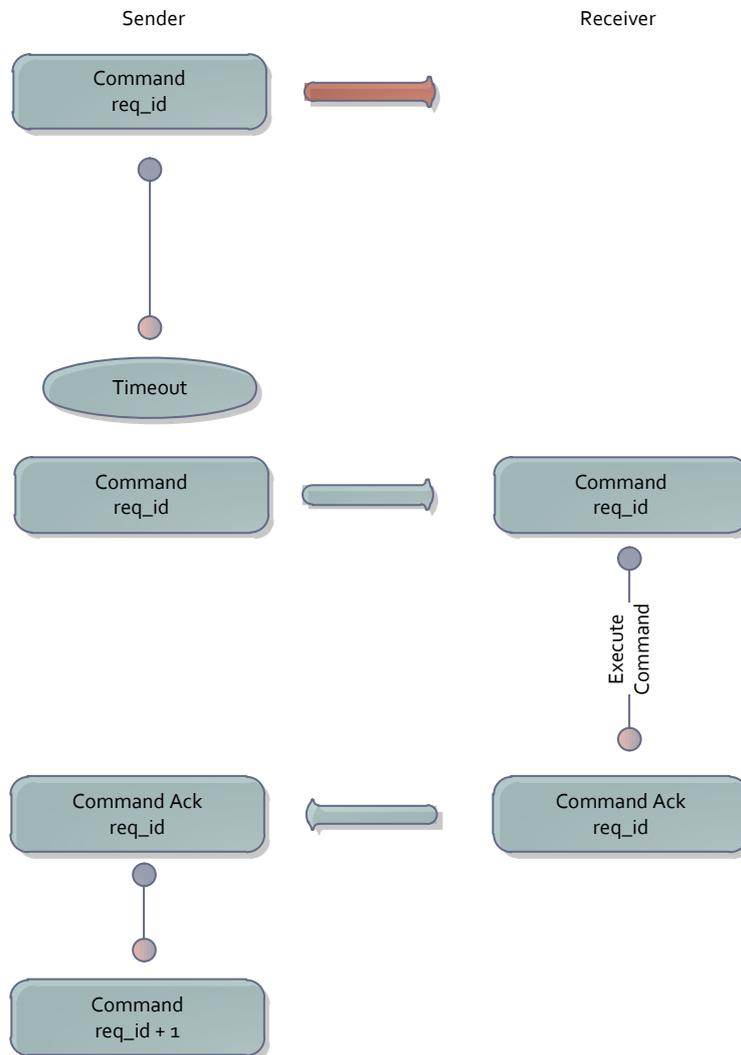
- the link speed
- the maximum PacketSize allowed on the link
- the timeout for the transfer of two consecutive bytes on a link

When there are errors on either side, the original command packet is resent from the sender as described in chapter 3.1.4.3 .

In case of failure the sender should retry 3 times to transmit the packet.

### 3.1.4.3. Command Packet Failure

If the command packet is lost on the link or if the command packet is received as corrupt the following actions are supposed to happen:



### 3.1.4.4. Acknowledge packet failure

If an acknowledge packet is lost on the link or if the CRC of the received packet is corrupt, the following actions are supposed to happen:

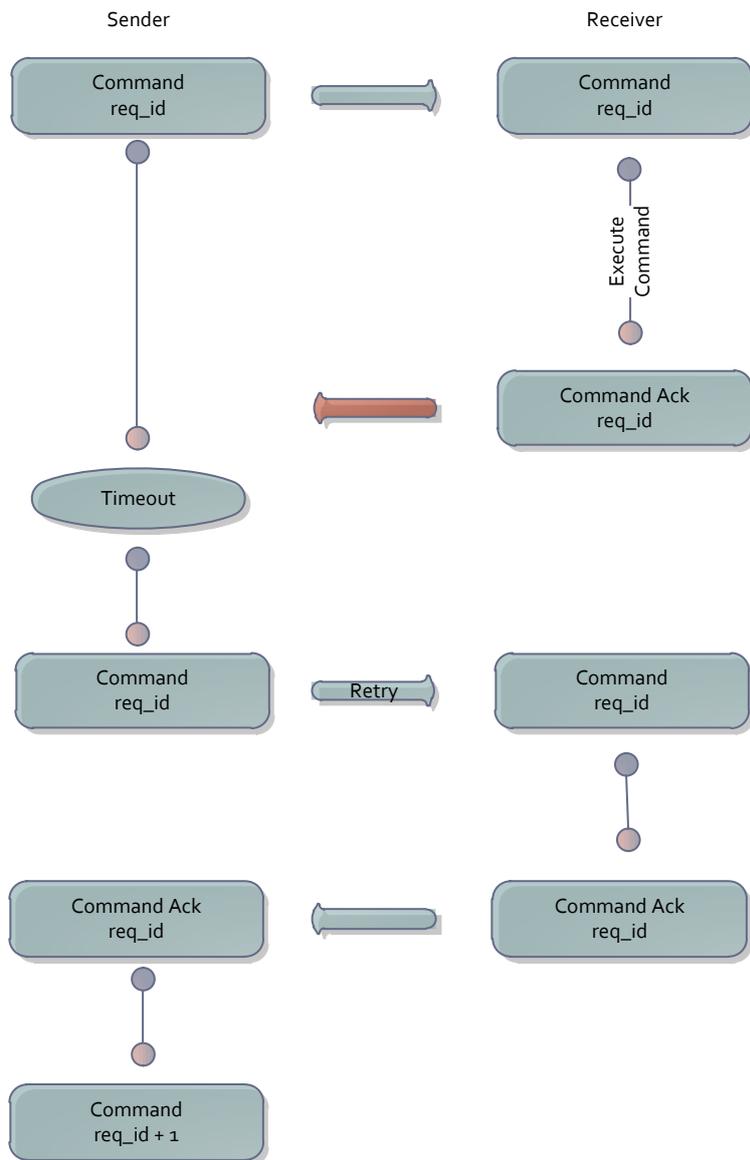


Fig. 5 – Ack Failure

The resend of the command packet uses the same request\_id as the original. This allows the receiver to identify a resend in case the request\_id is already processed. In this case the command must not be processed again but the previous result should be resent.

