GigE Vision

Video Streaming and Device Control over Ethernet Standard

version 1.2

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Published by

Automated Imaging Association
900 Victors Way, Suite 140
Ann Arbor, MI 48108

Table of Content

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

List of Figures

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

List of Tables

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

1 Introduction

1.1 Purpose

GigE Vision is a communication interface for vision applications based on the ubiquitous Ethernet technology. It allows for easy interfacing between the GigE Vision device and a network card using standard CAT-5 cable or any other physical medium supported by Ethernet.

The aim of this specification is to define the protocols used by GigE Vision compliant products to communicate with each other. The specification builds upon the Ethernet technology. As such, it does not cover the physical description of the transport media. Nor is any specific implementation of specialized high-performance IP stack or network driver part of the specification.

GigE Vision systems cover a wide spectrum of different network topologies. The simplest one is a point-to-point connection between a PC and a GigE video streaming device using a crossover cable; a more complex one is a video distribution system comprised of hardware and software video sources, sinks and processing units on an IP network using routers. It is crucial that GigE Vision products are fully compatible with other IP devices on the network. Therefore, it is mandatory to follow the guidelines provided by the RFC of the IETF. But at the same time, the number of mandatory requirements should be limited to allow for economical realization of GigE Vision devices.

Even though the name of this specification specifically refers to Gigabit Ethernet, it can be implemented for any Ethernet speed grade.

1.2 Technical Committee

The GigE Vision Standard Committee was formed in June 2003 to define an open transport platform based on GigE (Gigabit Ethernet) for high-performance vision applications. GigE is the high-speed, 1-Gb/s (gigabit-per-second) version of Ethernet, the world's dominant LAN connection protocol.

The committee is sponsored by the AIA (Automated Imaging Association, http://www.machinevisiononline.org/). The following member companies have participated over the years to the GigE Vision Technical Committee.

- Basler AG
- Baumer Optronic
- Coreco (acquired by DALSA Corp.)
- DALSA Corp.
- e2v semiconductors
- Euresys
- GigaLinx
- JAI A/S and JAI PULNiX
- KAPPA opto-electronics
- Leutron Vision AG
- Matrox Electronic Systems
- MVTec Software
- National Instruments
- Pleora Technologies
- Sensor to Image
- STEMMER IMAGING
- Toshiba TELI Corp.

1.3 Definitions and Acronyms

1.3.1 Definitions

Application	GigE Vision control application software running on a host. Typically a software application running on a PC but can also be of another nature such as micro code running on a field programmable gate array (FPGA).
Control protocol	GigE Vision control protocol (GVCP) defining commands supported by GigE Vision compliant devices.
Device	GigE Vision compliant controllable device. Typically a camera but can also be of another nature such as a software video server running on a PC.
Event Generator	Entity generating event messages according the GigE Vision specification.
Event Receiver	Entity receiving and capable of interpreting event messages according the GigE Vision specification.
Gratuitous ARP	An ARP request sent by the device to check if another device is using the same IP address.
GVSP Transmitter	Entity producing a stream of data according to the GigE Vision Streaming Protocol.
GVSP Receiver	Entity receiving and capable of de-encapsulating a stream of data according to the GigE Vision Streaming Protocol.
Management Entity	Application responsible for the configuration and control of devices deployed in a system.

Multihomed	A host is said to be multihomed if it has multiple IP addresses.
Persistent IP	A persistent IP address is hard-coded in non-volatile memory of the device. It is re-used by the device on power-cycle when Persistent IP is enabled.
Primary application	Application having exclusive or control access (read/write) to the device. This includes control access with switchover enabled.
Primary application switchover request	The condition under which another application is trying to take control over a device that is under the control of a primary application that was granted control access with switchover enabled.
Secondary application	Application having monitoring access (read-only) to the device.
Standard GVCP port	UDP port used on a device to receive GVCP commands.
Server	Server that provides IP server functionality (DHCP) used to assign IP addresses. The server can optionally be included in the application, but most of the time it is separated.
Static IP	A static IP address is set by the application into the device volatile memory. It is lost on power-cycle. Static IP is mainly used by an application to "force" an IP address into a device (for instance, when an invalid Persistent IP is memorized by the device).

1.3.2 Requirements Terminology

This specification uses the following convention to list requirements.

Table 1-1: Requirements Terminology

Term	Description	Representation
Absolute Requirement	Feature that MUST be supported by the product. It is mandatory to support the feature to ensure interoperability.	[Rx-y<suffix>]
Conditional Requirement	Feature that MUST be supported IF another feature is present. It is mandatory to support the feature when another feature is supported.	[CRx-y<suffix>]
Absolute Objective	Feature that SHOULD be supported by the product. It is recommended, but not essential.	[Ox-y<suffix>]
Conditional Objective	Feature the SHOULD be supported IF another feature is present.	[COx-y<suffix>]

Each requirement is represented by a unique number in brackets. Each number is composed of up to 4 elements:

1. Requirement Type: Absolute **R**equirement, **C**onditional **R**equirement, Absolute **O**bjective, **C**onditional **O**bjective

2. Section number: Specification section number
3. Sequence number: Unique number in a section (same list used for Absolute Requirement, Conditional Requirement, Absolute Objective and Conditional Objective)
4. Suffix: Identifies if the requirement or objective is applicable to application software, devices, GVSP transmitters, GVSP receivers or some combination of these according to the terminology presented in Table 1-2.

Table 1-2: Suffix Terminology

Suffix	Description
ca	Represents a requirement or objective exclusive to application software.
cd	Represents a requirement or objective exclusive to devices.
c	Represents a requirement or objective applicable to both applications and devices.
st	Represents a requirement or objective exclusive to GVSP transmitters.
sr	Represents a requirement or objective exclusive to GVSP receivers.
s	Represents a requirement or objective applicable to GVSP transmitters and receivers.
No suffix	Represents a requirement or objective applicable to applications, devices, GVSP transmitters and receivers.

For instance, [R4-6cd] is requirement 6 in section 4 of this specification that only applies to GigE Vision devices.

A compliancy matrix is provided at the end of this document to summarize which requirements are applicable to a given GigE Vision product.

1.3.3 Acronyms

AOI	Area of Interest
BOOTP	Bootstrap Protocol
CCP	Control Channel Privilege (a bootstrap register)
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
GigE	Gigabit Ethernet
GVCP	GigE Vision Control Protocol

GVSP	GigE Vision Streaming Protocol
IANA	Internet Assigned Numbers Authority
IP	Internet Protocol
IPv4	Internet Protocol version 4
LFSR	Linear Feedback Shift Register
LLA	Link-Local Address
LSB	Least Significant Bit/Byte
MAC	Media Access Control
MSB	Most Significant Bit/Byte
MTU	Maximum Transmission Unit
NAT	Network Address Translation
OUI	Organizationally Unique Identifier
PC	Personal Computer
RFC	Request For Comments
ROI	Region of Interest
SPI	Stateful Packet Inspection
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1.4 Reference Documents

IEEE Standards	
IEEE 802-2001	IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture
IEEE 802.3-2002	IEEE Standard for Information technology--Telecommunications and information exchange between systems--Local and metropolitan area networks--Specific requirements--Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications
Internet Standard	
STD3 (RFC1122 + RFC1123)	Requirements for Internet Hosts - Communication Layers + Requirements for Internet Hosts - Application and Support
STD5 (RFC791 + RFC792 + RFC919 + RFC922 + RFC950 + RFC1112)	INTERNET PROTOCOL + Internet Control Message Protocol + Broadcasting Internet Datagrams + Broadcasting Internet datagrams in the presence of subnets + Internet Standard Subnetting Procedure + Host extensions for IP multicasting

STD6 (RFC768)	User Datagram Protocol
STD13 (RFC1034 + RFC1035)	Domain Concepts and Facilities Domain Implementation and Specification
STD33 (RFC1350)	THE TFTP PROTOCOL (REVISION 2)
STD37 (RFC826)	Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses To 48.Bit Ethernet Address For Transmission On Ethernet Hardware
STD41 (RFC894)	Standard For The Transmission Of IP Datagrams Over Ethernet Networks
STD43 (RFC1042)	Standard For The Transmission Of IP Datagrams Over IEEE 802 Networks
Internet RFC	
RFC951	Bootstrap Protocol
RFC1534	Interoperation Between DHCP and BOOTP
RFC1542	Clarifications and Extensions for the Bootstrap Protocol
RFC1951	DEFLATE Compressed Data Format Specification version 1.3
RFC2131	Dynamic Host Configuration Protocol
RFC2132	DHCP Options and BOOTP Vendor Extensions
RFC3171	IANA Guidelines for IPv4 Multicast Address Assignments
RFC3376	Internet Group Management Protocol, Version 3
RFC3927	Dynamic Configuration of IPv4 Link-Local Addresses
Others	
GenICam	Generic Interface for Camera device, EMVA, version 1.1.2.

1.5 System Overview

As mentioned in the previous section, GigE Vision systems cover a wide spectrum of different network topologies. The simplest one is a point-to-point connection between a PC and a GigE video streaming device using a crossover cable or even over an Ethernet network while a more complex one is a video distribution system comprised of hardware and software video sources, sinks, processing units and peripherals on an IP network.

Figure 1-1 demonstrates the traditional point-to-point camera to host control and receiving application use case as defined in the previous versions of this specification while Figure 1-2 demonstrates a more complex use case.

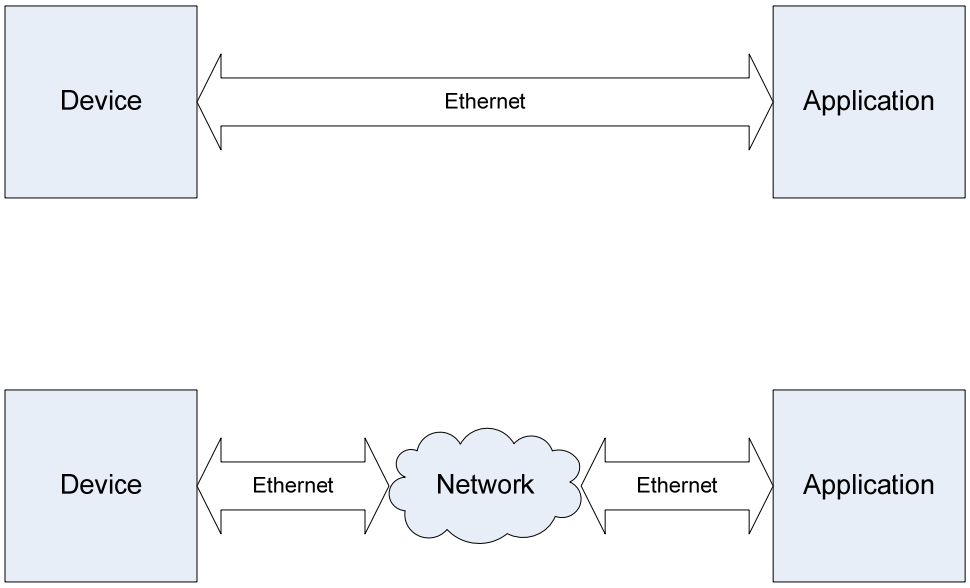


Figure 1-1: Traditional streaming device to control and receiving application

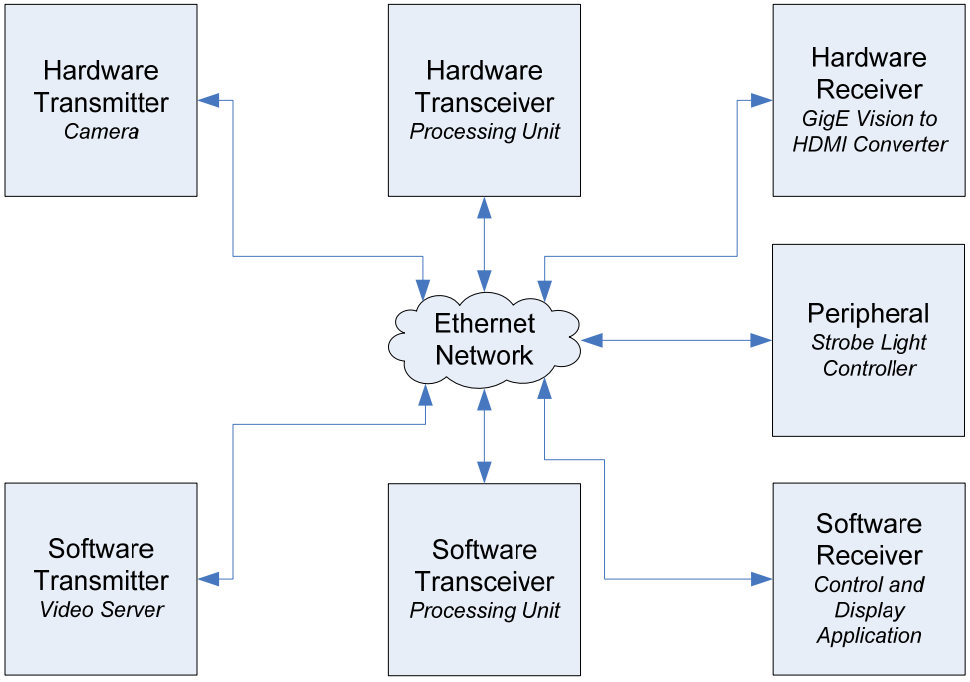


Figure 1-2: Advanced Video over Ethernet Distribution System

Although, the GigE Vision protocol could be leveraged effectively and successfully in more complex systems, the previous versions of this specification limited the class of products that could claim GigE Vision compliance. That is because the requirements terminology being used in the previous versions of this specification focused more or less on the use case where a device would be a streaming device transmitting video data to a control and receiving application.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

This specification removes this limitation by simply decoupling control and streaming in the requirements terminology. In other words, a device can be transmitting and/or receiving data while the opposite holds true for an application.

Therefore, this specification defines new classes of products and generalizes concepts.

The main two new concepts are:

1. A product must be an application, a device or both.
2. A product doesn't need to have a stream channel.

In addition to this, the following device classes are defined.

1. ***Transmitter***: A transmitter is a device capable of streaming data. It includes one or more GVSP transmitters. This can be a camera.
2. ***Receiver***: A receiver is a device capable of receiving data. It includes one or more GVSP receivers. This can be a GigE Vision to HDMI converter.
3. ***Transceiver***: A transceiver is a device capable of receiving and transmitting data. It includes one or more GVSP receivers and one or more GVSP transmitters. This can be a GigE Vision image processor.
4. ***Peripheral***: A peripheral is a device that cannot transmit or receive data. However, it can execute tasks controlled using the GigE Vision control protocol. This can be a GigE Vision strobe light controller.

This specification also dictates that if a product is to be controlled over Ethernet, it must be possible to do so using the GigE Vision control protocol. This is in order to insure interoperability amongst products.

It should be noted that a product can act as the control application of more than one device. However, the parameters of a product can only be controlled by a single control application at a given point in time. This limitation is due to the standard GVCP port. Please refer to Section 7.1 for more details.

This version of the specification also maintains the streaming devices to control and receiving applications backward compliance. However, given the fact that new type of products are introduced, one cannot expect forward compatibility if using a legacy product with a newly defined one. For instance, an application created to handle streaming devices in the past may have some difficulties to cope with a newly defined peripheral device if the application tries to instantiate a stream channel on the device. Nevertheless, as mentioned above, backward compliance for the traditional point-to-point camera to host control and receiving application is maintained.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

PART 1 – Device Discovery

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

2 Device Discovery Summary

2.1 Overview

Device Discovery covers the sequence of events required for a controllable device to obtain a valid IP address using standard IP protocols, and for a control application to enumerate devices on the network. Once Device Discovery is completed, an application is ready to send control requests to a device.

> **Note**: By definition, a **device** is a GigE Vision compliant **controllable device** and an application is a GigE Vision compliant **control application** software running on a host. Therefore, the terms **device** and **application** are used in the remainder of this document is order to simplify the standard text.

Device Discovery is separated in 2 sequential steps:

1. IP configuration
2. Device enumeration

The first step, IP configuration, uses standard IP protocols. It is initiated by the device. Its goal is to assign a unique IP address to the device. GigE Vision devices must support DHCP and Link-Local address. GigE Vision devices may support Persistent IP. Persistent IP is defined as static IP address that persists across power cycle or reset. It is kept in non-volatile storage in the device.

The second step, device enumeration, is initiated by the application to gather information about device presence on the network. If the application does not know the device IP address, this can be achieved on the application's subnet using a UDP broadcast command. If the client's IP address is known, this can be accomplished using unicast UDP. This step is realized using GVCP message exchange between the device and the application. A device that receives an enumeration request should always answer it, even if it does not have a valid IP address. This answer incorporates various pieces of information about the device, such as the manufacturer, device model, etc.

> **Warning**: This specification does not identify how an application can retrieve an IP address of a device residing on a different subnet. This could be achieved using information provided by the DHCP server, by multi-casting a device enumeration request or using DNS. A future version of the specification may address this topic.

2.2 Goals

The following design goals are set as a target for IP Configuration and Device Discovery.

1. It should have distinctive steps for IP configuration and for device enumeration.
2. It must assign a valid IP address to the GigE Vision device.
3. It must enumerate all GigE Vision devices on the same subnet as the GigE Vision application.
4. It should offer provisions to find GigE Vision devices on a different subnet than the application.
5. Where possible, it should provide a plug-and-play solution that does not require any networking configuration.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

6. It should ensure devices are reachable within 20 seconds after being powered-up or reset.
7. It should provide device specific information to GigE Vision application to uniquely identify devices (including model and manufacturer).
8. It should leverage on existing IP protocols whenever possible to ensure interoperability with IP networks and minimize development efforts.
9. It should optionally allow the user to force a persistent IP address to a device. This address would be used on all subsequent re-initialization of the GigE Vision device.
10. It should minimize IP stack complexity in the GigE Vision device. It should try to push the complexity on the application side where CPU power and memory are more readily available.

At the end of the Device Discovery process, the application must have identified all GigE Vision compliant devices on its subnet and each device must be ready to receive requests.

2.3 Scope

This part covers GigE Vision Device Discovery. This includes IP configuration and device enumeration.

Other protocols used to communicate between a device and application software (such as GVCP) are covered in other section of this specification.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

3 IP Configuration

This section lists the requirements a device has to follow to obtain a valid IP address.

[R3-1cd] All devices MUST execute IP configuration upon power-up or device reset.

[R3-2cd] A device MUST support the following IP configuration protocols:

- DHCP
- Link-Local Address (LLA)

Optionally, a device may support a user configurable Persistent IP address. This address is stored in the device non-volatile memory (bootstrap registers) to be used on device power-up or reset. When a persistent IP address is used, it is up to the user to ensure the selected IP address will not cause any conflict on the network.