In case of a corrupt acknowledge packet the sender may issue the command resend immediately without waiting for the timeout.

### **3.1.4.5. Pending Acknowledge Packet Failure**

There are two possible failure cases using pending acknowledge.

- A complete pending acknowledge packet is lost. In this case the sender will generate a timeout as if the pending acknowledge would not have been sent and it will issue a resend of the command packet with the same request\_id. Following chapter 3.1.2, the receiver will reissue a pending acknowledge packet.
- A pending acknowledge packet is received corrupt by the sender. This will trigger a resend of the command packet.

## **3.2. Heartbeat**

In order to maintain control in case of an unexpected abrupt detach of the controlling application, a watchdog timer is implemented in the device. This mechanism is called Heartbeat. On start-up of the command sender application, the Access Privilege Register in the device's BRM must be set. With that the Heartbeat timer in the device starts. This Heartbeat timer has to be triggered periodically by a read/write register access from the host to the device. The timeout of the Heartbeat can be adjusted through a register in the bootstrap register map. The presence of a Heartbeat mechanism is indicated by a bit in the device capability register in the device's BRM. It may be disabled through a bit in the device configuration register in the BRM.

In case the Heartbeat counter is not triggered by a register access longer than what is specified in the Heartbeat Timeout register, the device stops streaming and resets the access privilege status.

The Access Privilege register can be set to

- Available – The device is available. The device does not stream data.
- Open (Exclusive) – Only the controlling application has read and write access to the device. It is depending on the technology how this is observed. Other applications/hosts will receive an error trying to access the device's register map. The exception to this rule is the Access Privilege register itself. This register can be read any time.

### **3.3. GenICam File**

A GenCP device must be register based. A manufacturer must provide access to a GenICam file describing the register map of the device.

The GenICam file must be stored within the device so that it can be retrieved by the host. The file may be stored and delivered either in uncompressed or compressed format. In case it is compressed it is up to the controlling host to deflate the file.

#### **3.3.1. Manifest Table**

A GenCP device may provide multiple GenICam files complying with different GenICam Schema versions. A so called “Manifest Table” register block contains a list of entries, providing information like file versions, complying schema versions, and register addresses. A description of the Manifest register block can be found in the Bootstrap Register Map section of this document.

#### **3.3.2. Retrieval**

It is the responsibility of the host software to retrieve the file from the device reading the device’s register space using the GenCP Protocol.

### **3.4. Serial**

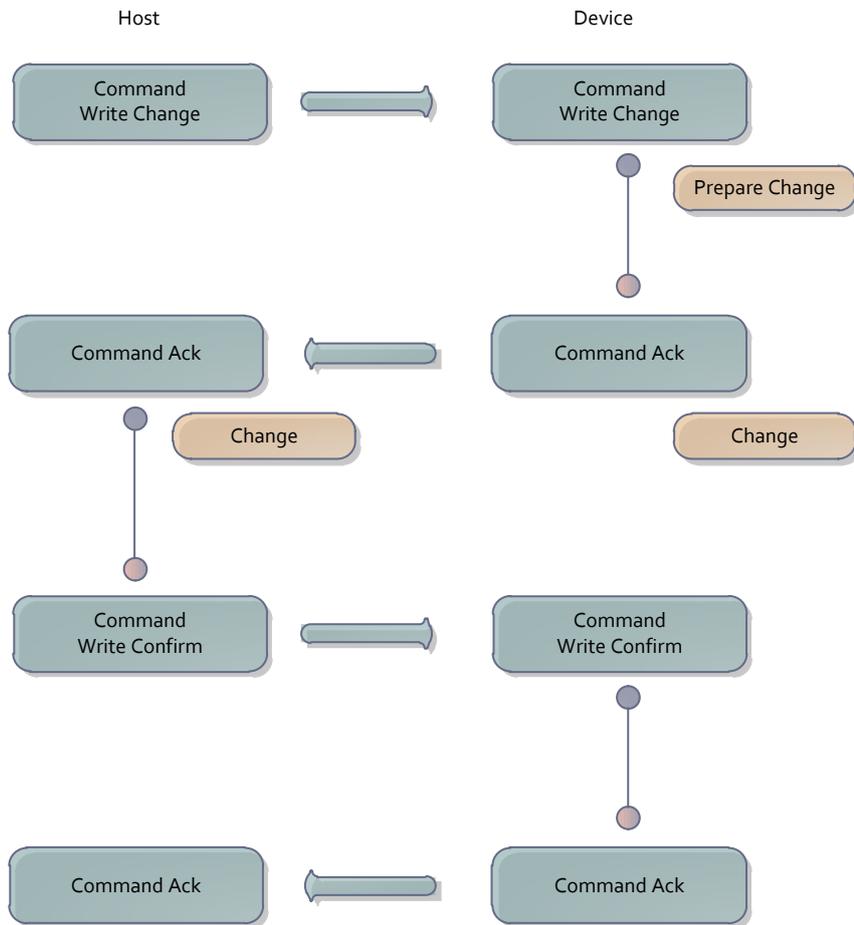
#### **3.4.1. Packet Size**

In order to maintain reasonable response times even with low link speeds the packets must not exceed 1024 Bytes per packet.

#### **3.4.2. Serial Parameters**

The link uses 8Bit, No Parity, 1 Stop Bit encoding and 9600 Baud per default. The Link can be switched to other comm. parameters and/or higher baud rates after a communication has been established.

When switching to other communication parameters the procedure is as follows:



The confirmation command rewrites the register which was written in the change step.

In case the device does not receive the confirming write command with the new parameters within 250 ms after sending the acknowledge it falls back to the original parameter set.

In case the write confirm fails the host must wait for 500 ms and then retry using the original parameter set.

## 4. Packet Layout

The protocol defines the communication between two entities. An entity is either a device or a host. The role of a device and host are defined by the initiator of the default communication. The host is the initiator of the communication on the default channel (see chapter 2.8) and the device responds to that.

### 4.1. General Packet Layout

The generic packet layout is divided into four parts:



Fig. 7 – General Packet Layout

- Prefix describes a technology specific section of the packet. This section covers
  - Addressing
  - Protocol type identification
  - CRC
  - channel\_id etc.

If compared to UDP/IP a Prefix would be omitted since everything is covered by the underlying protocol. For CL we would not need to cover addressing because it is a serial connection, but we need to identify a communication channel (by `channel_id`) and we need a CRC and we need a preamble to identify the protocol.

- The Common Command Data section contains data which describes the command. For example this section contains the actual command identifier and the `request_id`.
- The Command Specific Data section is technology agnostic. It carries data which is specific for a given command. For example, for a read command it would contain the address to read from and the number of bytes to read.
- The Postfix section is again technology specific. It carries for example a CRC Checksum in case it is needed for a given technology. This section is only mandatory if defined for a given technology.

## 4.2. Prefix

In case the underlying technology does not provide an addressing schema for multiple communication channels or does not provide a checksum mechanism the protocol needs to provide such services. A packet then contains not only command specific data but also has to mimic an addressing scheme between the device and host. Also we need to be able to support multiple communication channels on a given Link and a checksum.

In case such services are provided by the underlying technology the Prefix can simply be omitted.

### 4.2.1. Serial

For a serial connection we do not have to handle addressing between device and host because it is a point to point connection but we do need to mimic multiple communication channels. In addition a packet preamble allows to identify a GenCP packet and differentiate it from other (ASCII based) protocols.

For the default communication channel the `channel_id` is always 0.

Width (Bytes)	Offset (Bytes)	Description
2	0	<b>0x0100 (preamble)</b> Leading binary 0x1 (SOH) 0x00 (NULL) send on the link to identify a GenCP package to allow the application layers above to distinguish between different protocols.
2	2	<b>CCD-CRC-16</b> CRC-16 build from the <code>channel_id</code> and CCD
2	4	<b>SCD-CRC-16</b> CRC-16 build from <code>channel_id</code> , CCD, SCD and Postfix
2	6	<b>channel_id</b> A 16 bit number identifying a communication channel. Channel 0 is reserved the for the default communication channel.

Table 4 - Serial Prefix

This prefix layout is identical for command and acknowledge.

The Checksum is the 16-bit one's complement of the one's complement sum of the whole packet including preamble. The computation algorithm is the same as for the UDP checksum referenced in RFC 768.

### 4.3. Common Command Data

The Common Command Data section is technology agnostic.

#### 4.3.1. Command Packet Layout

Width (Bytes)	Offset (Bytes)	Description												
Prefix														
2	0	<p><b>flags</b></p> <p>Flags to enable/disable command options or to provide additional info on the specific command.</p> <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 10px;"> <thead> <tr style="background-color: #f2f2f2;"> <th style="width: 25%;">Bit offset (lsb &lt;&lt; x)</th> <th style="width: 15%;">Width (bits)</th> <th style="width: 60%;">Description</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">14</td> <td>Reserved, set to 0</td> </tr> <tr> <td style="text-align: center;">14</td> <td style="text-align: center;">1</td> <td><b>RequestAck</b> If set the sender requests an acknowledge packet from the command receiver.</td> </tr> <tr> <td style="text-align: center;">15</td> <td style="text-align: center;">1</td> <td><b>CommandResend</b> If set the command is sent as a retry of a previous sent that failed.</td> </tr> </tbody> </table>	Bit offset (lsb << x)	Width (bits)	Description	0	14	Reserved, set to 0	14	1	<b>RequestAck</b> If set the sender requests an acknowledge packet from the command receiver.	15	1	<b>CommandResend</b> If set the command is sent as a retry of a previous sent that failed.
Bit offset (lsb << x)	Width (bits)	Description												
0	14	Reserved, set to 0												
14	1	<b>RequestAck</b> If set the sender requests an acknowledge packet from the command receiver.												
15	1	<b>CommandResend</b> If set the command is sent as a retry of a previous sent that failed.												
2	2	<p><b>command_id</b></p> <p>Command id as specified in the Command ID chapter 4.3.3</p>												
2	4	<p><b>length</b></p> <p>Length of the Specific Command Data depending on the command ID not including Prefix, Postfix and CCD</p>												
2	6	<p><b>request_id</b></p> <p>Sequential number to identify a single command. This id is provided by the command sender and incremented every time a new command is issued.</p>												
SCD														
Postfix														