In this section, a valid IP address represents the combination of:

1. An IP address
2. A corresponding subnet mask
3. A default gateway

This specification asks for the IP address to be assigned in a reasonable amount of time (ideally in less than 20 seconds in the worst case). The user expects the device to be up and running within seconds after powering the unit. This could be difficult to achieve if hundreds of devices are powered-up simultaneously, thus overloading the DHCP server. In order to limit the IP configuration time, each configuration scheme, except link-local address, can optionally be disabled. Note that the application might have its own restriction to obtain an IP address: some operating systems might take up to 1 min. to assign a LLA address to a network interface. The connection between a device and an application can only be realized when they both have a valid IP address on the same subnet.

Note: GigE Vision does not require BOOTP and RARP. A particular implementation might elect to support BOOTP and/or RARP.

This section explains how the device selects which IP configuration protocol to use and how specific options of these protocols can be enabled.

[CR3-3cd] If a device supports multiple network interfaces, then each interface MUST perform IP configuration independently.

[CR3-4cd] If a device supports multiple network interfaces, then a different set of IP configuration registers MUST be available for each interface. These registers are part of the bootstrap registers.

This allows each network interface to be configured using a different IP configuration protocol.

3.1 Protocol Selection

Each device has to iterate through a selection of IP configuration scheme.

[R3-5cd] On a device, the sequence of execution of each IP configuration protocol MUST be

1. Persistent IP (if supported and enabled)
2. DHCP (if enabled)
3. Link-Local Address

Note: The above requirement can be interpreted in 2 ways which are both acceptable.

The first interpretation has the DHCP client stopped when the state machine moves from DHCP stage to Link-Local Address. This is a strict sequential approach which better matches figure 3-1, but contradicts a recommendation of RFC3927.

The second interpretation keeps the DHCP client running according the recommendation of RFC3927. Therefore, even if a LLA address was assigned, the DHCP client can still obtain an address from a DHCP server. This interpretation is in-line with the behavior of typical operating systems, such as Windows and Linux. Note that when this situation happens, the device will replace its LLA IP address with the newly obtained DHCP IP address.

[R3-6cd] Factory defaults of a device MUST have Persistent IP disabled and DHCP enabled. LLA is always enabled.

This ensures a standardized way to access a new device, even when devices from different manufacturers are put on the same network segment.

Note that GVCP also offers a command to force a static IP address into a device. This address overrides the IP address obtained using the normal IP configuration selection described in this section. But this address is lost on power cycle or device reset.

The procedure to select the IP assignation protocol is illustrated below.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

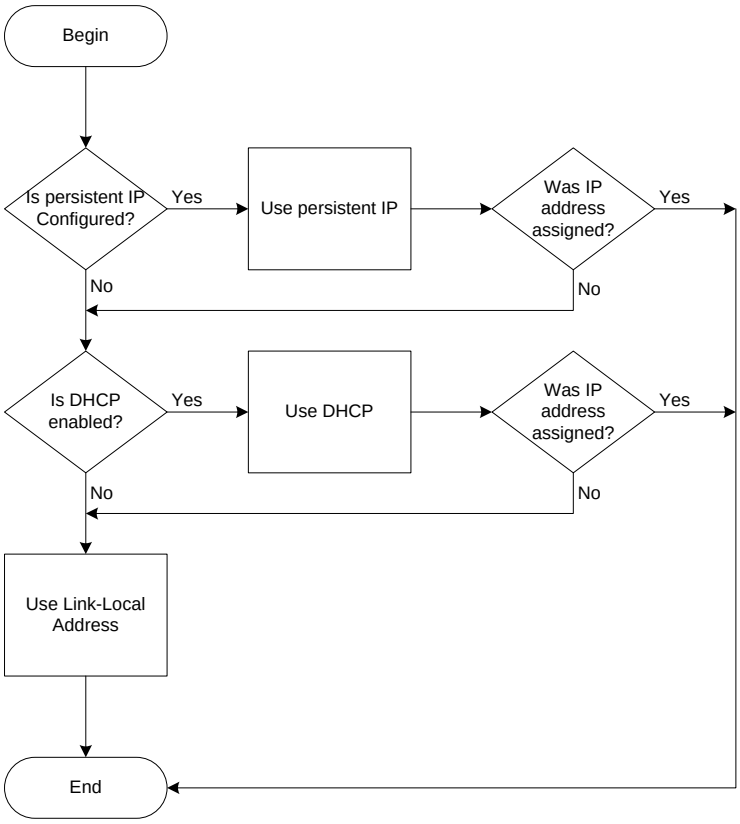


Figure 3-1: Protocol Selection Flowchart

[R3-7cd] Once a valid IP address is assigned to the device, it MUST be copied into the following bootstrap registers:

1. Current IP address (address 0x0024 for first network interface)
2. Current subnet mask (address 0x0034 for first network interface)
3. Current default gateway (address 0x0044 for first network interface)

3.2 Persistent IP

A device may elect to support Persistent IP on any of its network interface.

[CR3-8cd] If a device supports Persistent IP, then it MUST do so using the appropriate bootstrap registers and it MUST provide some non-volatile memory to store those settings.

The following bootstrap registers are used to support Persistent IP:

1. Supported IP configuration (address 0x0010 for first network interface): bit 31 indicates if Persistent IP is supported by this device (bit set) or not (bit cleared).
2. Current IP configuration procedure (address 0x0014 for first network interface): bit 31 indicates if Persistent IP has been activated (bit set) or not (bit cleared) by the user.

3. Persistent IP address (address 0x064C for first network interface): this is the Persistent IP address assigned by the user. It is up to the user to ensure this is a valid IP address.
4. Persistent IP subnet mask (address 0x065C for first network interface): this is the subnet mask associated with the Persistent IP address.
5. Persistent default gateway (address 0x066C for first network interface): this is the default gateway when Persistent IP is activated.

[CR3-9cd] If a device supports Persistent IP, then the Persistent IP address, its associated subnet mask and its default gateway MUST be stored in non-volatile memory of the device. This is true for all network interfaces supporting Persistent IP.

If a device does not have non-volatile memory, then it cannot support Persistent IP.

It is up to the end-user to correctly assign the persistent IP address of the device and to ensure it does not create a conflict on the network. RFCs relevant to IP address assignment should be respected to avoid IP configuration problems.

Note: It is not permitted to have a Persistent IP address of 0.0.0.0 or 255.255.255.255 when the interface is configured. If such a configuration is set, then the device moves to the next valid IP configuration method.

~~[CO3-10d] If a device supports Persistent IP, then it SHOULD validate the Persistent IP address class against its subnet mask. If such a validation is performed, and if the subnet mask does not match the IP address class, then the device MUST move to the next IP configuration scheme.~~

[CO3-11cd] If a device supports Persistent IP, then it SHOULD ARP for the Persistent IP address before using it in order to detect a potential address conflict (this is often referred to as "gratuitous ARP"). If such a conflict is detected, the device MUST NOT use this IP address and SHOULD signal this problem to the user (for example, by flashing a status LED). The way the device signals this problem is left as a quality of implementation. The device MUST then move to the next IP configuration scheme.

When an unidentified IP address is assigned through Persistent IP, it is possible for an application to gain control of the device using the FORCEIP_CMD message of GVCP. The application can then modify the Persistent IP information of the bootstrap registers to a valid state or simply disable Persistent IP.

3.3 DHCP

DHCP is a very well known scheme to obtain an IP address. All network interfaces of a device are to support DHCP.

[R3-12cd] A device MUST support DHCP. Refer to RFC2131 (Dynamic Host Configuration Protocol) and RFC2132 (DHCP Options and BOOTP Vendor Extensions) for more information on DHCP.

[O3-13cd] A device SHOULD implement a DHCP enable flag in non-volatile storage to indicate if DHCP should be enabled on next device reset. This flag is part of the Current IP Configuration Procedure bootstrap register (one per network interface).

[CR3-14cd] If this enable flag is supported and is set to TRUE, then the device MUST try to obtain an IP address using DHCP.

[CR3-15cd] If this enable flag is supported and is set to FALSE, then the device MUST moves to the next IP configuration protocol given by the protocol selection flowchart without using DHCP.

[CR3-16cd] If this non-volatile storage is not present, then the device MUST react as if DHCP was enabled (the DHCP enable flag is hard-coded to TRUE and is not user configurable).

This DHCP enable flag is located in the bootstrap registers. The following bootstrap registers are used to support DHCP:

1. Supported IP configuration (address 0x0010 for first network interface): bit 30 indicates if DHCP is supported by this device (bit set) or not (bit cleared). This bit is always 1.
2. Current IP configuration procedure (address 0x0014 for first network interface): bit 30 indicates if DHCP has been activated (bit set) or not (bit cleared) by the user.

The following figure illustrates a DHCP message. The way this message is constructed is given by RFC2131. This specification provides additional information on how specific fields can be used by GigE Vision devices.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

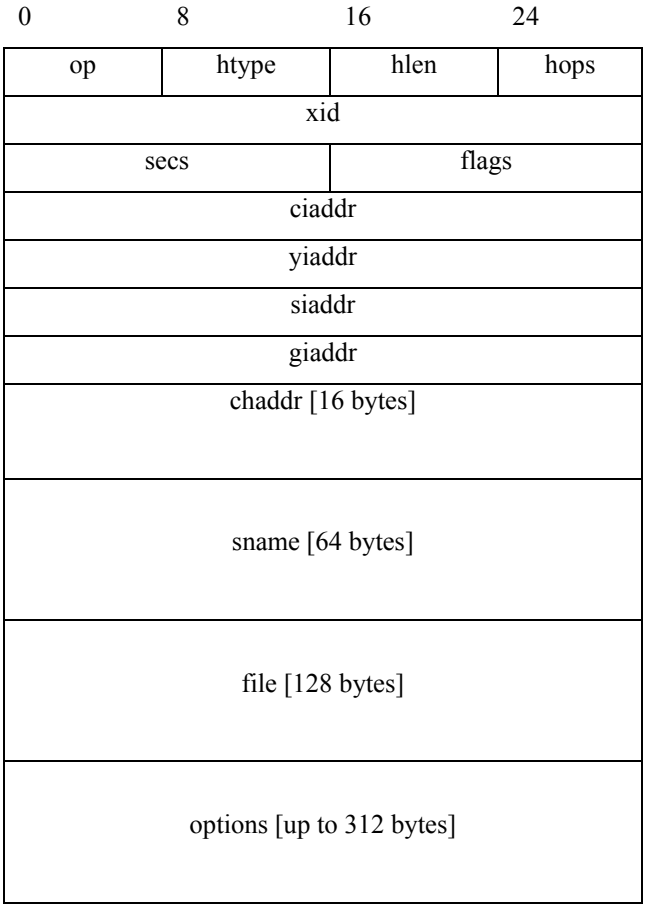


Figure 3-2: DHCP message

~~[O3-17d] The 'sname' field SHOULD be the ASCII null terminated string "GigE Vision".~~

[O3-18cd] A device SHOULD support the following DHCP options (see RFC2132):

- **Subnet mask option** (code 1). The subnet mask is useful to identify to which subnet a device belongs. This is particularly true for multi-homed configurations where the IP address may not be sufficient to identify to which network card the device is connected.
- **Router option** (code 3). The router option is used to identify a default gateway. This is useful when the application does not reside on the same subnet as the device.

3.3.1 DHCP Retransmission Strategy

RFC2131 provides general guidelines for DHCP retransmission. Because GigE Vision iterates through various IP configuration mechanism and the user expects the device to be accessible quickly upon power-up (within 20 seconds for this specification), this specification requires a different strategy. This ensures the device obtains an IP address in a reasonable amount of time.

When using DHCP, a device may time out for two different messages:

1. DHCPDISCOVER: When the device sends a DHCPDISCOVER message, the server is expected to answer with a DHCPOFFER message.
2. DHCPREQUEST: When the device sends a DHCPREQUEST message, the server is expected to answer with a DHCPACK or DHCPNAK message.

The device times out and retransmits the message if it does not receive any valid answer message from the DHCP server. A maximum of 2 retransmissions are allowed (for a total of 3 DHCPDISCOVER messages followed by 3 DHCPREQUEST messages in the worst case).

[R3-19cd] The DHCP retransmission strategy of a device MUST follow the following algorithm:

1. On the first transmission, if the device does not receive an answer message from a server within **2 seconds**, optionally randomized by the value of a uniform random number chosen from the range –1 to +1, the device performs a first retransmission.
2. On the first retransmission, if the device does not receive an answer message from a server within **4 seconds**, optionally randomized by the value of a uniform random number chosen from the range –1 to +1, the device performs a second retransmission.
3. On the second retransmission, if the device does not receive an answer message from a server within **6 seconds**, optionally randomized by the value of a uniform random number chosen from the range –1 to +1, the device moves to the next IP configuration protocol given by the protocol selection flowchart.

Using this algorithm, a device will typically pass 12 seconds in the DHCP stage when no DHCP server is present before moving to the next IP configuration scheme (15 seconds in worst case).

3.3.2 DHCP Lease Expiration

When the device is operated in DHCP mode, it must follow RFC2131. Of particular importance is the expiration of the lease for the IP address provided by the DHCP server. When this happens, the device must stop using this IP address and restart the IP configuration cycle. When no DHCP server is present on the network, then the device will fall back to LLA (assuming PersistentIP has not been enabled).

3.4 Link-Local Address

[R3-20cd] The device MUST support Link-Local Address (LLA). Refer to RFC3927 (Dynamic Configuration of IPv4 Link-Local Addresses) for a complete description of Link Local Address.

[R3-21cd] On a device, LLA MUST always be activated (it cannot be disabled).

LLA is sometime called “Auto IP” and is mapped to IP address range 169.254.xxx.xxx.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

~~[O3-22d] If LLA fails (no valid IP address), then device SHOULD fall back to IP address 0.0.0.0 so the application can discover the device and force an IP address into the device using GVCP.~~

The following bootstrap registers are used to support Link-local address configuration:

1. Supported IP configuration (address 0x0010 for first network interface): bit 29 indicates if Link-local address is supported by this device (bit set) or not (bit cleared). In the current GigE Vision specification, it is always 1.
2. Current IP configuration procedure (address 0x0014 for first network interface): bit 29 is always set since LLA MUST always be supported and activated (default IP configuration scheme).

4 Device Enumeration

[R4-1cd] Once the device has completed IP configuration, it MUST answer device discovery requests coming from any application if it has a valid IP address.

[R4-2cd] A device MUST NOT answer device discovery requests before it has completed IP configuration.

Device discovery messages are an integral part of GVCP.

An application may perform Device Enumeration to find devices, but it is not required to do so. This could be helpful to minimize bandwidth usage when the IP address and other information from the device are known ahead of time.

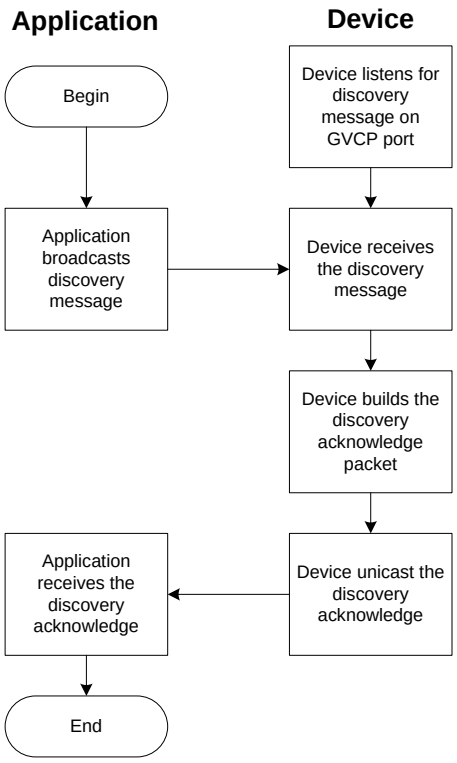


Figure 4-1: Device Discovery Flowchart

4.1 Broadcast Device Discovery

Broadcast Device Discovery messages can be used by an application to look for devices residing on the same subnet. It can be realized using a UDP broadcast message with an IP destination address of 255.255.255.255. This is defined as a "limited broadcast" by RFC1122 (Requirements for Internet Hosts -- Communication Layers). Note this message will not cross routers. This is why only devices on the same subnet will receive it. Using broadcast device discovery, it is not possible to enumerate devices on a different subnet than the network card(s) of the application.

This specification does not define to which network card(s) the limited broadcast message should be sent when the application resides on a multihomed host. Note that RFC1122 takes no stand on this issue either. To overcome this ambiguity, an application can use a "subnet directed" broadcast.

[R4-3cd] If a device has a valid IP address, then it MUST answer a broadcast device discovery message using a unicast answer message to the application. This applies to any type of broadcast requests (including limited broadcast and subnet directed broadcasts).

Definitions of broadcast types are provided in section 3.3.6 of RFC1122. Any time this specification refers to broadcast, this implicitly covers any type of broadcast.

[O4-4cd] If a device and an application don't reside on the same subnet, the device SHOULD send back a UDP broadcast to respond to a device discovery message if the application indicated that the discovery request could be answered using a broadcast message.

[R4-5cd] In the answer message, the device MUST set the source IP address, subnet mask and default gateway equal to the IP information obtained during IP configuration.

4.2 Unicast Device Discovery

Unicast Device Discovery messages can only be used when the device IP address is known beforehand by the application. It is realized by sending a UDP packet directly to the device IP address.

This specification does not state how the application knows the IP address of a device ahead of time. Forcing a fixed IP address using Persistent IP scheme or using a DNS are possible ways to do this.

[R4-6cd] A device MUST answer a unicast device discovery message using a unicast answer message to the application.

4.3 Associating the Device to the Enumeration List

Some criteria are recommended to allow easy identification of the device.

[O4-7cd] In order to easily associate a device to its entry in the discovery list, the device SHOULD have a serial number and MAC address label on the device casing.

This could be matched against the information returned by the device discovery message.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

5 Device Attachment and Removal

An integral part of Device Discovery is live attachment or removal of a device from the network. The 'live' designation indicates the application is started and it has already enumerated the devices when the topology of the network is changed.

[O5-1ca] An application SHOULD be able to dynamically react to a change of device network topology (adding or removing a device from the network).

5.1.1 Removal

Live removal is mainly handled by the control protocol. The application is going to timeout on the message commands it has sent (no acknowledge is received from the device when the application sends a heartbeat). Or alternatively, a control and receiving application can timeout on the video stream not coming anymore from the GVSP transmitter.

Because devices are not notified when other devices are removed from a network, they are unaffected by the removal of a device.

The way the application signals a device removal to the user, when this feature is supported, is left as a quality of implementation.

5.1.2 Attachment

On live attachment, usage of DHCP lets the server recognize a new device has been added to the network when it receives the DHCP request. This way, the server can nicely react and inform the application of the additional device. This unfortunately requires a close relationship between the server and the application. This might be difficult to achieve, especially if the server and the application do not reside on the same host.

Alternatively, another way for the application to know a new device is added to the network is to periodically send a DISCOVERY command. But sending periodical broadcast message consumes bandwidth out of the network, especially if many devices are to answer each time. One way to minimize bandwidth usage is to offer the user with a control to refresh the list of devices.

Because devices are not notified when other devices are added to a network, they are unaffected by the attachment of a new device (except for the amount of network bandwidth required to configure the new device).

The way the application signals a device attachment to the user, when this feature is supported, is left as a quality of implementation.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

PART 2 – GVCP

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

6 GVCP Summary

6.1 Overview

GVCP is an application layer protocol relying on the UDP transport layer protocol. It basically allows an application to configure a device (typically a camera) and to instantiate stream channels (GVSP transmitters or receivers, when applicable) on the device, and the device to notify an application when specific events occur.

GVCP provides necessary support for only one application (the primary application) to control a device (to write to a device). Nevertheless, it is possible for many applications to monitor a device (to read from a device) if this is allowed by the primary application. It is also possible for an application to request control of a device already under the control of a primary application provided that this is supported by the device and allowed by the primary application. Under GVCP, the application is the master and the device is the slave. Command requests are always initiated by the application.

Command and acknowledge messages must each be contained in a single packet. The application must wait for the acknowledge message (when requested) before sending the next command message. This creates a very basic handshake ensuring minimal flow control. The acknowledge message provides feedback that a command was actually received by the device and it also provides any response data requested by the command.

The current version on the specification uses UDP IPv4 as the transport layer protocol. Because UDP is unreliable, GVCP defines mechanisms to guaranty the reliability of packet transmission and to ensure minimal flow control.

6.2 Goals

The goals for GigE Vision Control Protocol (GVCP) are:

- Allow command and acknowledge messages to be exchanged between a GigE Vision device and a GigE Vision application.
- Provide a way for the application to instantiate a stream channel from the device, when applicable.
- Define a mechanism for a GigE Vision device to send asynchronous event messages to a GigE Vision application (using an event generator).
- Provide a uniqueness access scheme so only one application can control the device.
- Minimize IP stack complexity in the GigE Vision device.

6.3 Scope

This part provides:

- Specification of the control protocol.
- Description of GVCP headers.
- List of all command and acknowledge messages.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

- List of the different status codes representing error on the device.
- List of the different asynchronous event codes.

This control protocol does not cover GigE Vision Streaming Protocol (GVSP). This is discussed in a different part of the standard.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

7 GVCP Transport Protocol Considerations

For this version of the GigE Vision specification,

[R7-1c] GVCP MUST use UDP with IPv4 as the transport protocol.

[R7-2c] Device and application MUST NOT use any IP options in the IP datagram for GVCP.

This way, the IP header size is fixed at 20 bytes. This is to allow efficient hardware implementation of UDP/IP offload engines.

This section defines various mechanisms implemented by GVCP to guaranty the reliability of the transmission between the application and the device.

7.1 UDP

UDP is a connectionless protocol that adds no reliability, flow-control or error recovery to IP. Because of this lack of functionality, GVCP imposes restrictions on how connections are handled.

[R7-3c] GVCP MUST use UDP port 3956 of the device.

This port has been registered with IANA (http://www.iana.org/assignments/port-numbers). It is referred to as the "standard GVCP port" in the rest of this document. The application can use any available dynamic port number. When an application sends a GVCP packet, it uses the device GVCP port (3956) as the destination and its dynamically allocated UDP port number as the source. The device answers GVCP requests by using the application dynamic UDP port as the destination and its GVCP port (3956) as the source.

7.1.1 Fragmentation

Fragmentation defines the way a large message is broken into smaller segments suitable to be transmitted over the IP protocol.

[R7-4c] In GVCP, command and acknowledge messages MUST each be contained in a single packet.

This is to guaranty the packet will never necessitate IP fragmentation. The communication stack of the device can thus be simplified as it does not need to handle IP de-fragmentation. A device may reject fragmented IP datagram.

[O7-5c] The "do not fragment" bit SHOULD always be set for all GVCP commands and acknowledges.

For IP, the MTU to avoid IP fragmentation is 576 bytes (The maximum size datagram that all hosts are required to accept or reassemble from fragments is 576 bytes).

[R7-6c] Applications and devices MUST send GVCP packets containing at most 576 bytes.

These 576 bytes include the IP header, UDP header, GVCP header and payload (Ethernet header is not part of the 576 bytes). The following table lists the number of bytes required by each header. Note that GVSP might use packets larger than 576 bytes.

Table 7-1: GVCP Packet Header Size

Layer	Size (in bytes)
IP header (options not allowed)	20
UDP header	8
GVCP header	8
Max. GVCP payload	540
Total	**576**

Note that even though the application and the device might handle packets larger than 576 bytes without IP fragmentation, other network components (such as switches and routers) might inflict this restriction.

This limit of 576 bytes maximum packet size imposes a restriction on the GVCP message themselves. This is incorporated in the message definition provided by this specification.

7.1.2 Packet Size Requirements

To simplify packet handling, the following restriction is established.

[R7-7c] GVCP payload MUST have a size multiple of 32-bit.

The GVCP header is itself a multiple of 32-bit.

7.1.3 Reliability and Error Recovery

Reliability ensures any given packet has been correctly received by the destination. In GVCP, this is realized by the application optionally requesting an acknowledge from each command message. This is illustrated in the following figure.

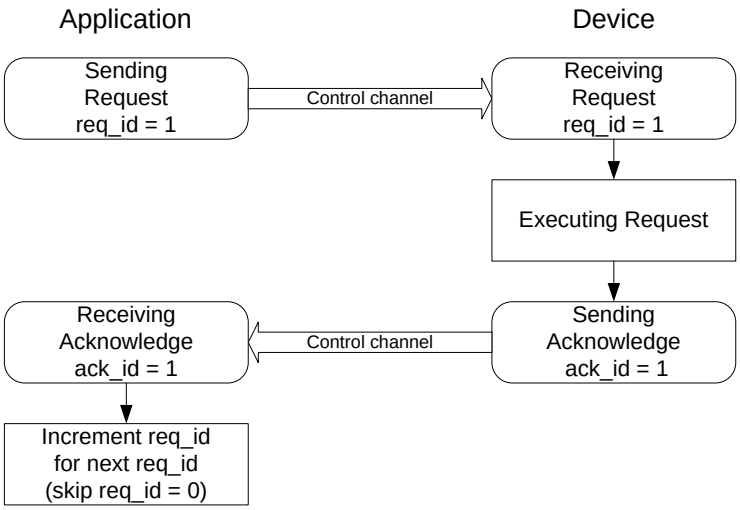


Figure 7-1: Use of Acknowledge

After sending a command, the application waits for the acknowledge (when requested). In order to detect if a message was not received by the device, the application implements a counter to signal timeout conditions. Note that an application is not required to request an acknowledge. This is left as a quality of implementation but can be useful to manage flow control.

[O7-8ca] When an acknowledge is requested, if after a user-configurable timeout the application did not receive the acknowledge, it SHOULD send the command again.

[R7-9ca] When resending the command, the application MUST leave the req_id field unchanged.

If the application reaches the maximum number of retries, it reports the error to the upper software layer. The way this error is reported by the application is left as a quality of implementation.

[O7-10ca] For a control channel, the application SHOULD provide a user programmable timeout value for GVCP message.

This acknowledge timeout may have a default value of 200 ms.

Note: This specification does not provide a requirement on the default acknowledge timeout value since this is considered system dependent.

[O7-11ca] The application SHOULD provide a user programmable number of retries value for GVCP message.

The number of retries may have a default value of 3.

Because of the symmetry between control and message channels, the device's bootstrap registers provide transmission timeout and retry count registers for the message channel (when this channel is available).

[R7-12ca] The req_id field of GVCP messages MUST be incremented from one message to the next by the application.

The initial value of req_id is not specified, but it cannot be 0. The req_id may be reset to the initial value when the control channel is closed.

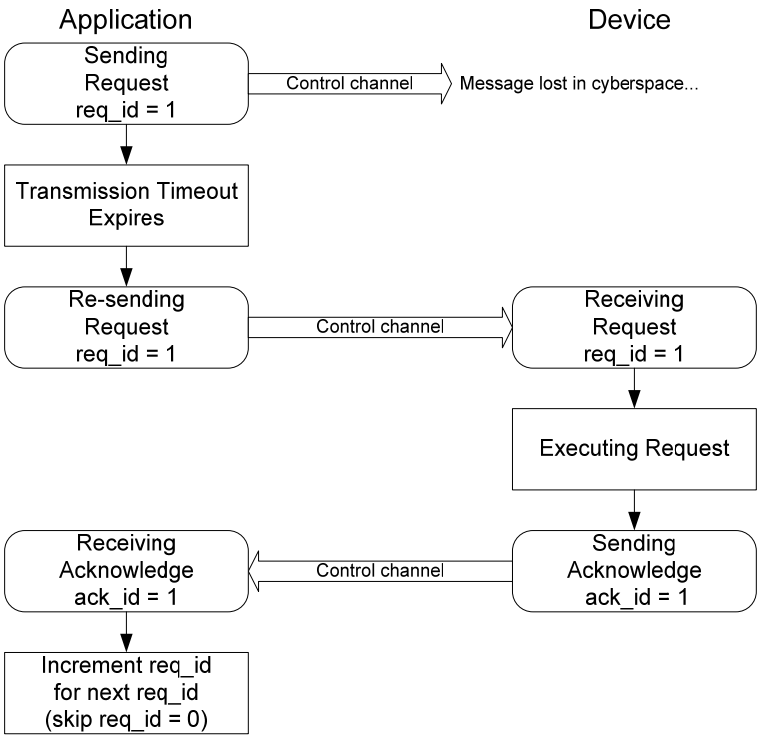


Figure 7-2: Timeout on Request

On the other end, if the device receives a command with the same req_id field as the previously received command, it knows the same command has been received twice. If running the command twice makes no difference to the device state, it is allowed to do that. Otherwise the command is only executed once.

[R7-13cd] When receiving a command with the same req_id, if running the command twice makes a difference, the device MUST return the corresponding acknowledge message again without executing the command the second time.

[R7-14cd] In all cases where an acknowledge is requested, the device MUST return an acknowledge for each command it receives (the original and the repeated commands).

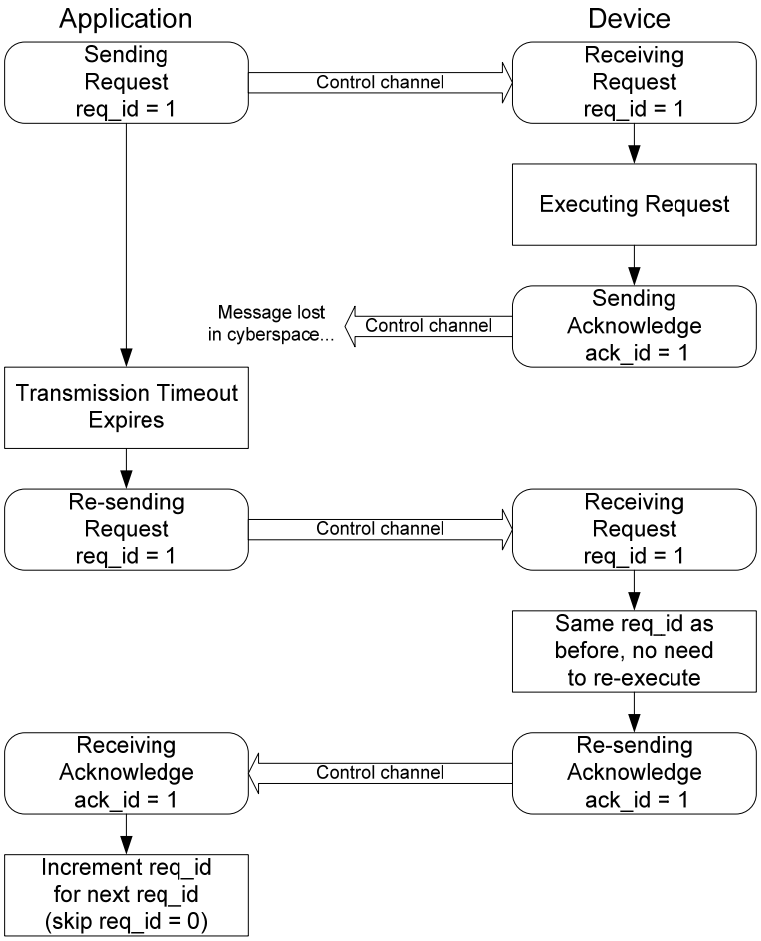


Figure 7-3: Timeout on Acknowledge

Ethernet provides a CRC to guaranty data integrity. If the data does not match the CRC, then the packet is silently discarded. UDP checksums are not required by GVCP, but may be used.

The GVCP header provides an 8-bit field containing the hexadecimal key value 0x42. This enables devices and applications to identify GVCP packets.

~~[O7-15cd] The device SHOULD validate the key field before processing the command. If this field does not contain the hexadecimal value 0x42, then the packet is considered erroneous.~~

~~[R7-16c] If a device or an application receives an erroneous GVCP packet, it MUST silently discard it.~~

The timeout mechanism will ensure retransmission of a corrupted GVCP message. Watchout for Sorcerer's Apprentice Syndrome (explained in RFC1123 – Requirements for Internet Hosts – Application and Support): when the application receives an ACK message that does not match the current ack_id, it must silently discard it.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

7.1.4 Flow Control

Flow control ensures the device does not overflow its internal communication buffers when receiving messages. A simple scheme is required by GVCP:

An application sends one packet and then it must wait for the acknowledge message when an acknowledge is requested. It is not permitted to send a second packet before the first acknowledge message has been received unless the transmission timeout expires or no acknowledge had been requested by the application.

~~[R7-17ca] An application sends one packet and then it MUST wait for the acknowledge message when an acknowledge is requested. It is not permitted to send a second packet before the first acknowledge message has been received unless the transmission timeout expires or no acknowledge had been requested by the application.~~

In the case that no acknowledge is requested, flow control management is left as a quality of implementation. In this case, the application does not have to wait for acknowledge from the device. It is up to the application to ensure it does not overflow the device.

7.1.5 End-to-End Connection

UDP is a connectionless protocol. As such, it does not inherently support a scheme to guaranty the remote device is still connected.

GVCP requires the device to support a heartbeat counter. Any valid command or GVCP packet coming from the primary application resets the heartbeat counter.

[R7-18ca] An application MUST provide a user programmable heartbeat timeout parameter.

[R7-19ca] If no GVCP message is sent by the application, then the programmable heartbeat timeout MUST ensure a message is sent to the device, so the device's heartbeat counter is reset.

Only the primary application has to send heartbeat messages.

The device offers a heartbeat timeout bootstrap register. It is recommended to set the application heartbeat frequency parameter to a little less than one third of the device heartbeat timeout. This ensures at least two heartbeat UDP packets can be lost without the connection being automatically closed by the device because of heartbeat timeout. Depending on the quality of the network connection, it is possible to adjust these numbers.

8 The Channel Concept

In order to handle multiple connections each carrying different types of information, GigE Vision introduces the concept of "channels".

A channel is a virtual link used to transfer information between GigE Vision entities. GigE Vision supports three types of channels:

1. ***Control channel*** (there is always 1 control channel for the primary application)
2. ***Stream channel*** (from 0 to 512 stream channels)
3. ***Message channel*** (0 or 1 message channel)

Different channels can use the same network interface. In this case, they are each assigned a different UDP port. The following figure represents those channels and highlights the direction of data flow for a basic use case while Figure 8-2 presents an advanced one.