Table 5 - Common Command Data

### 4.3.2. Acknowledge Packet Layout

Width (Bytes)	Offset (Bytes)	Description															
Prefix																	
2	0	<p><b>status code</b></p> <p>Status code, indicating the result of the operation.</p> <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Bit offset (lsb &lt;&lt; x)</th> <th>Width (bits)</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>12</td> <td><b>Status Code</b></td> </tr> <tr> <td>12</td> <td>1</td> <td><b>Reserved</b> Set to 0</td> </tr> <tr> <td>13</td> <td>2</td> <td><b>Namespace</b> 0 = GenCP Status Code 1 = Technology specific Code 2 = Device specific Code</td> </tr> <tr> <td>15</td> <td>1</td> <td><b>Severity</b> 0 = Warning/Info 1 = Error</td> </tr> </tbody> </table> <p>See chapter 4.3.2.1 for a list of codes.</p>	Bit offset (lsb << x)	Width (bits)	Description	0	12	<b>Status Code</b>	12	1	<b>Reserved</b> Set to 0	13	2	<b>Namespace</b> 0 = GenCP Status Code 1 = Technology specific Code 2 = Device specific Code	15	1	<b>Severity</b> 0 = Warning/Info 1 = Error
Bit offset (lsb << x)	Width (bits)	Description															
0	12	<b>Status Code</b>															
12	1	<b>Reserved</b> Set to 0															
13	2	<b>Namespace</b> 0 = GenCP Status Code 1 = Technology specific Code 2 = Device specific Code															
15	1	<b>Severity</b> 0 = Warning/Info 1 = Error															
2	2	<p><b>command_id</b></p> <p>Command id as specified in the command_id chapter 4.3.3</p>															
2	4	<p><b>length</b></p> <p>Length of the Specific Command Data depending on the command in bytes.</p>															
2	6	<p><b>request_id</b></p> <p>Sequential number used to identify a single acknowledge. This id is provided by the command sender and incremented every time a new command is issued.</p>															
SCD																	
Postfix																	

Table 6 - Acknowledge layout

### 4.3.2.1. Status Codes

This section lists status codes that can be returned through an acknowledge packet. Each status code has 16 bits.

Status Code (Hex)	Name	Description
0x0000	GENCP_SUCCESS	Success
0x8001	GENCP_NOT_IMPLEMENTED	Command not implemented in the device.
0x8002	GENCP_INVALID_PARAMETER	At least one command parameter of CCD or SCD is invalid or out of range.
0x8003	GENCP_INVALID_ADDRESS	Attempt to access a not existing register address.
0x8004	GENCP_WRITE_PROTECT	Attempt to write to a read only register.
0x8005	GENCP_BAD_ALIGNMENT	Attempt to access registers with an address which is not aligned according to the underlying technology.
0x8006	GENCP_ACCESS_DENIED	Attempt to read a non-readable or write a non-writable register address.
0x8007	GENCP_BUSY	The command receiver is currently busy.
0x800B	GENCP_MSG_TIMEOUT	Timeout waiting for an acknowledge.
0x800E	GENCP_INVALID_HEADER	The header of the received command is invalid. This includes CCD and SCD fields but not the command payload.
0x800F	GENCP_WRONG_CONFIG	The current receiver configuration does not allow the execution of the sent command.
...		
0x8FFF	GENCP_ERROR	Generic error.

Table 7 – Status Codes

### 4.3.3. *Command IDs*

This chapter describes the `command_ids` available for the command field in the Common Command Data section of a GenCP command packet.

Command Name	command_id
READMEM_CMD	Hex 0800
READMEM_ACK	Hex 0801
WRITEMEM_CMD	Hex 0802
WRITEMEM_ACK	Hex 0803
PENDING_ACK	Hex 0805
EVENT_CMD	Hex 0C00
EVENT_ACK	Hex 0C01

Table 8 – Command Identifier

## 4.4. Command Specific Data

### 4.4.1. ReadMem Command

Start address and length of any read access is byte aligned unless the underlying technology states different rules.

Width (Bytes)	Offset (Bytes)	Description
Prefix		
CCD (command_id = READMEM_CMD)		
8	0	<b>register address</b> 64 bit register address.
2	8	<b>reserved</b> Reserved, set to 0
2	10	<b>read length</b> Number of bytes to read.
Postfix		

Table 9 - ReadMem SCD-Fields

### 4.4.2. ReadMem Acknowledge

Width (Bytes)	Offset (Bytes)	Description
Prefix		
CCD-ACK (command_id = READMEM_ACK)		
x	0	<b>Data</b> Data read from the remote device's register map. If the number of bytes read is different than the specified in the relating READMEM_CMD the status of the READMEM_ACK must indicate the reason.
Postfix		

Table 10 - ReadMem Ack SCD-Fields

### 4.4.3. WriteMem Command

Any write access start address and length is byte aligned unless the underlying technology states different rules. The number of bytes to write is deduced through the length field of the CCD header.

Width (Bytes)	Offset (Bytes)	Description
Prefix		
CCD (command_id = WRITEMEM_CMD)		
8	0	<b>register address</b> 64 bit register address.
x	8	<b>data</b> Number of bytes to write to the remote device's register map.
Postfix		

Table 11 - WriteMem Command SCD-Fields

### 4.4.4. WriteMem Acknowledge

The WriteMem acknowledge states the result of a WriteMem command.

Width (Bytes)	Offset (Bytes)	Description
Prefix		
CCD-ACK (command_id = WRITEMEM_ACK)		
2	0	<b>reserved</b> This reserved field is only sent if the length_written field is sent with the acknowledge. If it is sent it is to be set to 0.
2	2	<b>length written</b> Number of bytes successfully written to the remote device's register map. The length written field must only be sent if the according bit in the Device Capability register is set.
Postfix		

Table 12 - WriteMem Ack SCD-Fields

The length field in CCD section of the WriteMem Ack must be set to 0 or 4 depending on the bit in

the Device Capability register. In case the write\_length field (and the 2 reserved bytes) is sent, the length field is to be set to 4. In case the length\_written field is not sent the length field is 0.