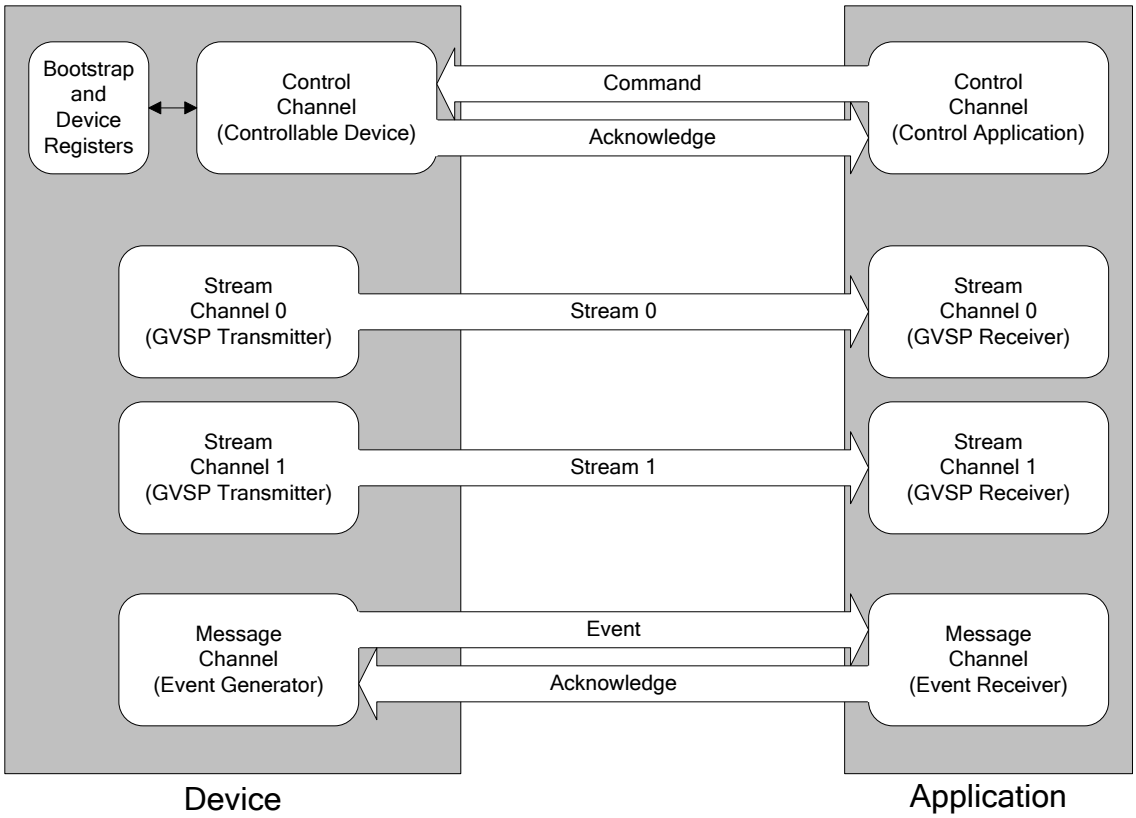


Figure 8-1: Basic Channels Example

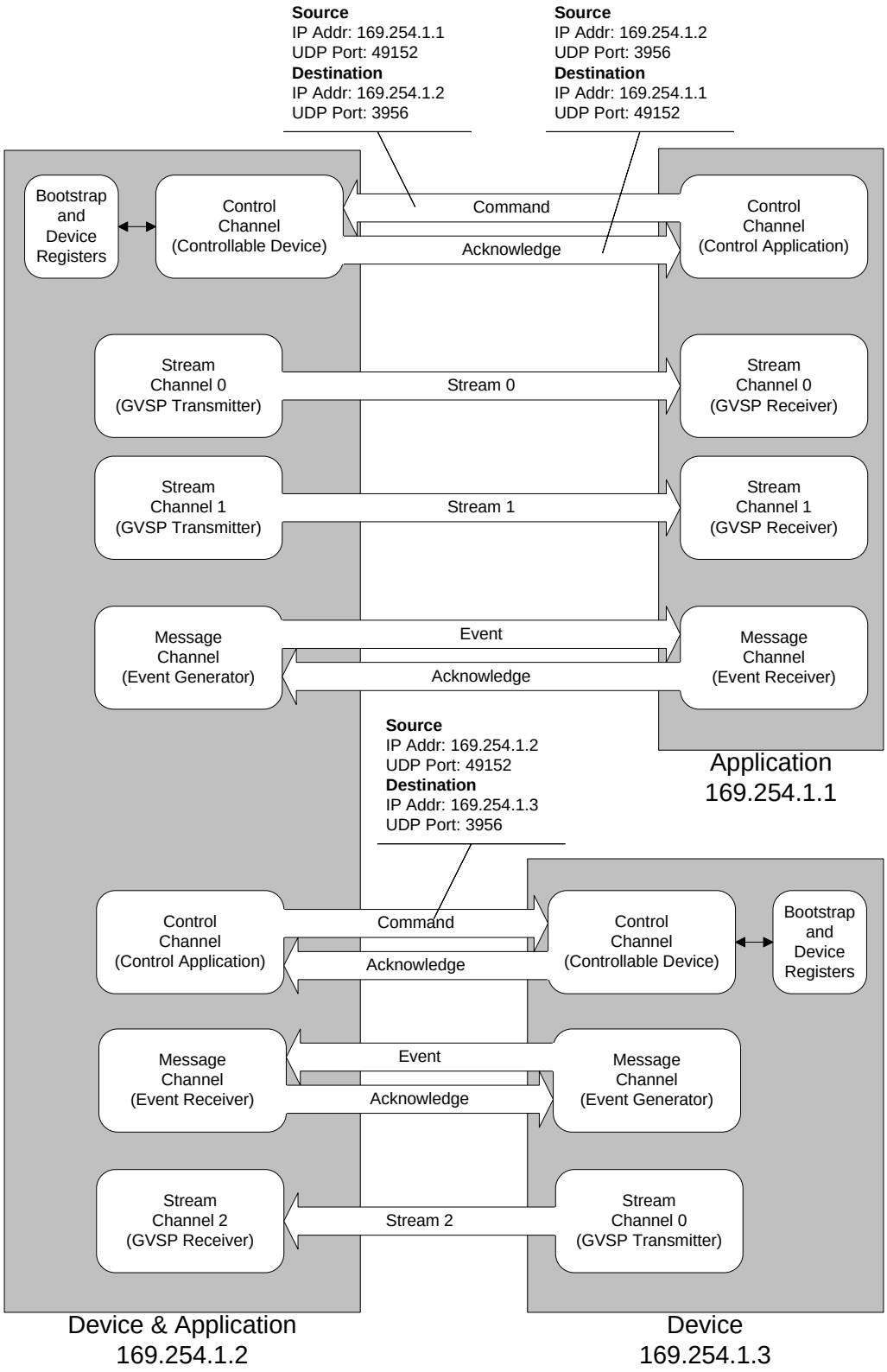


Figure 8-2: Advanced Channels Example

Channels are created dynamically by the application and the device. Except for the standard GVCP port used by the device for the control channel, channels can use any available UDP port.

[R8-1s] A product supporting GVSP streaming MUST offer at least one and at most 512 stream channels.

Note: A product is not required to support streaming. In this case, it doesn't include any stream channel.

[R8-2cd] A device MUST offer either one or no message channel (event generator).

The application can open or close these channels as needed.

[R8-3ca] An application MUST instantiate a control channel to access the device registers.

The device bootstrap registers allow the primary application to create the message and stream channels. It is not possible for an application to instantiate message or stream channels if it does not first have a control channel with exclusive or control access (without or with switchover enabled) to the device.

9 Control Channel

A control channel is used by an application to communicate with a device. GVCP defines two types of control channels:

1. ***Primary control channel***: A primary control channel is created by the primary application. The primary application is the one that is allowed to write to the device registers. Only one application is allowed to be the primary application for a device.
2. ***Secondary control channel***: A secondary control channel is created by any secondary applications. Secondary applications can only read from the device registers (they cannot write into them). This can be used for monitoring or debugging. A device may support many secondary applications. A device is allowed to support no secondary application.

Note: Even though only one primary application is supported, it is possible to send/receive image/data streams to/from multiple recipients using UDP broadcast or multicast. But all streams are under the control of the primary application (except for PACKETRESEND requests).

Note: A device has to answer a device discovery request from any application after IP configuration is completed, even if the request is not coming from the primary application.

A control channel is required to send commands from the application to the device and to receive the corresponding acknowledge (when one is requested).

[R9-1cd] A device MUST support one primary control channel.

A device may support none or any number of secondary control channels.

A control channel must be instantiated by the primary application before stream or message channels can be created.

The packet flow sequence on a control channel is:

1. Application sends a command packet
2. Device receives the command packet
3. Device executes the command
4. Device sends the acknowledge packet (if acknowledge is requested)
5. Application receives the acknowledge packet (if acknowledge is requested)

[O9-2cd] The device SHOULD send back the acknowledge (if one is requested) as soon as the immediate action requested by the command has been executed.

For instance, if the application has requested to write to a register to initiate an image capture, the device should send the acknowledge when the register has been written to (the immediate action), not when the image capture is completed (the indirect reaction of writing into the register).

[R9-3ca] An application MUST NOT send a second command before it has received the first acknowledge (the only exceptions being a DISCOVERY_CMD, a PACKETRESEND_CMD, a retry packet when the transmission timeout expires or when no acknowledge is requested).

It is recommended that a device executes all command requests and sends back the acknowledge message very efficiently so an application can send the next request.

9.1 Control Channel Privileges

GVCP defines four levels of privileges:

1. ***Exclusive access***: An application with exclusive access is the primary application for this device. It can read or write to the device. No other application can read or write to this device.

[R9-4cd] A device MUST support exclusive access.

[R9-5cd] When an application has exclusive access, the device MUST not allow a secondary application to create a secondary control channel.

A device can grant exclusive access if there is no application registered with exclusive access or with control access. If an application has control access with switchover enabled, then the device can grant exclusive access to another application provided that the latter application has the right credentials.

[R9-6cd] When a device has granted exclusive access, it MUST return an error (GEV_STATUS_ACCESS_DENIED) in the acknowledge message to any other application sending command message, with the exception of ACTION_CMD, DISCOVERY_CMD, PACKETRESEND_CMD and FORCEIP_CMD. FORCEIP_CMD must respect [R14-12cd].

Exclusive access provides a minimal level of security since other application cannot access the device. Exclusive access is granted by writing into the CCP register. A device has to answer any DISCOVERY_CMD message at all time. This is true even if a primary application has exclusive control of the device.

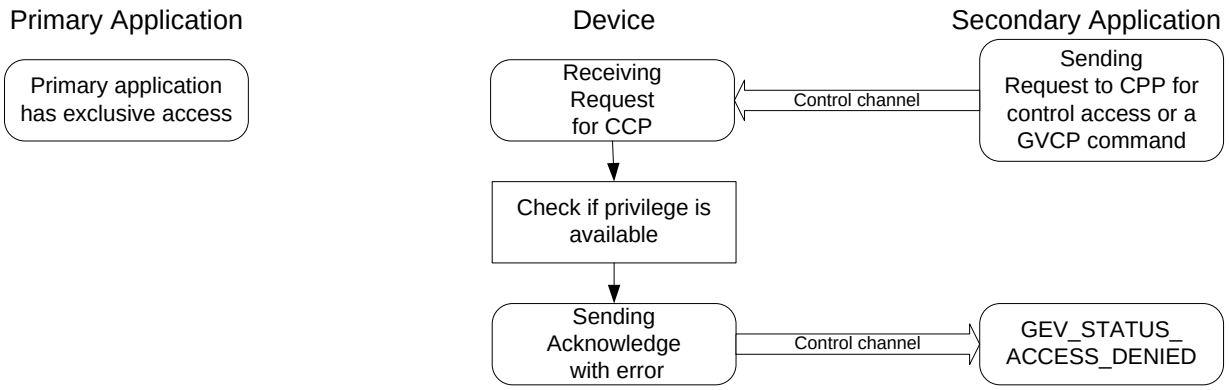


Figure 9-1: Exclusive Access

2. ***Control access***: An application with control access is the primary application for the device. It can read or write to the device. But other applications are allowed to read from the device. Note that a device IS NOT required to support control access.

[CO9-7cd] When a device supports control access and when an application has control access, the device SHOULD allow a secondary application to create a control channel supporting the READREG and READMEM messages.

[CR9-8cd] When a device supports control access, it can grant control access if there is no application registered with exclusive access or with control access. If an application has control access with switchover enabled, then the device can grant access to another application if the latter has the right credentials. An application has the right credentials if the control_switchover_key field of the CCP register is set to the value held by the Control Switchover Key bootstrap register when access is requested by this application. Otherwise, it MUST return an error (GEV_STATUS_ACCESS_DENIED).

Control access is granted by writing into the CCP register.

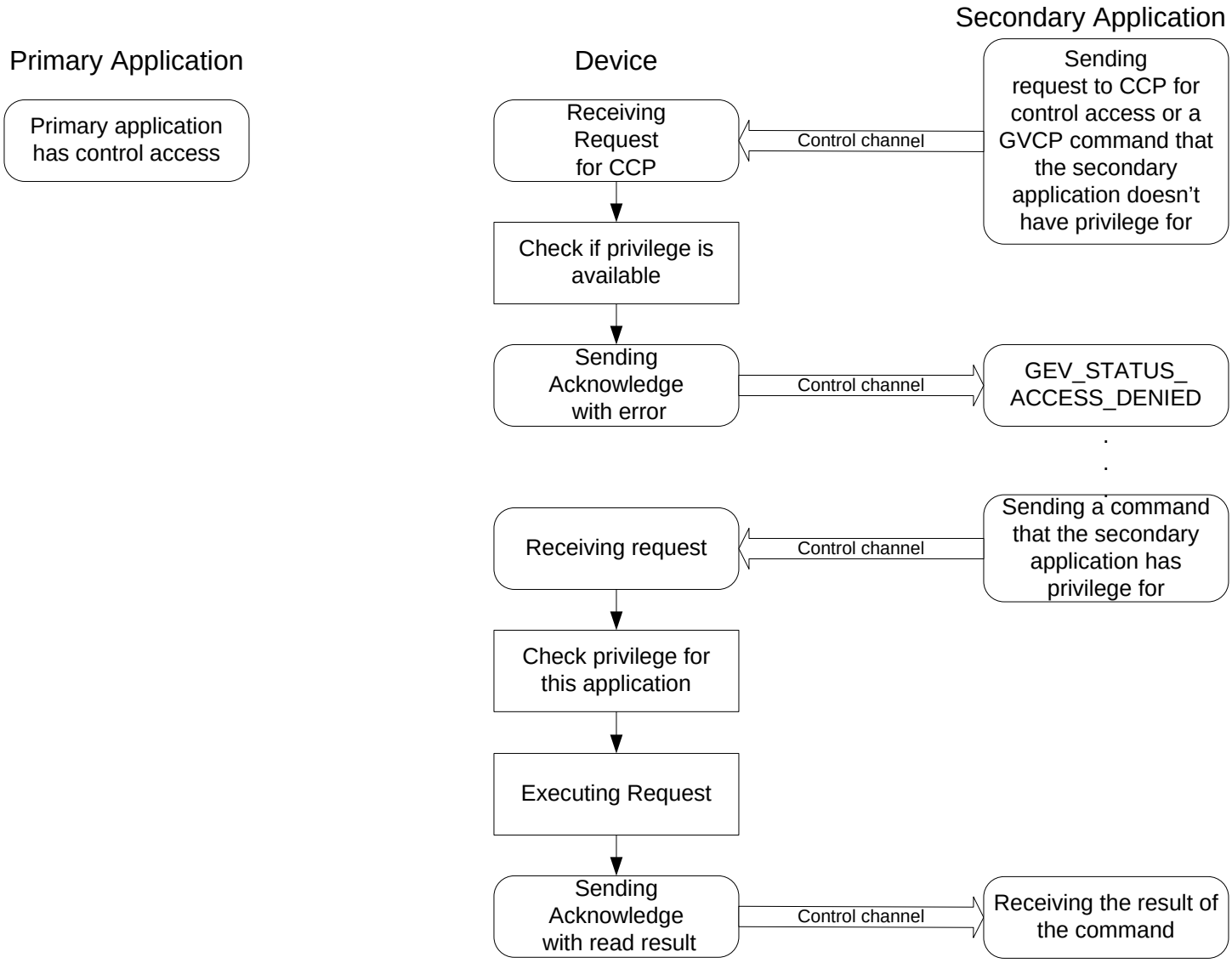


Figure 9-2: Control Access

3. ***Control access with switchover enabled***: Just like in the control access mode, an application with control access with switchover enabled is the primary application for the device. It can read or write to the device. Other applications are allowed to read from the device. However, this mode enables another application to take control over the device provided that it has the right credentials. Note that a device IS NOT required to support control access with switchover enabled.

[CO9-17cd] When a device supports control access with switchover enabled and when an application has control access with switchover enabled, the device SHOULD allow a secondary application to create a control channel supporting the READREG and READMEM messages.

[CR9-18cd] When a device supports control access with switchover enabled, it can grant control access if there is no application registered with exclusive access or with control access. If an application has control access with switchover enabled, then the device can grant access to another application if the latter has the right credentials. An application has the right credentials if the control_switchover_key field of the CCP register is set to the value held by the Control Switchover Key bootstrap register when access is requested by this application. Otherwise, it MUST return an error (GEV_STATUS_ACCESS_DENIED).

Control access with switchover enabled is granted by writing into the CCP register.

4. ***Monitor access***: An application with monitor access has no particular privilege over the device. It is a secondary application that can only read from the device when no application with exclusive control access is linked with the device. A secondary application is typically used to help debugging. For instance, it can be used to dump the registers of the device. A device can allow monitor access only if there is no application with exclusive access. If this privilege is available, the secondary application can directly send a READREG_CMD or READMEM_CMD message to the device. An application does not write to the CCP register to get monitor access. Note that a device is not required to support monitor access. A device is not required to track the req_id of a secondary application: each command from a secondary application, even a retried command, gets executed.

It is perfectly legal for a device to only support exclusive access. In this case, only the primary application can access this device.

[R9-9cd] The device MUST memorize the context associated with the primary application. This context is at least composed of:

1. The IP address of the application
2. The source UDP port for the application
3. The granted privilege (exclusive, control access or control access with switchover enabled)

The device checks this context to determine if it can grant privilege to other applications and if received command messages are valid.

Once a device has been discovered, an application can take command of it by opening a primary control channel. This ensures only a single application can control the device.

Note: Channel is not a security scheme. It only ensures one application is bound to control the device.

Control channels are created using dynamic port number (chosen arbitrarily) on the application side and the standard GVCP port on the device end. The application must select a port number for the life period of the channel. The application must also indicate if it wants to take exclusive control of the device or allow other

application to monitor the device. If supported by the device, the application can also enable another application to take control over the device provided that the later application has the right credentials.

Note: Having non-exclusive access to the device may be useful during application development to allow other debugging tools monitor the device.

Note: Enabling another application to take control over a device may be required in systems were redundancy and fault recovery is necessary. Please refer to Section 9.9 for details.

By always using the standard GVCP port for the channel on the device side, the connection procedure is made simpler.

9.2 Control Channel Registers

The control channel provides the following registers:

1. Control Channel Privilege register (CCP)
2. Control Switchover Key register
3. Heartbeat Timeout register
4. Pending Timeout register

Refer to the bootstrap registers section for a complete description of the fields in these registers.

Note: The list above is not an exhaustive list of all the registers associated with the control channel. It lists the main registers related to the concepts presented in this section.

9.3 Opening a Control Channel

An application opens a control channel to take exclusive or control access (without or with switchover enabled). There is no need to open a control channel for monitor access.

To open a channel, an application writes the requested privileges to the CCP register and check for status returned by the acknowledge message of device. If successful, application now has the privilege and the device returns GEV_STATUS_SUCCESS. Otherwise, device returns status GEV_STATUS_ACCESS_DENIED. The application knows from the status code if it has been granted the requested privilege.

When the device receives a command to write to the CCP register it tries to create a channel context if the application is asking for exclusive or control access (without or with switchover enabled). Any secondary application does not have to write to the CCP register: it can directly read from the device. The device verifies whether it can give the desired privilege and it updates the context if necessary.

[R9-10cd] A primary application is allowed to ask for the same privilege without closing the control channel. In this case, the device MUST accept the request and return GEV_STATUS_SUCCESS.

The primary application is allowed to directly switch to another control privilege when writing the CCP bootstrap register.

[R9-11cd] A device MUST allow an application to directly switch between Exclusive, Control and Control with Switchover Enabled (when supported) privilege without having to first disconnect the control channel.

Another application that has the right credentials can request, and get granted, device control when the current primary application has been granted control access with switchover enabled. Control access with switchover enabled is detailed in Section 9.9.

9.4 Closing a Control Channel

A primary application closes its opened control channel by writing 0 to the CCP register. Another application cannot write into the CCP register unless the primary application has been granted control access with switchover enabled. In this case, the CCP request can be granted if the other application has the right credentials. Otherwise, it gets a GEV_STATUS_ACCESS_DENIED from the device.

[R9-12cd] When the primary control channel is closed, the device MUST stop streaming, unless it has been configured otherwise via the unconditional streaming enabled bit of the Stream Channel Configuration registers, reset all connection states (including message and GVSP transmitter stream channels) by at least clearing the MCP register and the SCPx registers of the GVSP transmitters (if applicable), resetting the Primary Application Port and Primary Application IP Address registers (if supported) and make itself available for another connection upon control channel closure. Device's registers (including bootstrap registers) MUST represent the new state after the operation is completed.

Note: When closing a primary control channel, the application might leave the device with an incoherent set of registers. It is always up to the primary application to ensure it provides a complete coherent set of registers when it takes exclusive or control access of a device (without or with switchover enabled). For instance, a device might have a default pixel size of 10-bit. The first primary application might set pixel size to 8-bit (non-default value) then close the control channel leaving 8-bit programmed into the device. A second application might then program the device for acquisition, but leave the pixel size register unchanged, assuming the device still uses its default value. Obviously the set of device registers does not match what the second application expects.

Note: Resetting the GVSP receiver stream channels when the control channel is closed is left as an implementation detail on devices including GVSP receivers but it is recommended to leave them open. That is because a simple receiver product may want to keep displaying incoming video feeds even if a primary control channel is not open on the device.

Note: In some video distribution systems over Ethernet, it is desirable for video sources to keep transmitting video even if the primary application crashes or is closed. This version of the specification enables GVSP transmitters to continue streaming independently of the state of the control channel or of any ICMP messages received by their associated device (such as destination unreachable messages). When this feature is supported by GVSP transmitters, it is enabled with configuration bits. Please refer to Section 10.8 for more details on the topic.

9.5 Control Channel Heartbeat

Because UDP is a connectionless protocol, the GVCP must implement a recovery mechanism when one of the two participants is abruptly detached without a proper channel close procedure. This is achieved using a heartbeat sequence. The application must periodically run this sequence if there is no activity on the control channel. The rate at which the sequence must be run is user programmable. The default value is once per second and it is programmable in the application.

[R9-13ca] The application MUST provide a parameter to indicate the heartbeat rate.

[R9-14cd] The device MUST provide a bootstrap register to control the heartbeat timeout.

[R9-15cd] Any valid GVCP command from the primary application MUST reset the heartbeat counter on the corresponding device, with the exception of the PACKETRESEND_CMD, the DISCOVERY_CMD, the ACTION_CMD and the FORCEIP_CMD. FORCEIP_CMD must respect [R14-12cd].

Note: ACTION_CMD can be coming from a different UDP port than the Control Channel. Hence it cannot reset the heartbeat.

Otherwise, it is recommended for the application to read the CCP register as the method to reset heartbeat counter. This allows the application to ensure the granted privilege still matches the previously granted privilege.

Note: The heartbeat counter can only be reset by the primary application. Read access by secondary application have no effect on the heartbeat counter.

[R9-16cd] If the device does not receive a control message for more than the user-programmable value in bootstrap registers (three seconds is the default value) and the heartbeat capability has not been disabled on the device, then it MUST close the control channel as described in the previous section.

The application detects a device malfunction when it cannot read the CCP register or if it reads an unexpected value. If the application reads an unexpected value, it assumes the connection has been lost. The application must then establish new connection with the device by instantiating the control channel.

9.6 Controlling the Device

Once an application has opened a primary control channel with a device, it can send any command supported by GVCP. Refer to the dictionary sections of this document (Control Channel Dictionary and Message Channel Dictionary).

Secondary applications can send READREG and READMEM commands to read device state.

DISCOVERY command may be sent at any time by any application and be correctly answered by the device. A device has to always answer DISCOVERY commands from any application after it has completed its IP configuration cycle.

If the device and the application don't reside on the same subnet, then the device can broadcast the DISCOVERY_ACK message. Otherwise, it can use unicast transfer to the source of the discovery request.

ACTION and PACKETRESEND commands can also be sent at any time by any application and be processed properly by the device.

Finally, a FORCEIP command received from a secondary application must be handled according to [R14-12cd].

9.7 Use of Pending Acknowledge

GVCP provides a very simple handshake between the device and the application through the acknowledge message. But sometimes, command execution can take longer to complete than what is expected by the application. It is thus useful to have a mechanism to communicate:

1) The maximum amount of time to execute a typical request;
2) Have a way to notify the application when request execution time is going to exceed this value so the application can correctly wait the extra amount of time necessary to complete the request.

Version 1.0 of this specification does not provide any mechanism for the device to indicate to the application that the request execution time will exceed the timeout. Therefore, when a request takes too long to execute, the application is likely to consider the packet was lost and try to retransmit it using the same req_id, resulting in inefficient handling of the command.

Version 1.1 of this specification introduces a special ACK message called PENDING_ACK that is sent by the device to tell the application the current command execution time is going to take longer than the value reported by the Pending Timeout bootstrap register.

[O9-2cd] provides a guideline to quickly complete the execution of the immediate action. In this situation, no PENDING_ACK needs to be issued. The following diagram illustrates this scenario.

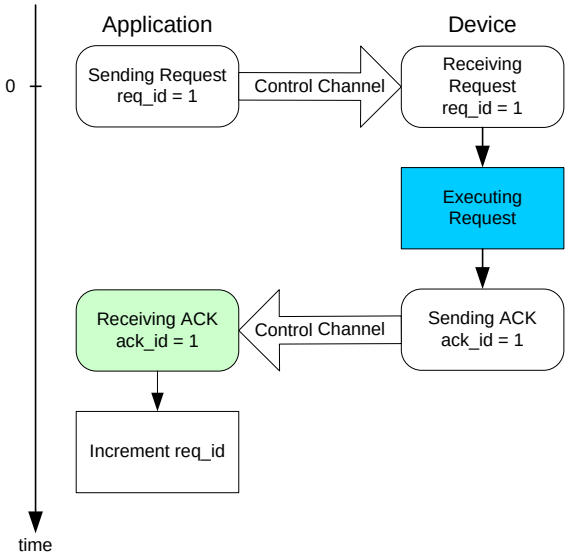


Figure 9-3: No PENDING_ACK

The following diagram presents an example of the retransmission of a command as a result of the expiration of the application ACK timeout before the end of the execution of the command on the device.

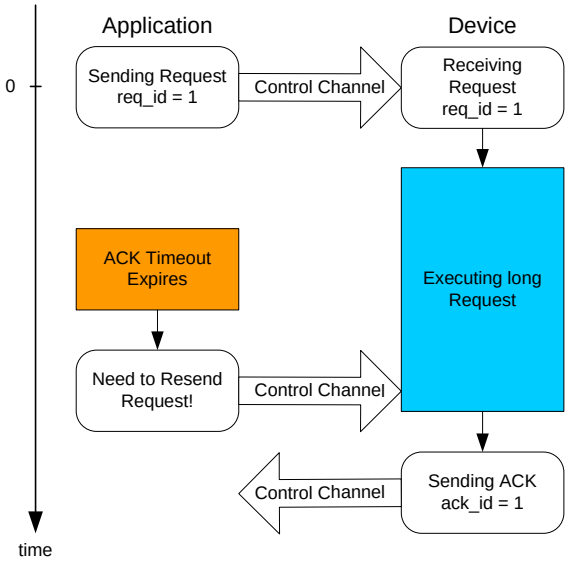


Figure 9-4: Request Retransmission (PENDING_ACK not enabled)

Note that in a situation like this, it is very plausible the ACK of the initial request will be interpreted by the application as the ACK for the retransmission. Typically, a device will not re-execute a request with the same req_id. Nevertheless, the device *might* be re-executing the long command a second time (see [R7-13cd] and the paragraph before). This might lead to another timeout on the next request coming from the

application! Moreover, as per [R7-14cd], a device will return 2 ACKs: one for the initial command and one for the retransmitted one.

Starting with the previous timeout scenario, if the application enables the PENDING_ACK, the device has a way to notify the application that a command may take longer to execute. The device returns a PENDING_ACK to reset the application ACK timeout (to 2000 ms in the following example). Therefore, the application has a clear indication of the amount of time required to complete the original request (if it doesn't like it, application can decide to timeout anyway!). Note the ack_id of PENDING_ACK must stay the same as for the initial request. This clearly indicates the PENDING_ACK belongs to this request.

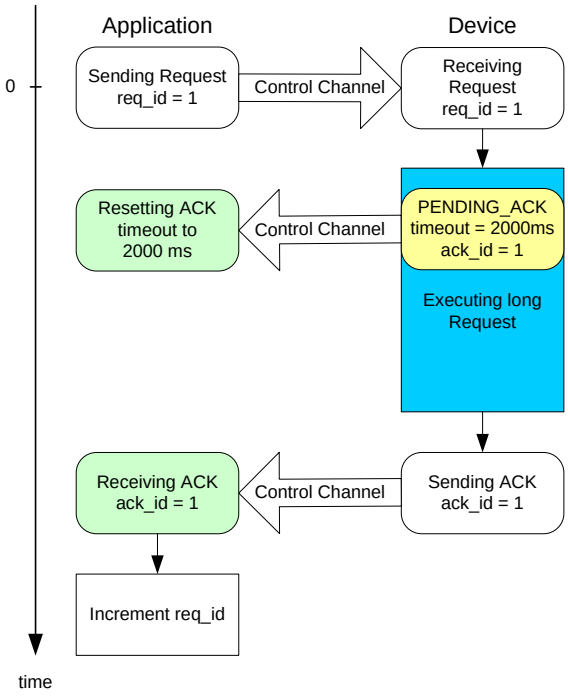


Figure 9-5: Device using PENDING_ACK

An application receiving a PENDING_ACK has to wait for the indicated timeout to complete or for the actual request ACK to arrive before sending the next command. Also note that if the PENDING_ACK message is lost, then the Application will behave in same way as with version 1.0 of this specification. It is important to notice that devices must still answer DISCOVERY_CMD message between the time a PENDING_ACK is issued and the command ACK is sent. Otherwise, other applications on the network might not correctly enumerate devices on the network.

Note: How a device should react when it receives a GVCP command (other than DISCOVERY_CMD) after a PENDING_ACK has been issued, but before the final acknowledge is sent, is left as a quality of implementation.

For instance, the device may have the capacity to process requests from different applications simultaneously. Alternatively, it could respond with a PENDING_ACK to the new request with a time to completion value equal to the remaining time of the previous command plus the time required to execute

the new one (even if the new command would not normally return PENDING_ACK). This would indicate that the new command was queued for processing and should not be re-issued. Another option would be for the device to respond with a GEV_STATUS_BUSY status and let the application resend the command at a later time. However, a device can simply queue the new command as any other packets which would increase the likelihood for this command to timeout since the application issuing the new request would not have received the PENDING_ACK of the previous command.

If a device responds with a PENDING_ACK to a request from a secondary application, then it is assumed that the primary and all secondary applications of the system can support PENDING_ACK. However, how a primary application can determine if all the secondary applications of a system can support PENDING_ACK is beyond the scope of this specification and is left as system level consideration.

9.8 Action Commands

Triggering an action in multiple devices at roughly the same time can be accomplished through the optional ACTION_CMD message. Using a distinct command (instead of a WRITEREG_CMD) has the advantage that it does not rely on a specific register map. Therefore, different type of devices can be targeted with the same message. Note that due to the nature of Ethernet, this is not as synchronous as a hardware trigger since different network segments can have different latencies. Nevertheless in a switched network, the resulting jitter is acceptable for a broad range of applications and this scheme provides a convenient way to synchronize devices by means of a software command.

ACTION_CMD message can be unicasted or broadcasted by the primary or any secondary applications to the GVCP port of the device. The acknowledge (ACTION_ACK) sent back by the device is always a unicast message.

The most typical scenario is when an application desires to trigger a simultaneous action on multiple devices. This case is showed by the figure below for a primary application. The application fires a broadcast ACTION_CMD that will reach all the devices on the subnet.

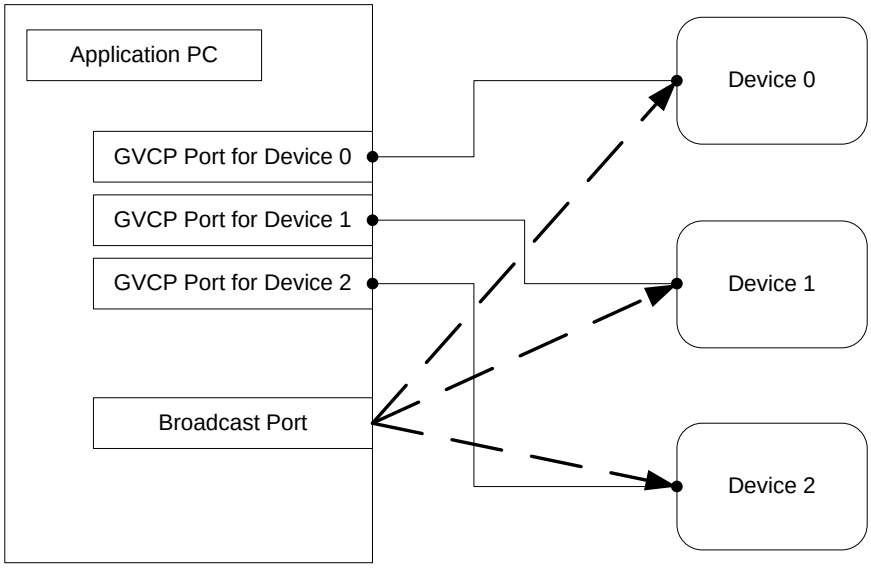


Figure 9-6: ACTION_CMD broadcast from primary application

But ACTION_CMD can also be used by secondary applications. This can even be another device on the same subnet. This is depicted in the following figure.

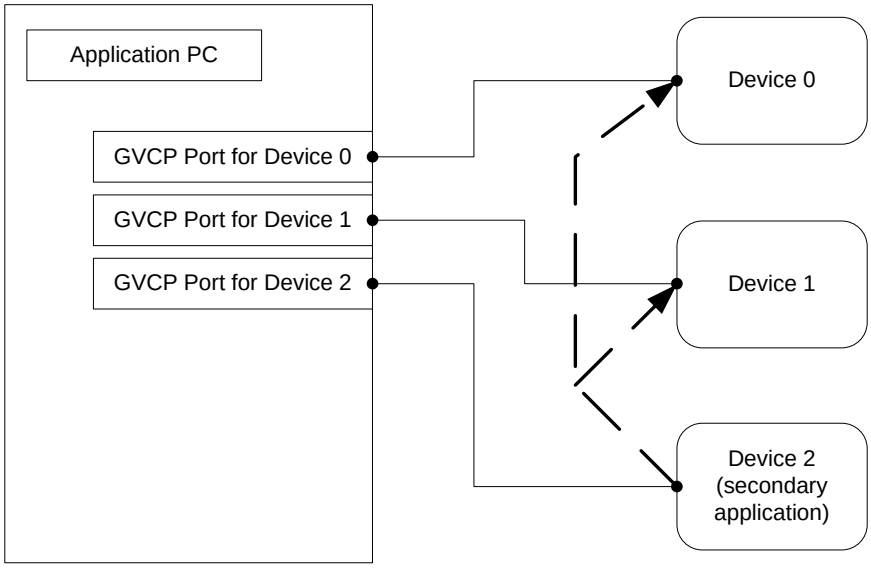


Figure 9-7: ACTION_CMD broadcast from another source (secondary application)

Upon reception of an ACTION_CMD message, the device will decode the information to identify which internal action signal is requested. An action signal is a device internal signal that can be used as a trigger for functional units inside the device (ex: a frame trigger). It can be routed to all signal sinks of the device.

Each ACTION_CMD message contains information for the device to validate the requested operation:

1. **device_key** to authorize the action on this device.
2. **group_key** to define a group of devices on which actions have to be executed.
3. **group_mask** to be used to filter out some of these devices from the group.

Action signals can only be asserted if the device has an open primary control channel.

Table 9-1: ACTION command four conditions

The conditions for an action signal ACTION_x to be asserted by the device are: 1. the device has an open primary control channel. 2. the **device_key** in the ACTION_CMD packet and the **Action Device Key** configured in the boostrap register (at address 0x090C) must be equal. 3. the **group_key** in the ACTION_CMD packet and the **ACTION_GROUP_KEYx** of the corresponding device action bootstrap register for ACTION_x must be equal. 4. the logical AND-wise operation of **group_mask** in the ACTION_CMD packet against the

ACTION_GROUP_MASKx of the corresponding device action bootstrap register for ACTION_x must be non-zero. That is, they must have at least one common bit set at the same position in the 32-bit register.

When these 4 conditions are met for at least one action signal ACTION_x, then the device internally asserts the requested action and returns an ACTION_ACK packet. As these conditions could be met on more than one action, the device could assert more than one action signal in parallel. When one of the 4 conditions is not met for any supported action signal, then the device ignores the ACTION_CMD request and does **not** even return an ACTION_ACK.

9.8.1 ACTION_CMD examples

The following examples illustrate the behavior of the ACTION_CMD message in various scenarios. The figure below shows 4 different ACTION_CMD requests issued by a primary or secondary application. Content of the ACTION_CMD packet is illustrated on the left side of the figure.