#### 4.4.5. Pending Acknowledge

The pending acknowledge informs the sender that the command, sent with the given request\_id, needs more time to execute than stated in the MDRT register. This allows the temporary adjustment of the timeout mechanism on the command sender side. This “new” temporary timeout is only valid for the command referenced by request\_id.

Width (Bytes)	Offset (Bytes)	Description
Prefix		
CCD-ACK (command_id = PENDING_ACK)		
2	0	<b>reserved</b> Reserved, set to 0.
2	2	<b>temporary timeout</b> Temporary timeout for the command sent with the given request_id. The timeout is specified in ms.
Postfix		

Table 13 - Pending Ack SCD-Fields

#### 4.4.6. **Event Command**

Width (Bytes)	Offset (Bytes)	Description
Prefix		
CCD (command_id = EVENT_CMD)		
2	0	<b>reserved</b> Reserved, set to 0
2	2	<b>event_id</b> The event_id is a number identifying an event source. The schema of the event_id follows the description in chapter 3.1.3.1
8	4	<b>timestamp</b> 64 bit timestamp value in ns as defined in the timestamp bootstrap register.
X	12	<b>data</b> Optional event specific data.
Postfix		

Table 14 - Event Command SCD-Fields

#### 4.4.7. **Event Acknowledge**

Width (Bytes)	Offset (Bytes)	Description
Prefix		
CCD-ACK (command_id = EVENT_ACK)		
Postfix		

Table 15 - Event Acknowledge SCD-Fields

### 4.5. **Postfix**

The Postfix carries data like a CRC in case the underlying protocol layers do not provide such services. The Postfix is conditional mandatory depending on the technology.

#### 4.5.1. ***Serial***

We do not need a Postfix section for serial links.

## 5. Bootstrap Register Map

### 5.1. *Technology Agnostic Bootstrap Register Map*

The Technology Agnostic Bootstrap Register Map (ABRM) uses the first 64 Kbytes of the register space. The table below shows the layout of the technology agnostic part of that bootstrap register map. This part also contains pointers to various other parts like the Manifest which provides access to the device GenICam files or the technology specific bootstrap registers.

### 5.2. *String Registers*

String registers not fully used are to be filled with 0. In case the full register is used the terminating 0 can be omitted. The encoding of the content of a string register must match the Device Capability register.

### 5.3. *Conditional Mandatory Registers*

Conditional Mandatory (CM) registers are registers which may or may not be implemented depending on the Device Capability register. Access to a CM register which is indicated as being not available will return a GENCP\_INVALID\_ADDRESS status code.

## 5.4. Register Map

Width (Bytes)	Offset (Bytes)	Support	Access	Description
4	0x00000	M	R	<b>GenCP Version</b> Complying GenCP specification Version
64	0x00004	M	R	<b>Manufacturer Name</b> String containing the self-describing name of the manufacturer
64	0x00044	M	R	<b>Model Name</b> String containing the self-describing name of the device model
64	0x00084	CM	R	<b>Family Name</b> String containing the name of the family of this device
64	0x000C4	M	R	<b>Device Version</b> String containing the version of this device
64	0x00104	M	R	<b>Manufacturer Info</b> String containing additional manufacturer information
64	0x00144	M	R	<b>Serial Number</b> String containing the serial number of the device
64	0x00184	CM	RW	<b>User Defined Name</b> String containing the user defined name of the device
8	0x001C4	M	R	<b>Device Capability</b> Bit field describing the device's capabilities
4	0x001CC	M	R	<b>Maximum Device Response Time</b> Maximum response time in ms
8	0x001D0	M	R	<b>Manifest Table Address</b> Pointer to the Manifest Table
8	0x001D8	CM	R	<b>SBRM Address</b> Pointer to the Technology Specific Bootstrap Register Map
8	0x001E0	M	RW	<b>Device Configuration</b> Bit field describing the device's configuration

Width (Bytes)	Offset (Bytes)	Support	Access	Description
4	0x001E8	CM	RW	<b>Heartbeat Timeout</b> Heartbeat Timeout in ms
4	0x001EC	CM	RW	<b>Message Channel ID</b> channel_id used for the message channel
8	0x001F0	CM	R	<b>Timestamp</b> Last latched device time in ns
4	0x001F8	CM	W	<b>Timestamp Latch</b>
8	0x001FC	CM	R	<b>Timestamp Increment</b>
4	0x00204	CM	RW	<b>Access Privilege</b>
4	0x00208	CM	R	<b>Protocol Endianess</b> Endianess of protocol fields and bootstrap registers.
4	0x0020C	CM	R	<b>Implementation Endianess</b> Endianess of device implementation registers
65008	0x00210	M	no	Reserved Register Space

Table 16 - Technology agnostic BRM

- Width            Size of the register in bytes.
- Offset:        Address of the register (Offset in Bytes) in the device's BRM
- Support        M=Mandatory/R=Recommended/ CM=Conditional Mandatory (depending on the capability bits)
- Access         R=READONLY, W=WRITEONLY, RW=READWRITE
- Description   Name and Very short hint on the meaning

### 5.4.1. *GenCP Version*

Version of the GenCP specification this Bootstrap Register Map complies with.

Offset	Hex 0
Length	4
Access Type	R
Support	M
Data Type	2 x 16bit fields
Factory Default	Implementation specific

Bit offset (lsb << x)	Width (bits)	Description
0	16	<b>Minor Version</b> Minor Version of the Standard this BRM and the protocol the device's implementation complies to.
16	16	<b>Major Version</b> Major Version of the Standard this BRM and the protocol the device's implementation complies to.