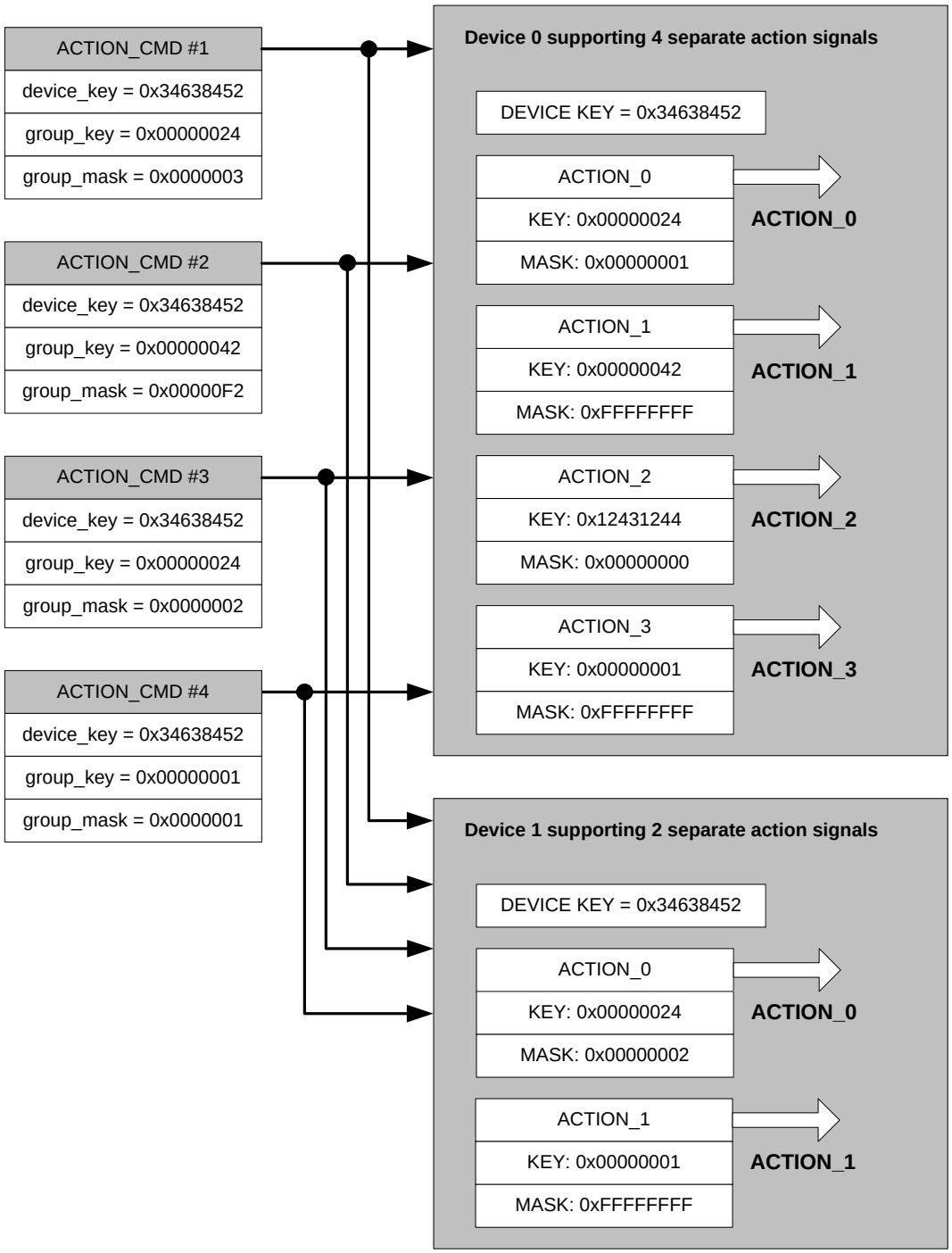


Figure 9-8: ACTION_CMD examples

The content of the ACTION_CMD request must be examined against the 4 conditions listed in Table 9-1 for each supported action signal.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

For the first request (ACTION_CMD #1):

	ACTION_CMD #1	Device 0			Device 1
		ACTION_0	ACTION_1	ACTION_2	ACTION_0
device_key	0x34638452	0x34638452	0x34638452	0x34638452	0x34638452
group_key	0x00000024	0x00000024	**0x00000042**	**0x12341244**	0x00000024
group_mask	0x00000003	0x00000001	0xFFFFFFFF	**0x00000000**	0x00000002

Device 0 receives the request and looks for the 4 conditions.

1. a primary control channel is open
2. device_key matches
3. group_key matches
4. Logical AND-wise comparison of requested group_mask is non-zero

All 4 conditions are met only for ACTION_0, thus the device immediately sends back an ACTION_ACK with GEV_STATUS_OK and asserts the internal signal represented by ACTION_0.

The same steps are followed by Device 1. Only the group_mask is different, but nevertheless the logical bitwise AND operation produces a non-zero value, leading to the assertion of ACTION_0 by Device 1 and the transmission of an acknowledge.

For the second request (ACTION_CMD #2):

	ACTION_CMD #2	Device 0			Device 1
		ACTION_0	ACTION_1	ACTION_2	ACTION_0
device_key	0x34638452	0x34638452	0x34638452	0x34638452	0x34638452
group_key	0x00000042	**0x00000024**	0x00000042	**0x12341244**	**0x00000024**
group_mask	0x000000F2	**0x00000001**	0xFFFFFFFF	**0x00000000**	0x00000002

Looking for the 4 conditions, Device 0 will assert ACTION_1 while Device 1 will not assert any signal because the group_key condition is not met. Therefore, Device 1 ignores the request and does not send back an acknowledge.

For the third request (ACTION_CMD #3):

	ACTION_CMD #3	Device 0			Device 1
		ACTION_0	ACTION_1	ACTION_2	ACTION_0
device_key	0x34638452	0x34638452	0x34638452	0x34638452	0x34638452
group_key	0x00000024	0x00000024	**0x00000042**	**0x12341244**	0x00000024
group_mask	0x00000002	**0x00000001**	0xFFFFFFFF	**0x00000000**	0x00000002

In the third example, the group_mask and group_key of Device 0 does not match with ACTION_CMD #3 for any of the ACTION_0 to ACTION_2. Therefore Device 0 ignores the request. Device 1 asserts ACTION_0 since the 4 conditions are met.

The ACTION_CMD is flexible enough to accommodate "simultaneous" triggering of the same functional action in functionally different devices.

For instance, let's assume the software trigger of Device 0 can only be associated to its ACTION_3 and that the software trigger of Device 1 can only be associated to its ACTION_1. The ACTION_CMD defined in this specification enables an ACTION_CMD to trigger the same functional action provided that their respective action group key and masks are set in order to meet the condition of Table 9-1.

For the fourth request (ACTION_CMD #4):

	ACTION_CMD #4	Device 0	Device 1
		ACTION_3	ACTION_1
device_key	0x34638452	0x34638452	0x34638452
group_key	0x00000001	0x00000001	0x00000001
group_mask	0x00000001	0xFFFFFFFF	0xFFFFFFFF

In this case, Device 0 asserts ACTION_3 and Device 1 asserts ACTION_1 since the conditions of Table 9-1 are met. As a result of this, the software trigger of both devices can be "simultaneously" triggered even though they are associated to different action numbers.

9.9 Primary Application Switchover

For systems where redundancy and fault recovery are required, it is often necessary for a second application to take control over a device that is already under the control of a primary application. This may be needed in the case where the primary application, or part of it, becomes irresponsive or crashes and quick recovery of devices is mandatory. It can also be required for an other application to take over when dictated by an higher level system management entity.

It is very important to indicate that it is outside the scope of this specification to describe when an application should take control over another one and how another application should be notified to do so. These are left as higher level system integration considerations. It is up to the system architect to define the communication mechanisms between various applications and when the switchover needs to occur. What is critical is that this version of the specification makes application switchover possible.

Primary application switchover is made possible by using the control access with switchover enabled access mode. By connecting to a device using this mode, a primary application enables another application to take control provided that it has the right credentials. This means that the other application needs to know the key required to grant the switchover. This key is held by the Control Switchover Key write-only registers. When a device has granted control access with switchover enabled, another application can request control over the device by performing a write access to the CCP register. One field of the CCP register enables the other application to provide a 16-bit data value to use in the switchover authentication process. If the value provided matches the value held by the Control Switchover Key register, then the switchover is granted. This authentication process provides a minimal protection to prevent another application to take control over a device by mistake.

In order to notify the primary application that a switchover has occurred, a device can optionally be configured to send an event (GEV_EVENT_PRIMARY_APP_SWITCH) to make the primary application aware that it lost control of the device. In this case, the device sends the event before granting access to the new primary application. This is provided that the primary application has instantiated a message channel and enabled the GEV_EVENT_PRIMARY_APP_SWITCH event. A device issuing this event should not request an acknowledge since the message channel may be under the control of a new primary application by the time the previous primary application acknowledges the packet. It should be noted that a primary application switchover doesn't close the message channel. It is up to the new primary application to manage it.

The pseudo-code algorithm below presents how a device reacts to a write access to the CCP register. It highlights the rules that must be met in order for a primary application switchover to occur.

```
If an application writes to the CCP register then
    If the device access mode is "open access" then
        Grant the requested access mode.
    Else
        If the write comes from the primary application then
            Grant the requested access mode.
        Else
            If the device is in control access with switchover enabled then
                If the control_switchover_key field matches the value set in
                the Control Switchover Key register then
                    Grant the primary application switchover with the requested
                    access mode.
                Else
                    Deny the request.
                End if
            Else
                Deny the request.
            End if
        End if
    End if
```

```
End if
```

9.9.1 Primary Application Switchover Setup Example

This section provides guidelines regarding how primary application switchover can be setup on a device. In the process described below, it is assumed that the device is not under the control of any application at the beginning of the following procedural steps.

1. The primary application requests, and gets granted, exclusive access.
2. The primary application verifies that the device supports primary application switchover by consulting the GVCP Capability register.
3. The primary application configures the Control Switchover Key register.
4. The primary application requests, and gets granted, control access with switchover enabled. Please note that this is done without closing the control channel.
5. The device is now setup for primary application switchover. Another application that has the right credentials, i.e. that knows the key, can request, and gets granted, device control.

It should be noted that this section provides best usage guidelines but one could follow different steps. Its intent is not to cover all possible scenarios. For instance, one may decide that it is not necessary to consult the GVCP Capability register to determine if primary application switchover is supported by a device if it is already known that the device supports this feature.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

10 Stream Channel

A stream channel enables data transfer from a GVSP transmitter to a GVSP receiver. This transfer uses the GigE Vision Streaming Protocol (GVSP).

[R10-1s] A product supporting GVSP streaming MUST support at least one stream channel.

[R10-2s] A product supporting GVSP streaming MUST support stream channel sequentially starting from index 0. It is not allowed to leave a gap in the index of supported stream channels.

A product supporting GVSP streaming may support up to 512 stream channels. The stream can transport any data type, such as images, histogram, image statistics, etc. Each stream channel can be a transmitter (data source) or a receiver (data sink) but cannot be both. In other words, the direction of the stream channel cannot be programmed. Bootstrap register bits capture the direction of each stream channel.

Note: In order to promote compatibility with applications compliant with the previous versions of this specification, it is recommended, for transceiver devices, that stream channels transmitting data are defined in the register map before receiving stream channels. In other words, stream channels transmitting data use a lower index in the bootstrap registers list than the receiving stream channels.

10.1 Stream Channel Registers

A given stream channel can be a transmitter or a receiver. The following registers are associated to transmitter and receiver stream channels:

1. Stream Channel Port register (SCP)
2. Stream Channel Packet Size register (SCPS)
3. Stream Channel Destination Address register (SCDA)
4. Stream Channel Configuration Registers (SCCFGx)

In addition to this, the following register is associated to transmitter stream channels.

1. Stream Channel Packet Delay register (SCPD)
2. Stream Channel Source Port register (SCSP)

Note: While the Stream Channel Packet Size register (SCPS) for transmitters is used to configure the size of the packet to use when streaming data, it is used to report the maximum packet size supported by a receiver.

Refer to the bootstrap registers section for a complete description of the fields in these registers.

10.2 Opening a Stream Channel

A primary application opens a stream channel by writing the host port to the SCPx register and the destination IP address to the SCDAx register. Only the primary application is allowed to configure stream channels.

[O10-3st] A GVSP transmitter SHOULD use any dynamic port number as the UDP source port for a given stream channel.

10.3 Closing a Stream Channel

[R10-4ca] A primary application MUST close a stream channel by writing a 0 to the SCPx register.

Since SCPx is used to open and close the stream channel, it has to be the last stream channel register accessed by the application upon opening the channel and the first register accessed when closing the stream.

A GVSP transmitter should stop streaming data as soon as possible after the stream channel is closed. When closing a stream channel, it is possible that the last block of data is not totally sent on the link. But any packet has to be sent in its entirety. It is up to the GVSP receivers to correctly handle this situation.

[R10-5st] A GVSP transmitter MUST NOT send an incomplete streaming packet.

The proper way to stop the stream on a nice block boundary is by using device-specific registers. For instance, a camera offers AcquisitionStart and AcquisitionStop registers.

A GVSP transmitter is not required to send the data trailer when the stream channel is closed during transmission of a given block_id packets.

10.4 Packet Size

GVSP does not impose any restrictions on packet size. It is up to the management entity of the system and GVSP transmitters to negotiate a suitable packet size. One way to determine the maximum packet size that does not lead to IP fragmentation is by sending stream test packets (the “do not fragment bit” should be set by the application for test packets).

This can be achieved through the SCPSx bootstrap register of GVSP transmitters: bit 16 to 31 provides the packet size while bit 0 indicates to fire a test packet. Bit 1 must be set to indicate IP fragmentation is not allowed by the application. All these bits are written at once to SCPSx. This results in the GVSP transmitter sending a single test packet with the total size (including IP and UDP headers) equal to the requested maximum packet size.

To discover the optimal packet size, the management entity sets up the stream channel by writing into SCPx, SCPDx and SCDAx registers. It can then perform a simple iterative binary search (or any other search) by writing into the SCPSx register with various packet sizes. If the test packet does not reach the GVSP receivers, then the management entity can reduce the packet size. If the test packet reaches the GVSP receivers, then the management entity can increase the packet size. This way, the management entity can converge to the optimal packet size. In most scenarios, the larger the packet size, the lower the overhead of packet transmission.

Note: The management entity of the system can easily determine if the test packet reached the destination in a point-to-point configuration when the management entity is the application and the GVSP receiver. However, it could be more complicated for the management entity to determine if the test packet reached its destination(s) in other situations. It is beyond the scope of this specification to define how a management entity determines that a test packet is received by a GVSP receiver. It is left as a quality of implementation.

[R10-6st] If the packet size requested in SCPSx is not supported by the GVSP transmitter, then it MUST NOT send a test packet when it is requested to do so.

The management entity will need to find a packet size supported by the GVSP transmitter and by all network elements between the GVSP transmitter and the GVSP receivers.

Note: This is not a bullet-proof method of finding the maximum packet size when multiple routes are available from the GVSP transmitter to the GVSP receivers. Different routes might very well support different MTU. One cannot guaranty all data packets will travel using the same route as the test packet.

10.5 Multicasting

A stream channel might use multicasting if the data stream must be sent to multiple locations. Multicasting is activated by setting the destination IP address of the GVSP transmitter to a multicast group in the SCDA register. When multicasting, any destination is allowed to send a PACKETRESEND command on the control channel.

Note: When multicasting, the destination IP address might range from 239.0.0.0 to 239.255.255.255 for Administratively-scoped (local) multicast addresses and from 224.0.1.0 to 238.255.255.255 for Globally-scoped (Internet-wide) multicast addresses. Refer to RFC3171 (IANA Guidelines for IPv4 Multicast Address Assignments) for more information.

10.6 Impact of Multiple Network Interfaces

GVSP allows multiple network interfaces for stream channels to increase the bandwidth available for streaming.

[CR10-7s] When multiple network interfaces are supported, the specific interface to use for the stream channel MUST be specified in the SCPx register.

It is up to the primary application to correctly handle the network interfaces available on the device using the bootstrap registers.

Bootstrap registers might optionally be available to report the current link speed of each network interface.

10.7 Traversing Firewalls or Network Address Translation Devices

Due to the unidirectional nature of GVSP traffic, it is common for the streaming channel data to be blocked by firewalls or other types of network since the incoming traffic cannot be matched against an outgoing packet. To address this, the SCSPx registers are designed to allow the application associated to a GVSP

receiver to create a simulated bidirectional traffic conversation between the GVSP receiver and the GVSP transmitter streaming channel by informing the application of the remote UDP port. This information would otherwise be unknown until the first streaming packet arrives, which may never happen if it was blocked. Additionally, this information may be used in other ways to help configure software and/or networking equipment.

The stream channel capability register (at address 0x092C) can be used to query if SCSPx registers are supported by this device.

[CR10-8st] If a product supports SCSPx for one GVSP transmitter, then it MUST support it for all its GVCP transmitter stream channels.

In a camera to control and receiving application use case, the suggested usage of SCSPx is to query it after opening the image stream via SCDAx and SCPx but before starting the camera's flow of image data. If desired, the application can query it earlier and cache it off (only if it is non-zero) so that it does not need to query it each time the streaming channel is opened. Refer to [CR27-45cd] for proper behavior of SCSPx.

Next, the application can send a UDP packet from the same source port it programmed into SCPx to the port specified by SCSPx. The content of this packet is ignored, but the suggested payload should be smaller than the maximum size of a GVCP command packet. This operation should be realized on all the open stream channels.

At this point the application can continue starting the acquisition on the camera. Depending on the configuration of the firewall or other impediment, the application may choose to continue periodically sending packets similar to the first one for the duration of the streaming session. The suggested interval is 30 seconds and could be configurable. Different firewalls might require adjustment to this interval.

[CR10-9ca] When SCSPx is supported by the application, the application MUST not send packets to the port specified by SCSPx unless it has the stream channel opened and successfully reads a non-zero value back from SCSPx.

The application should not expect the device to answer the packet it sends to the GVSP transmitter stream channel. But this mechanism provides a way to open-up the UDP port in the firewall. The implementation is left as a quality of implementation and is likely to vary with different firewall types and operating systems.

Note that the same mechanism is also available for the message channel.

10.8 Unconditional Streaming

In a number of video distribution systems over Ethernet, especially the ones involving simultaneous GVSP receivers, it is often mandatory to make sure that GVSP transmitters can continue streaming no matter what happens with their primary application or with the state of the network. For instance, it might be required that a device continues streaming even if its primary application crashes or is closed and moved to a different host computer.

This version of the specification enables such scenarios by enabling GVSP transmitters, supporting this capability, to be configured to continue streaming independently of the state of the control channel or any ICMP messages received by the device associated to a GVSP transmitter.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

It is then possible for an application to configure GVSP transmitters for unconditional streaming by setting a bit in the Stream Channel Configuration registers (SCCFGx) provided that the capability is supported by the GVSP transmitters in question. When configured in this mode, a GVSP transmitter will not stop if the control channel of the device is closed. Moreover, it will not stop streaming if its associated device receives ICMP destination unreachable messages. In other words, the GVSP transmitter continues streaming as long as dictated by its primary application. The Stream Channel Capability registers (SCCx) indicate if GVSP transmitters can support unconditional streaming.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

11 Message Channel

A message channel allows a device to send asynchronous messages to the application. For instance, a device may want to signal that a camera trigger has been detected.

A message channel is very similar to a control channel, but requests are emitted in the opposite direction. The device always initiates transactions on the message channel. Therefore, the message channel headers are identical to the control channel headers. Support of message channel by a device is optional.

The packet flow sequence on a message channel is:

1. Device sends a message packet
2. Application receives the message packet
3. Application process the message
4. Application sends the acknowledge packet (if acknowledge is requested)
5. Device receives the acknowledge packet (if acknowledge is requested)

[CR11-1cd] When a device supports a message channel, the req_id field on the message channel MUST increment from one message to the next (except for retransmission). When req_id wraps-around to 0, then its value must be set to 1 (0 is invalid for req_id). Therefore, req_id 65535 gets incremented to 1.

This allows the application to detect if a UDP message has been lost, even when no acknowledge is requested.

11.1 Message Channel Registers

When supported, the message channel provides the following registers:

- Message Channel Port register (MCP)
- Message Channel Destination Address register (MCDA)
- Message Channel Transmission Timeout register (MCTT)
- Message Channel Retry Count register (MCRC)
- Message Channel Source Port register (MCSP)

Refer to the bootstrap registers section for a complete description of the fields in these registers.