Table 17 - Register GenCP Version

### 5.4.2. *Manufacturer Name*

Manufacturer Name is a string containing a human readable manufacturer name.

Offset	Hex 4
Length	64
Access Type	R
Support	M
Data Type	String
Factory Default	Device specific

### 5.4.3. **Model Name**

The register contains a string with a human readable model name.

Offset	Hex 44
Length	64
Access Type	R
Support	M
Data Type	String
Factory Default	Device specific

### 5.4.4. **Family Name**

Family Name is a string containing a human readable name referring to multiple (similar) models of a single manufacturer.

Offset	Hex 84
Length	64
Access Type	R
Support	CM
Data Type	String
Factory Default	Device specific

### 5.4.5. **Device Version**

A string containing a Device Version.

Offset	Hex C4
Length	64
Access Type	R
Support	M
Data Type	String
Factory Default	Device specific

### 5.4.6. **Manufacturer Info**

Manufacturer info is a string containing manufacturer specific information. If there is none, this field should be all 0.

Offset	Hex 104
Length	64
Access Type	R
Support	M
Data Type	String
Factory Default	Device specific

### 5.4.7. **Serial Number**

The register contains a string representing the serial number of the device.

Offset	Hex 144
Length	64
Access Type	R
Support	M
Data Type	String
Factory Default	Device specific

### 5.4.8. **User Defined Name**

A string containing a user defined name. This Register is conditional mandatory depending on the Device Capabilities register. A write to this register must instantly persist without explicitly being stored to non-volatile memory.

Offset	Hex 184
Length	64
Access Type	RW
Support	CM
Data Type	String
Factory Default	Empty String

### 5.4.9. Device Capability

Device capability bits describe implementation specific details.

Offset	Hex 1C4
Length	8
Access Type	R
Support	M
Data Type	Bitfield
Factory Default	Implementation specific

Bit offset (lsb << x)	Width (bits)	Description
0	1	<b>User Name Available</b> Set if User Defined Name available.
1	1	<b>Access Privilege Available</b> Set if Heartbeat/Access Privilege is available.
2	1	<b>Message Channel Supported</b> Set if the device supports a Message Channel.
3	1	<b>Timestamp Supported</b> Set if the device supports a timestamp register.
4	4	<b>String Encoding</b> String Encoding of the BRM <ul style="list-style-type: none"> <li>- 0x0 -&gt; ASCII</li> <li>- 0x1 -&gt; UTF8</li> <li>- 0x2 -&gt; UTF16</li> <li>- 0x3-0xF -&gt; Reserved</li> </ul>
8	1	<b>Family Register Available</b> Set if the device supports the Family Name register.
9	1	<b>SBRM Supported</b> Set if the device supports a SBRM.
10	1	<b>Endianness Registers Supported</b> Set if the device supports the Protocol Endianness and Implementation Endianness registers.
11	1	<b>Written Length Field Supported</b> Set to 1 if the device sends the length_written field in the SCD section of the WriteMemAck command.
12	52	<b>Reserved</b> Set to 0.

Table 18 - Register Device Capabilities

### 5.4.10. **Maximum Device Response Time (MDRT)**

Integer value containing the maximum time in milliseconds until a device reacts upon a received command. This is not including the time needed to receive the command or send the acknowledge but only the time needed to execute the command. In case a device needs longer to process a command it must send a pending ack back.

The maximum time needed to transfer the message is depending on the link speed and the maximum size of the message.

This number may have direct impact on the behavior of software layers above. It is to be kept as short as possible.

The maximum response time must not exceed 300 ms in order to guarantee a good device's behavior.

Reading this register must not exceed 50 ms processing time.

Offset	Hex 1CC
Length	4
Access Type	R
Support	M
Data Type	UINT32
Factory Default	Implementation Specific

Bit offset (lsb << x)	Width (bits)	Description
0	32	<b>Maximum Device Response Time</b> Maximum time until a device sends a response upon a received command not including the time needed to send the response over the link in ms.

Table 19 - Register Maximum Device Response Time

### 5.4.11. **Manifest Table Address**

Pointer to the Manifest table containing the URLs for the GenICam files for this device. (See chapter 5.6.1)

Offset	Hex 1D0
Length	8
Access Type	R
Support	M
Data Type	UINT64
Factory Default	Implementation specific

Bit offset (lsb << x)	Width (bits)	Description
0	64	<b>Manifest Table Address</b> 64-bit register address of the Manifest Table

Table 20 - Register Manifest Table Offset

### 5.4.12. **SBRM Address**

The register contains a pointer to the Technology Specific Bootstrap Register Map.

Offset	Hex 1D8
Length	8
Access Type	R
Support	CM
Data Type	UINT64
Factory Default	Implementation Specific

Bit offset (lsb << x)	Width (bits)	Description
0	64	<b>SBRM Address</b> Technology Specific Bootstrap Register Map Address

Table 21 - Register Technology Specific Bootstrap Register Map

### 5.4.13. **Device Configuration**

Device Configuration bits describing implementation specific details.

Offset	Hex 1E0
Length	8
Access Type	RW
Support	M
Data Type	Bitfield
Factory Default	Device specific

Bit offset (lsb << x)	Width (bits)	Description
0	1	<b>Heartbeat Enable</b> Set to enable the Heartbeat Timer.
1	63	<b>Reserved</b> Set to 0.

Table 22 - Register Device Configuration

### 5.4.14. **Heartbeat Timeout**

The register is available if the Access Privilege Available bit in the Device Capability register is set.