11.2 Opening the Message Channel

A primary application opens the message channel by writing the destination IP address to the MCDA register and then the host port to the MCP register. Only the primary application is allowed to open the message channel since it must have write access privilege.

[CO11-2cd] When a device supports a message channel, it SHOULD use any dynamic port number as the UDP source port for the message channel.

[CR11-7ca] When an application supports the message channel, it MUST acknowledge messages on the message channel using the UDP source port of the incoming packet as the UDP destination port for the acknowledge packet when such an acknowledge is requested.

11.3 Closing the Message Channel

[CR11-3ca] When a message channel is supported, a primary application MUST close it by writing a 0 to the MCP register.

Since MCP is used to open and close the message channel, it has to be the last message channel register accessed by the application upon opening the channel and the first register accessed when closing the channel.

When closing a message channel, it is not allowed for a message packet to be sent incomplete.

[CR11-4cd] When a message channel is supported, if a packet is being transmitted by the device while its message channel is being closed, then this message MUST be completely transmitted.

[CR11-5ca] When a message channel is supported, if a message is received by the application when its message channel is closed, then it MUST be silently discarded.

11.4 Asynchronous Events

A device can send asynchronous event messages on the message channel when the channel is opened. Each event is represented by a 16-bit ID. This document defines two categories of events:

- GigE Vision standard events (value 0 to 36863)
- Device-specific events (value 36864 to 65535)

Device-specific registers are used to enable/disable those events. XML device description files describe the events, its index and the register to enable/disable. Note that even the standard events are enabled/disabled using device-specific registers.

11.5 Multicasting

A message channel might use multicasting if the events must be sent to multiple locations.

[CR11-6cd] When a message channel is supported, in the case of event message multicast, acknowledge packets are not permitted (the acknowledge bit of the message MUST be cleared by the device).

Multicasting is activated by setting the destination IP address of the device to a multicast group in the MCDA register.

Note: When multicasting, the destination IP address might range from 239.0.0.0 to 239.255.255.255 for Administratively-scoped (local) multicast addresses and from 224.0.1.0 to 238.255.255.255 for Globally-

scoped (Internet-wide) multicast addresses. Refer to RFC3171 (IANA Guidelines for IPv4 Multicast Address Assignments) for more information.

11.6 Traversing Firewalls or Network Address Translation Device

The Message Channel suffers from the same issue related to firewalls than the stream channel. The same mechanism available to help open-up UDP ports for stream channels is available to the message channel.

The message channel capability register (at address 0x0930) can be used to query if the MCSP register is supported by this device.

The application's suggested usage of MCSP is to query it after opening the message channel via MCDA and MCP.

Next, the application can send a UDP packet from the same source port it programmed into MCP to the port specified by MCSP. The content of this packet is ignored, but the suggested payload should be smaller than the maximum size of a GVCP command packet.

Depending on the configuration of the firewall or other impediment, the application may choose to continue periodically sending packets similar to the first one for the duration of the session. The suggested interval is 30 seconds and could be configurable. Different firewalls might require adjustment to this interval.

[CR11-8ca] When MCSP is supported by the application, the application MUST not send packets to the port specified by MCSP unless it has the message channel opened and successfully reads a non-zero value back from MCSP.

The application should not expect the device to answer the packet it sends to the device message channel. But this mechanism provides a way to open-up the UDP port in the firewall. The implementation is left as a quality of implementation and is likely to vary with different firewall types and operating systems.

Note that the same mechanism is also available for stream channels.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

12 Device with Multiple Network Interfaces

A device may have multiple network interfaces.

[R12-1cd] A device MUST have at least one network interface.

[R12-2cd] A device might support up to 4 different network interfaces. In this case, each network interfaces MUST execute the IP configuration procedure independently.

The number of supported network interface is reported by register "Number of network interfaces" at address 0x0600 in bootstrap registers memory space. Other bootstrap registers in the 0x0600 to 0x07FF range are used to store IP configuration information of each network interface.

Each network interface has an index associated to it (from 0 to 3). Interface #0 is the main interface to control the device. It is the only interface supporting GVCP (for control and message channels). For instance, device discovery is always performed on interface #0. Other network interfaces are only used to support additional stream channels. This also means that these other interfaces only output or receive GVSP packets; they are not supporting GVCP.

[CR12-3cd] When a device supports multiple network interfaces, then it MUST provide independent storage for bootstrap registers related to IP configuration on each interface.

[O12-4cd] The bootstrap registers allocated to unsupported network interfaces cannot be accessed and SHOULD return a GEV_STATUS_INVALID_ADDRESS status code (same status code used to access any unsupported register).

[R12-5cd] It is only possible to use FORCEIP message with interface #0. Other interfaces MUST use Persistent IP, DHCP or LLA to get their IP configuration.

It is recommended to allow any supported stream channel to be associated to any network interface through the SCPx register, but it is acceptable to hard-code a stream channel to a specific network interface. In the later case, the "network interface index" field is read-only.

When multiple network interfaces are supported, it is recommended the application uses the same stream packet size (SCPSx) on each network interfaces.

Note: A product is also allowed to present each network interface as a different GigE Vision device. In this case, suitable resources (such as bootstrap registers) must be provided on each interface, and each interface is considered independent from the other. This is equivalent to grouping multiple devices together into a single unit.

12.1 Control Channel

[R12-6c] The control channel MUST always be instantiated on interface #0. The application MUST take control of the device using that device interface.

[R12-7cd] A device MUST NOT answer GVCP requests coming on an interface different than interface #0. In this case, the message is silently discarded.

12.2 Stream Channels

[R12-8st] When streaming data, the GVSP transmitter MUST use the network interface specified in the SCPx register for each stream channel.

[R12-11sr] When receiving data, a GVSP receiver MUST use the network interface specified in the SCPx register for each stream channel.

[R12-9ca] An application MUST send the PACKETRESEND_CMD on interface #0 as it is the only one supporting GVCP. In this case, the "stream channel index" field will tell the device on which network interface to resend the packet since it maps to SCPx.

12.3 Message Channel

[CR12-10c] If supported, the message channel MUST be instantiated on interface #0.

13 GVCP Headers

GVCP defines 2 types of header:

1. command header
2. acknowledge header

All protocol headers are defined to be 32-bit naturally aligned. Their representation is always big-endian.

In the following section, the nomenclature refers explicitly to the control channel, but a message channel must react in an equivalent manner.

13.1 Command Header

The following figure shows the command message header. Note the figure does not show Ethernet, IP and UDP headers.

[R13-1c] Command Headers MUST respect the layout of Figure 13-1.

[O13-2c] A recipient of a command message SHOULD perform minimal validation on the command header by verifying the presence of the 0x42 key in the first byte of the packet. A packet without this key is not considered to be a GVCP message.

[R13-3c] If the message is not a GVCP message, then it MUST be silently discarded.

[O13-4c] If the command requested in the command field is not supported by the recipient of the command, then the recipient SHOULD return a GEV_STATUS_NOT_IMPLEMENTED status code.

In the previous situation, the acknowledge message value to use to answer the unsupported command is the command message value + 1. The specification always define the acknowledge message value to be the associated command message value + 1. Refer to [R16-1c] for the command and acknowledge values.

[O13-5c] If any other field of the command message header is invalid, then the recipient SHOULD return a GEV_STATUS_INVALID_HEADER status code.

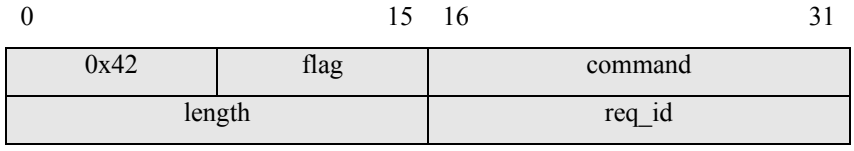


Figure 13-1: Command Message Header

0x42	8 bits	Hard-coded key code value used for message validation.
flag	8 bits	Upper 4 bits (bit 0 to bit 3) are specific to each GVCP commands. Lower 4 bits (bit 4 to bit 7) are common to all GVCP commands. All unused bits must be set to 0. bit 7 – ACKNOWLEDGE: SET = requires the recipient of this message to send an acknowledge message. CLEAR = recipient of this message shall not send an acknowledge message. Typically used for UDP. Some commands might not require acknowledge. This can be used to limit the amount of bandwidth for control. If this bit is cleared and the command normally requires an acknowledge, then this bit has precedence and NO ack message is sent (for instance, reading a register to reset the heartbeat would not send back the register value to the application). Acknowledge messages can be used to enforce flow-control. bit 6 – Reserved, set to 0. bit 5 – Reserved, set to 0. bit 4 – Reserved, set to 0. bit 0 to 3 – Specific to each command. Set to 0 if the command does not use them.
command	16 bits	Command message value (see dictionary section). Commands from 0 to 32765 are reserved for GigE Vision. Others are available for customization (device-specific).
length	16 bits	Number of valid data bytes in this message, not including this header. This represents the number of bytes of payload appended after this header.
req_id	16 bits	This field is the request ID generated by the application. The device copies this value in the corresponding acknowledges's ack_id field. The req_id is used to match a command with its acknowledge. The request ID must never be set to 0. The application must use sequential request ID where the value is incremented by 1 between each message (except when req_id wraps back to 0, then it must be set to 1as 0 is not a valid value).

13.2 Acknowledge Header

The following figure shows the acknowledge message header. Note the figure does not show Ethernet, IP and UDP headers.

[R13-6c] Acknowledge Headers MUST respect the layout of Figure 13-2.

[O13-7c] The recipient of an acknowledge message SHOULD perform minimal validation of the acknowledge header. This minimally includes the validation of the ack_id against the req_id sent previously, but it may also include the validation of the acknowledge field against a value listed in this specification.

~~[O13-8a] The GEV_STATUS_INVALID_HEADER status code SHOULD be returned by the application to upper software layers if the header is invalid.~~

Note the emitter of the acknowledge message is allowed to set the length field to 0 when it returns a suitable error code in the status field if no data payload is appended to this acknowledge message header.

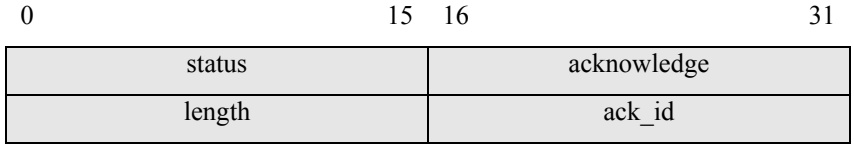


Figure 13-2: Acknowledge Message Header

status	16 bits	Status of the requested operation (see status code section)
acknowledge	16 bits	Acknowledge message value (see dictionary section).
length	16 bits	Number of valid data bytes in this message, not including this header. This represents the number of bytes of payload appended after this header.
ack_id	16 bits	This field is the acknowledge ID and has the same value as the corresponding command req_id.

13.3 Byte Sequencing

[R13-9c] GVCP MUST use network byte order, also known as big-endian.

The sequence of bytes of a message to send is provided by Appendix B of RFC791. The order of transmission of the header data is illustrated in the figure below, from byte 1 to byte 8. It goes from left to right, then from top to bottom. Notice that bit 0 is on the left, while bit 31 is on the right. Any GVCP data payload follows the same sequence when it is transmitted on the network.

The order of transmission of the header and data described in this document is resolved to the byte level. Whenever a diagram shows a group of bytes, the order of transmission of those bytes is the normal order in which they are read in English. For example, in the following diagram the bytes are transmitted in the order they are numbered.

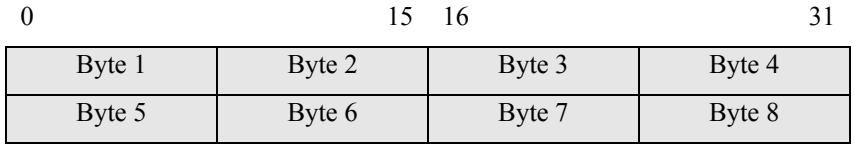


Figure 13-3: Byte Sequencing

Whenever a byte represents a numeric quantity the left most bit in the figure is the most significant bit. That is, the bit labeled 0 is the most significant bit, as shown in next figure.

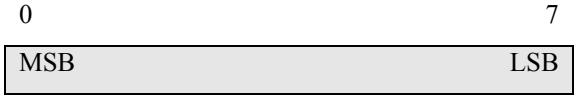


Figure 13-4: Byte sequencing for an 8-bit word

Similarly, whenever a multi-byte field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-byte quantity is transmitted the most significant byte is transmitted first.

For a 16-bit word:

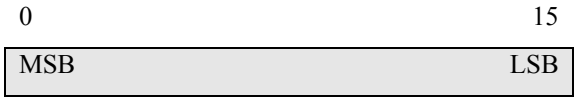


Figure 13-5: Byte sequencing for a 16-bit word

For a 32-bit word:

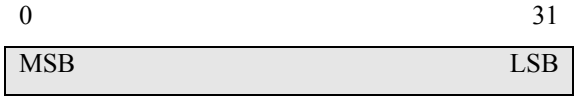


Figure 13-6: Byte sequencing for a 32-bit word

The following figure shows the breakdown of GVCP header into byte fields.

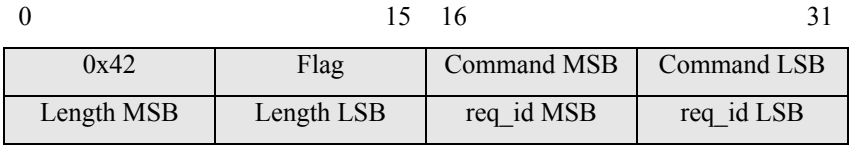


Figure 13-7: GVCP Header Showed in Byte Quantities

Therefore, for GVCP header provided above, the appropriate sequence of bytes on the network is:

Table 13-1: GVCP Header Byte Transmission

Byte	Value
1	Key value 0x42
2	Flag
3	Command MSB
4	Command LSB
5	Length MSB
6	Length LSB
7	req_id MSB
8	req_id LSB

This order of transmission is called network byte order. It is equivalent to big-endian. A little-endian machine must swap multi-byte fields (that is 16 and 32-bit word) to correctly interpret the value. 8-bit fields are not swapped (this includes 8-bit character strings).

The following figure shows the breakdown of a specific GVCP command header into byte fields.

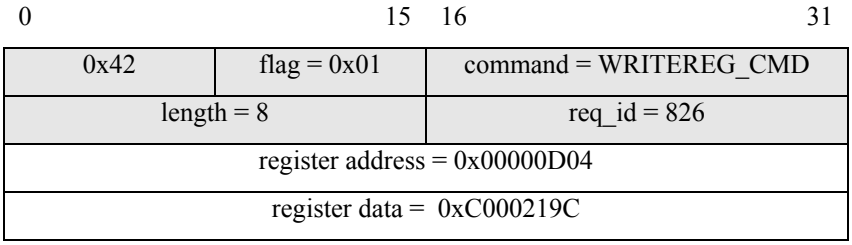


Figure 13-8: Example Command for Byte Ordering

Assume the following parameters:

Flag	0x01	bit 7 is SET, requiring the recipient of this message to send an acknowledge message. Since bit 7 is the least significant bit, the resulting value will be 0000 0001b (0x01).
Command	0x0082	WRITEREG_CMD
Length	0x0008	8
Req_id	0x033A	826
Register Address	0x00000D04	SCPS0
Register Data	0xC000219C	bit 0 and 1 are SET. This device will fire one unfragmented test packet of the size specified by bits 16-31. Since bit 0 is the most significant bit, the resulting value for the bit flags will be 1100 0000 0000 0000 0000 0000 0000 0000b (0xC0000000). The packet size will be 8604 bytes. Since bit 16 is the most significant bit for packet size field, the resulting value for the packet size will be 0000 0000 0000 0000 0010 0001 1001 1100b (0x0000219C). Combining the bit fields, we get the result 1100 0000 0000 0000 0010 0001 1001 1100b (0xC000219C).

Therefore, for GVCP command provided above, the appropriate sequence of bytes on the network is:

Byte	Definition	Value
1	Key value	0x42
2	Flag	0x01
3	Command MSB	0x00
4	Command LSB	0x82
5	Length MSB	0x00
6	Length LSB	0x08
7	req_id MSB	0x03
8	req_id LSB	0x3A
9	address MSB	0x00
10	address	0x00
11	address	0x0D
12	address LSB	0x04
13	data MSB	0xC0
14	data	0x00
15	data	0x21
16	data LSB	0x9C

Note: A device internal representation of each 32-bit register might be either big or little endian. This must be indicated in the Device Mode register. But GVCP messages must have the data in big-endian (network byte order).

Note: Implementations using the socket API can rely on the following standard byte sequencing functions: **ntohl**() to convert a 32-bit word from network to host byte order, **htonl**() to convert a 32-bit from host to network byte order, **ntohs**() to convert a 16-bit word from network to host byte order, and **htons**() to convert a 16-bit word from host to network byte order. 8-bit words do not require any conversion.

14 Control Channel Dictionary

Messages in this category are sent on the Control Channel, from the application to the device. Acknowledges are sent in the opposite direction. Before executing any command, the device verifies if the requester has the necessary privilege.

[R14-1cd] If the requester of a control message does not have the necessary privilege, the device MUST return a GEV_STATUS_ACCESS_DENIED in the status field of the acknowledge message.

14.1 DISCOVERY

[R14-2cd] The DISCOVERY command MUST be supported by all devices.

The DISCOVERY command is required to support device initialization. No privilege is required to execute this command. A device has to answer this request from any application (primary and secondary) at all time.

14.1.1 DISCOVERY_CMD

To enumerate devices, an application sends a DISCOVERY_CMD message.

[R14-3c] DISCOVERY_CMD MUST follow the layout provided by Figure 14-1.

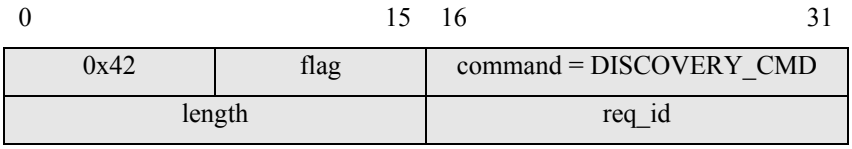


Figure 14-1: DISCOVERY_CMD message

flag	8 bits	bit 4 to 7 – Use standard definition from GVCP header bit 3 – Allows broadcast of acknowledge bit 0 to 2 – Reserved, set to 0.

Bit 3 of the "flag" field can be set by an application to indicate a device is allowed to broadcast its DISCOVERY_ACK message if the conditions mentioned in the DISCOVERY_ACK section below are met. When this bit is cleared, the device cannot broadcast its acknowledge message.