Offset	Hex 1E8
Length	4
Access Type	RW
Support	CM
Data Type	UINT32
Factory Default	3000

Bit offset (lsb << x)	Width (bits)	Description
0	32	<b>Heartbeat Timeout</b> Heartbeat timeout in ms.

Table 23 - Register Heartbeat Timeout

### 5.4.15. **Message Channel ID**

The register contains the `channel_id` to be used for the message channel. This register has to be written by the host to tell the device which channel to use for the message channel. At start up the register contains 0 indicating that it is not initialized by the host. A `channel_id` of 0 for the Message Channel is not valid since 0 is used for the command channel.

Offset	Hex 1EC
Length	4
Access Type	RW
Support	CM
Data Type	UINT32
Factory Default	0

Bit offset (lsb << x)	Width (bits)	Description
0	32	<b>Channel ID</b> Message Channel ID.

Table 24 - Register Message Channel ID

This register is present if the Message Channel bit in the Device Capability register is set.

### 5.4.16. *Timestamp*

A read of this register provides a timestamp of a free running, device internal clock in ns. Before reading the timestamp register must be latched to the device's internal clock by writing to the Timestamp Latch register.

Offset	Hex 1F0
Length	8
Access Type	R
Support	CM
Data Type	UINT64
Factory Default	0

Bit offset (lsb << x)	Width (bits)	Description
0	64	<b>Timestamp</b> Device Time in ns.

Table 25 - Register Timestamp

The Timestamp bit in the Device Capability register indicates if this register is present or not.

### 5.4.17. *Timestamp Latch*

A write with the Timestamp Latch bit set to 1 latches the current device time into the timestamp register.

Offset	Hex 1F8
Length	4
Access Type	W
Support	CM
Data Type	Bitfield
Factory Default	-

Bit offset (lsb << x)	Width (bits)	Description
0	1	<b>Timestamp Latch</b> Latch the current device time into the timestamp register. The bit is self clearing which means that you do not need to set it to 0.
1	31	<b>Reserved</b> Set to 0.

Table 26 - Register Timestamp Latch

The Timestamp bit in the Device Capability register indicates if this register is present or not. This register must be supported if the Timestamp register is supported.

### 5.4.18. *Timestamp Increment*

This register indicates the ns/tick of the device internal clock. This allows the application to deduce the accuracy of the timestamp provided by the bootstrap register. For example a value of 1000 indicates the device clock runs at 1MHz.

Offset	Hex 1FC
Length	8
Access Type	R
Support	CM
Data Type	UINT64
Factory Default	Device specific

Bit offset (lsb << x)	Width (bits)	Description
0	64	<b>Timestamp Increment</b> Timestamp increment in ns/tick.

Table 27 - Register Timestamp Increment

The Timestamp bit in the Device Capability register indicates if this register is present or not. This register must be supported if the Timestamp register is supported.

### 5.4.19. **Access Privilege**

This register reflects the current access privilege.

Offset	Hex 204
Length	4
Access Type	RW
Support	CM
Data Type	Bitfield
Factory Default	0

Bit offset (lsb << x)	Width (bits)	Description
0	3	<b>Access Privilege</b> Current Access Privilege as described in 3.2 0 = Available 1 = Open (Exclusive) 2-7 = reserved
3	29	<b>Reserved</b> Set to 0.

Table 28 - Register Access Privilege

This register is available if the Access Privilege Available bit in the Device Capability register is set.

### 5.4.20. **Protocol Endianness**

This register reflects the endianness of the protocol and the bootstrap registers. By reading the register the host can detect the endianness of the protocol fields and the bootstrap registers.

Offset	Hex 208
Length	4
Access Type	R
Support	CM
Data Type	UINT32
Factory Default	Device specific

Bit offset (lsb << x)	Width (bits)	Description
0	32	<b>Protocol Endianness</b> Endianness of the protocol implementation. 0 = big-endian 0xFFFFFFFF = little-endian

Table 29 - Register - Protocol Endianness

This register is available if the Endianness Registers Supported bit in the Device Capability register is set.

### 5.4.21. **Implementation Endianess**

This register reflects the endianess of the device implementation. By reading the register the host can detect the endianess of the device specific registers.

Offset	Hex 20C
Length	4
Access Type	R
Support	CM
Data Type	UINT32
Factory Default	Device specific

Bit offset (lsb << x)	Width (bits)	Description
0	32	<b>Implementation Endianess</b> Endianess of the device implementation. 0 = big-endian 0xFFFFFFFF = little-endian

Table 30 - Register - Implementation Endianess

This register is available if the Endianess Registers Supported bit in the Device Capability register is set.

## 5.5. **Technology Specific Bootstrap Register Map**

### 5.5.1. **Serial**

Width (Bytes)	Offset (Bytes)	Support	Access	Description
4	0	M	R	<b>Supported Baudrates</b>
4	4	M	(R)W	<b>Current Baudrate</b>

Table 31 - Serial BRM

### 5.5.1.1. Supported Baudrate

Bitfield indicating the supported baudrates.

Offset	Hex 000
Length	4
Access Type	R
Support	M
Data Type	Bitfield
Factory Default	0

Bit offset (lsb << x)	Width (bits)	Description
0	32	<p><b>Supported Baudrate</b></p> <p>BAUDRATE_9600 = 0x00000001            BAUDRATE_19200 = 0x00000002            BAUDRATE_38400 = 0x00000004            BAUDRATE_57600 = 0x00000008            BAUDRATE_115200 = 0x00000010            BAUDRATE_230400 = 0x00000020            BAUDRATE_460800 = 0x00000040            BAUDRATE_921600 = 0x00000080</p> <p>Multiple bits may be set according to the capability of the device.</p>

Table 32 - Register – Serial – Supported Baudrates

### 5.5.1.2. Current Baudrate

Register indicating the currently used baudrate. The register is RW with the exception that only one baud rate is supported. In this case the register may also be read only.

Offset	Hex 004
Length	4
Access Type	RW
Support	M
Data Type	Bitfield
Factory Default	0

Bit offset (lsb << x)	Width (bits)	Description
0	32	<p><b>Current Baudrate</b></p> <p style="margin-left: 40px;"> BAUDRATE_9600 = 0x00000001  BAUDRATE_19200 = 0x00000002    BAUDRATE_38400 = 0x00000004  BAUDRATE_57600 = 0x00000008  BAUDRATE_115200 = 0x00000010  BAUDRATE_230400 = 0x00000020  BAUDRATE_460800 = 0x00000040  BAUDRATE_921600 = 0x00000080 </p> <p>A single bit may be set according to the current baudrate setting.</p>

Table 33 - Register – Serial – Current Baudrate

### 5.5.2. USB3 Vision

The USB3 Vision specific registers are described in the USB3 Vision specification

## 5.6. Generic Tables

### 5.6.1. Manifest

The manifest provides a way to store multiple GenICam files in the device. These GenICam files may be available in different versions, in various formats or comply to different versions of the GenICam schema. The manifest table contains a list of Manifest Entries.

#### 5.6.1.1. Manifest Table

Width (Bytes)	Offset (Bytes)	Support	Access	Description
8	0	M	R	<b>MT Entry Count</b> Number of entries in the Manifest Table
64	8	M	R	<b>Manifest Entry 0</b> First entry in the Manifest Table
64	8 + 64	O	R	<b>Manifest Entry 1</b> Second entry in the Manifest Table
...	...			...
64	8 + n*64	O	R	<b>Manifest Entry n</b> (N+1)th entry in the Manifest Table

Table 34 – Manifest Table Layout

### 5.6.1.2. Manifest Entry

Each Manifest Entry describes the properties of a single file.

Width (Bytes)	Offset (Bytes)	Description													
4	0	<b>GenICam File Version</b>													
		<table border="1"> <thead> <tr> <th>Bit offset (lsb &lt;&lt; x)</th> <th>Width (bits)</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>16</td> <td><b>File-Subminor Version</b> Subminor version of the GenICam file referenced in this entry.</td> </tr> <tr> <td>16</td> <td>8</td> <td><b>File-Minor Version</b> Minor version of the GenICam file referenced in this entry.</td> </tr> <tr> <td>24</td> <td>8</td> <td><b>File-Major Version</b> Major version of the GenICam file referenced in this entry.</td> </tr> </tbody> </table>	Bit offset (lsb << x)	Width (bits)	Description	0	16	<b>File-Subminor Version</b> Subminor version of the GenICam file referenced in this entry.	16	8	<b>File-Minor Version</b> Minor version of the GenICam file referenced in this entry.	24	8	<b>File-Major Version</b> Major version of the GenICam file referenced in this entry.	
		Bit offset (lsb << x)	Width (bits)	Description											
		0	16	<b>File-Subminor Version</b> Subminor version of the GenICam file referenced in this entry.											
16	8	<b>File-Minor Version</b> Minor version of the GenICam file referenced in this entry.													
24	8	<b>File-Major Version</b> Major version of the GenICam file referenced in this entry.													
4	4	<b>Schema / Filetype</b>													
<table border="1"> <thead> <tr> <th>Bit offset (lsb &lt;&lt; x)</th> <th>Width (bits)</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>10</td> <td><b>Reserved</b> Set to 0.</td> </tr> <tr> <td>10</td> <td>6</td> <td><b>Filetype</b> File type of the file this entry points to. 0 = Uncompressed GenICam XML file 1 = ZIP containing a single GenICam XML file 2-63 = reserved</td> </tr> <tr> <td>16</td> <td>8</td> <td><b>Schema-Minor Version</b> Minor Version of the GenICam Schema the GenICam file complies with.</td> </tr> <tr> <td>24</td> <td>8</td> <td><b>Schema-Major Version</b> Major Version of the GenICam Schema the GenICam file complies with.</td> </tr> </tbody> </table>	Bit offset (lsb << x)	Width (bits)	Description	0	10	<b>Reserved</b> Set to 0.	10	6	<b>Filetype</b> File type of the file this entry points to. 0 = Uncompressed GenICam XML file 1 = ZIP containing a single GenICam XML file 2-63 = reserved	16	8	<b>Schema-Minor Version</b> Minor Version of the GenICam Schema the GenICam file complies with.	24	8	<b>Schema-Major Version</b> Major Version of the GenICam Schema the GenICam file complies with.
Bit offset (lsb << x)	Width (bits)	Description													
0	10	<b>Reserved</b> Set to 0.													
10	6	<b>Filetype</b> File type of the file this entry points to. 0 = Uncompressed GenICam XML file 1 = ZIP containing a single GenICam XML file 2-63 = reserved													
16	8	<b>Schema-Minor Version</b> Minor Version of the GenICam Schema the GenICam file complies with.													
24	8	<b>Schema-Major Version</b> Major Version of the GenICam Schema the GenICam file complies with.													
8	8	<b>Register Address</b>													

		64Bit Register Address at which the file can be read from.
8	16	<b>FileSize</b> Size of the file this manifest entry points to in bytes.
20	24	<b>SHA1-Hash</b> SHA1 Hash of the file or 0 in case the hash is not available.
20	44	<b>Reserved</b> Set to 0.

Table 35 - Manifest Entry Layout