[R14-62ca] For a DISCOVERY_CMD message, the ACKNOWLEDGE bit (bit 7) of the flag field MUST be set.

14.1.2 DISCOVERY_ACK

[R14-4cd] When a device receives a DISCOVERY_CMD command from an application, it MUST reply with a DISCOVERY_ACK command if it is on same subnet as the application.

[R14-5cd] A device MUST unicast the acknowledge directly to the application if they both reside on the same subnet.

Note: If its IP configuration is not properly configured, it might not be possible for the device to adequately validate the subnet mask of the application.

A device may optionally broadcast its DISCOVERY_ACK when the following 2 conditions are met:

1. the DISCOVERY_CMD was not received on the device subnet
2. bit 3 of the "flag" field of DISCOVERY_CMD was set by the application.

The above scenario typically happens when a device is configured with a Persistent IP address that does not match the subnet of the application. A device might decide not to broadcast the DISCOVERY_ACK even if the above 2 conditions are met. This is left as a quality of implementation.

[O14-6cd] If the "Discovery ACK Delay" bootstrap register is supported, the device SHOULD wait for a randomized delay with a uniform distribution from 0 up to the value in the "Discovery ACK Delay" bootstrap register before sending the DISCOVERY_ACK. Otherwise, the device SHOULD wait for a randomized delay with a uniform distribution from 0 to 1 second before sending the DISCOVERY_ACK.

This way, if a huge number of devices receive the broadcast message simultaneously, they are less likely to saturate the application.

[R14-7c] DISCOVERY_ACK MUST follow the layout provided by Figure 14-2.

0 – 15	16 – 31
status	answer = DISCOVERY_ACK
length	ack_id
spec_version_major	spec_version_minor
Device mode	
reserved = 0	Device MAC address (high)
Device MAC address (low)	
IP config options	
IP config current	
Reserved = 0	
Reserved = 0	
Reserved = 0	
Current IP	
Reserved = 0	
Reserved = 0	
Reserved = 0	
Current subnet mask	
Reserved = 0	
Reserved = 0	
Reserved = 0	
Default gateway	
manufacturer name [32 bytes]	
model name [32 bytes]	
device version [32 bytes]	
manufacturer-specific information [48 bytes]	
serial number [16 bytes]	
user-defined name [16 bytes]	

Figure 14-2: DISCOVERY_ACK message

The fields in this acknowledge, except for the GVCP header, identically match to the first few bootstrap registers. Device discovery is always performed on the first network interface (interface #0) since it is the only one to support GVCP. Refer to bootstrap registers for the definitions of these fields.

Note: All strings are zero-padded to fill the allocated space. Strings must be NULL-terminated. Therefore at least one byte is lost to store the NULL character. When serial number or user-defined name are not supported, they must be set to an all NULL string (filled with 0).

[R14-8cd] When an error is generated during the processing of the DISCOVERY_CMD, the device MUST reply with an appropriate status code set in the DISCOVERY_ACK header. In this case, the "length" field must be set to 0 and the acknowledge message only contains the header.

The above requirement allows an application to retrieve the cause of the error.

14.2 FORCEIP

In some situations, it might be useful for the application to "force" an IP address in a device. For instance, when a device is using an unidentified Persistent IP, then it might be difficult for the application to communicate with the device since the subnet of the application would likely be different from the subnet of the device. This issue does not happen with DHCP and LLA since the IP address is negotiated.

To solve this problem, GVCP provides a mechanism to force a static IP address into the device, as long as the application knows the MAC address of the device.

[R14-9cd] The FORCEIP command MUST be supported by all devices.

The FORCEIP command is required to force an IP address into the device identified by the MAC address. It offers provision to set the subnet mask and default gateway associated to this IP address.

14.2.1 FORCEIP_CMD

[CR14-10ca] If supported by the application, the FORCEIP_CMD MUST be broadcasted by the application on the GVCP port. It must contain the MAC address of the device to access.

[R14-11cd] If the MAC address field of FORCEIP_CMD does not match the device's MAC address, then the message MUST be silently discarded by the device.

[R14-12cd] A device MUST process FORCEIP_CMD only when there is no application with exclusive or control access (without or with switchover enabled). Otherwise, this request MUST be silently discarded.

The above requirement indicates that FORCEIP_CMD is not coming from the primary application. This is necessary to allow this command to be used to recover a device with an unknown IP address.

This request can be used to accomplish two different actions on the device with the specified MAC address:

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

[R14-13cd] If the static IP field of FORCEIP_CMD is 0, then the device MUST restart its IP configuration cycle without sending a FORCEIP_ACK back to the application.

[R14-14cd] If the static IP field of FORCEIP_CMD is different from 0, then the device MUST change its IP address to the value specified in the "static IP" field and only send back a FORCEIP_ACK (if requested) when the newly assigned static IP is effective. If assigning the static IP address requires the device to reset its communication stack and internal state, then static IP address MUST be maintained after this reset.

[R14-15cd] FORCEIP_CMD MUST follow the layout of Figure 14-3.

[CR14-76ca] If supported, FORCEIP_CMD MUST follow the layout of Figure 14-3.

0	15	16	31
0x42	flag	command = FORCEIP_CMD	
length		req_id	
reserved		MAC (high)	
MAC (low)			
Reserved = 0			
Reserved = 0			
Reserved = 0			
Static IP			
Reserved = 0			
Reserved = 0			
Reserved = 0			
Static subnet mask			
Reserved = 0			
Reserved = 0			
Reserved = 0			
Static default gateway			

Figure 14-3: FORCEIP_CMD message

FORCEIP_CMD	
flag	bit 4 to 7 – Use standard definition from GVCP header bit 3 – Allows broadcast of acknowledge (version 1.1 and above only) bit 0 to 2 – Reserved, set to 0.
MAC	MAC address of the device to access
Static IP	Static IP address to force into the device
Static subnet mask	Subnet mask to use with the Static IP address
Static default gateway	Default gateway to use with the Static IP address

For device supporting GigE Vision specification 1.1 and above, bit 3 of the "flag" field can be set by an application to indicate a device should broadcast its FORCEIP_ACK message. This can be useful if the subnet changes during the execution of this command. When this bit is cleared, the device cannot broadcast its acknowledge message. For GigE Vision 1.0 devices, this bit must stay cleared.

14.2.2 FORCEIP_ACK

The FORCEIP_ACK is sent back by the device after the forced static IP address has been assigned. The status field is used to signal any problem.

[R14-16cd] FORCEIP_ACK MUST follow the layout of Figure 14-4.

0	15	16	31
status		answer = FORCEIP_ACK	
length		ack_id	

Figure 14-4: FORCEIP_ACK message

This message must be sent with the newly forced IP address as the source address in the IP packet. This acknowledge should be broadcasted if the broadcast bit is set in the flag field of the FORCEIP_CMD.

14.3 READREG

[R14-17cd] The READREG command MUST be supported by all devices.

The application can issue a READREG message to read a device register. Multiple reads can be concatenated in one message, as long as the total packet size is lower or equal to 576 bytes. Therefore a maximum of 135 registers can be read with one message if the device supports concatenation of operations (36 bytes are used by the various headers). The number of read operations to perform can be deduced from the length field of the command header. It is equal to "length / 4".

[R14-18cd] The device MUST have a capability flag indicating if multiple register reads can be concatenated (bit 31 of GVCP Capability register, at address 0x0934).

[CR14-19cd] When concatenation is supported and multiple register reads are specified, they MUST be executed sequentially by the device.

[CR14-20cd] When concatenation is supported, if an error occurs, register read operation MUST stop at the location of the error. Remaining read operations are discarded and an appropriate error code is returned in the acknowledge message.

The length field of the acknowledge message is used to deduce number of read operations performed successfully (each register provides 4 bytes).

Note: The control protocol assumes the registers are 32-bit wide. It is up to the device to perform a suitable conversion if its internal registers have a different size.

Of particular importance is when an application accesses string registers using READREG. Even though READMEM command is the appropriate method to access string registers, READREG has to be supported and would treat the string register as consecutive 32-bit registers. A little endian device would therefore flip the bytes to convert the register to big-endian in the packet to transmit, as required by [R13-9c].

[R14-21cd] If no application has exclusive access, then the device MUST answer READREG messages coming from the primary or from any secondary application.

[R14-22cd] If an application has exclusive access, then the device MUST only answer READREG messages coming from the primary application. In this case, READREG messages coming from secondary applications are acknowledged with a status code of GEV_STATUS_ACCESS_DENIED and the length field is set to 0 (no data payload).

In order for a secondary application not to affect the behavior of a primary application, it is recommended that read operations do not modify the content of a register. Read-to-clear should be avoided, write-to-clear registers are preferable.

14.3.1 READREG_CMD

[R14-23c] READREG_CMD MUST follow the layout of Figure 14-5.

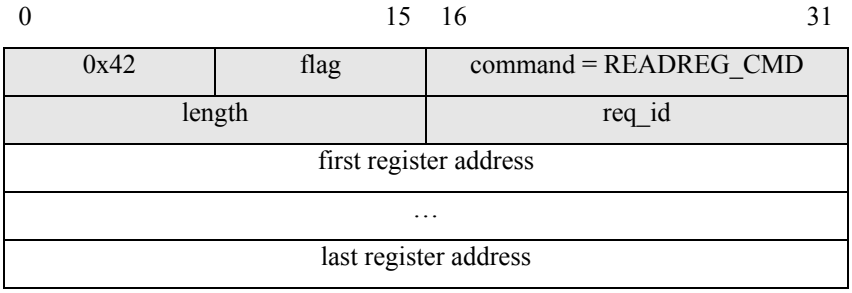


Figure 14-5: READREG_CMD message

READREG_CMD	
flag	ACKNOWLEDGE bit should be set
register address	Address of the data being requested. It must be aligned on a 32-bit boundary (bit 31 and 30 are cleared).

14.3.2 READREG_ACK

The acknowledge message packet is given by the following figure:

[R14-24c] READREG_ACK MUST follow the layout of Figure 14-6.

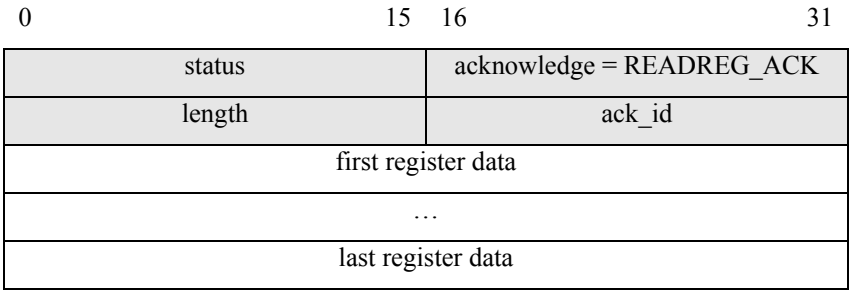


Figure 14-6: READREG_ACK message

READREG_ACK	
register data	Value of the data being requested

[R14-25cd] For READREG_ACK, acknowledge message length MUST reflect the number of bytes that were successfully read. This length MUST be a multiple of 4 bytes. Other registers are not put in the acknowledge message.

This way, the application knows immediately from the message length field how many bytes were successfully read (it can tell the number of registers in the acknowledge payload). An appropriate status code indicating the cause of failure shall be put in the status field.

14.4 WRITEREG

[R14-26cd] The WRITEREG command MUST be supported by all devices.

The application can issue a WRITEREG message to write to a device register. Multiple writes can be concatenated in one message, as long as the total packet size is lower or equal to 576 bytes. Therefore a maximum of 67 registers can be written with one message if the device supports concatenation of operations (36 bytes are used by the various headers). The number of write operations to perform can be deduced from the length field of the command header. It is equal to "length / 8".

[R14-27cd] The device MUST have a capability flag indicating if multiple writes can be concatenated (bit 31 of GVCP Capability register, at address 0x0934).

[CR14-28cd] When concatenation is supported and multiple register writes are specified, they MUST be executed sequentially by the device.

[CR14-29cd] When concatenation is supported, if an error occurs, write operation MUST stop at the location of the error. Remaining write operations are discarded and an appropriate error code is returned in the acknowledge message along with the 0-based index of the write operation that failed within the list provided by the command.

The application will thus know which write operations were executed successfully and which one is associated with the status code.

Note: The control protocol assumes the registers are 32-bit wide. It is up to the device to perform a suitable conversion if its internal registers have a different size.

Of particular importance is when an application accesses string registers using WRITEREG. Even though the optional WRITEMEM command is the appropriate method to access string registers, WRITEREG has to be supported and would treat the string register as consecutive 32-bit registers. A little endian device would therefore flip the bytes to convert the register to big-endian in the packet to transmit, as required by [R13-9c].

Only the primary application is allowed to send a WRITEREG message with the exception of primary application switchover requests.

[R14-30cd] A device MUST answer WRITEREG messages coming from the primary application.

[R14-31cd] A device MUST return a GEV_STATUS_ACCESS_DENIED status code to any secondary application trying to execute this command unless the device is under control access with switchover enabled and another application, with the right credentials, requests control over the device.

When there is no primary application associated to the device, then any application is allowed to write to the CCP register to try to get exclusive or control access (without or with switchover enabled). Once a device is granted exclusive or control access, then no other application can write to CCP. However, it is possible for

[R14-32cd] A device MUST accept WRITEREG_CMD accessing the CCP register from any application when no primary application is registered. This allows the creation of the control channel for the primary application.

14.4.1 WRITEREG_CMD

[R14-33c] WRITEREG_CMD MUST follow the layout of Figure 14-7.

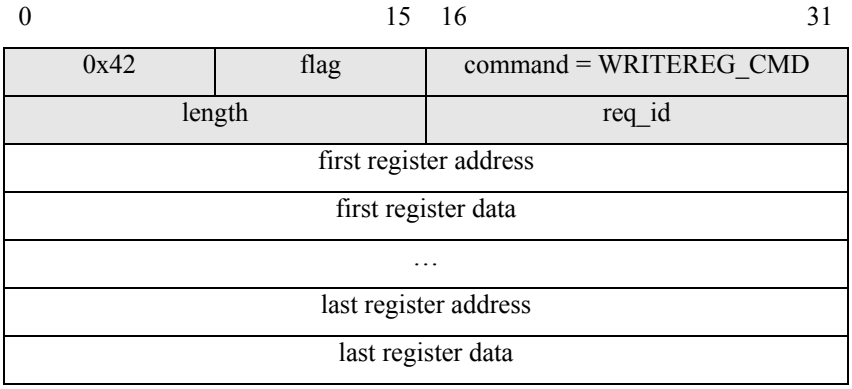


Figure 14-7: WRITEREG_CMD message

WRITEREG_CMD	
register address	Address of the data being written. It must be aligned on a 32-bit boundary (bit 31 and 30 are cleared).
register data	Value of the data to write

14.4.2 WRITEREG_ACK

[R14-34c] WRITEREG_ACK MUST follow the layout of Figure 14-8.

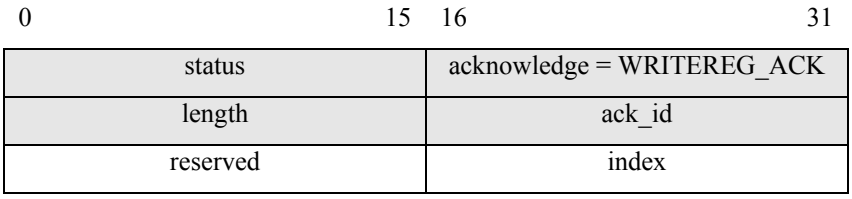


Figure 14-8: WRITEREG_ACK message

WRITEREG_ACK	
reserved	Always 0.
index	On success, this field indicates the number of write operation successfully completed. On failure, this field indicates the index of the register in the list (from 0 to 67) where the error occurred.

When an error occurs during a write operation, the acknowledge message puts the 0-based index of the register where the write error occurred (this index is between 0 and 67); otherwise it is set to the number of write operation successfully completed.

Write operations before the error are correctly completed by the device. All subsequent write operations after the index in this request are discarded.

[R14-35cd] On a WRITEREG failure, an appropriate status code indicating the cause of failure MUST be put in the status field of the acknowledge message by the device.

14.5 READMEM

[R14-36cd] The READMEM command MUST be supported by all devices.

This is required to permit a standard procedure to retrieve the device on-board XML description file.

The application can issue a READMEM message to read consecutive 8-bit locations from the device. This could be useful to read strings, such as the XML description file location, or to read an on-board XML description file.

[R14-37c] The number of 8-bit locations read by this message MUST be a multiple of 4.

[R14-38c] The address specified MUST be aligned on a 32-bit boundary.

The device does not perform any data re-ordering due to little or big-endian conversion since all data is considered as byte. It is up to the application to perform this conversion to correctly interpret the data when it is multi-byte. Endianess of the device can be retrieved from the Device Mode register (at address 0x0004).

The maximum size of memory that can be read by a single READMEM command is 536 bytes (36 bytes are used by the various headers, 4 bytes by the address).

[R14-39cd] If no application has exclusive access, then the device MUST answer READMEM messages coming from the primary or from any secondary application.

[R14-40cd] If an application has exclusive access, then the device MUST only answer READMEM messages coming from the primary application. In this case, READMEM messages coming from secondary applications are acknowledged with a status code of GEV_STATUS_ACCESS_DENIED and the length field is set to 0 (no data payload).

14.5.1 READMEM_CMD

[R14-41c] READMEM_CMD MUST follow the layout of Figure 14-9.

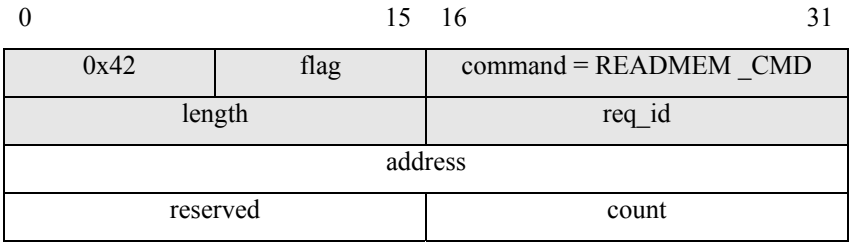


Figure 14-9: READMEM_CMD message

READMEM_CMD	
flag	ACKNOWLEDGE flag should be set
address	Address to read from. Address is automatically incremented for successive data. It must be aligned on a 32-bit boundary (bit 30 and 31 must be cleared)
reserved	Always 0
count	Number of bytes to read from device memory. It must be a multiple of 4 bytes (bit 30 and 31 must be cleared).

14.5.2 READMEM_ACK

[R14-42c] READMEM_ACK MUST follow the layout of Figure 14-10.

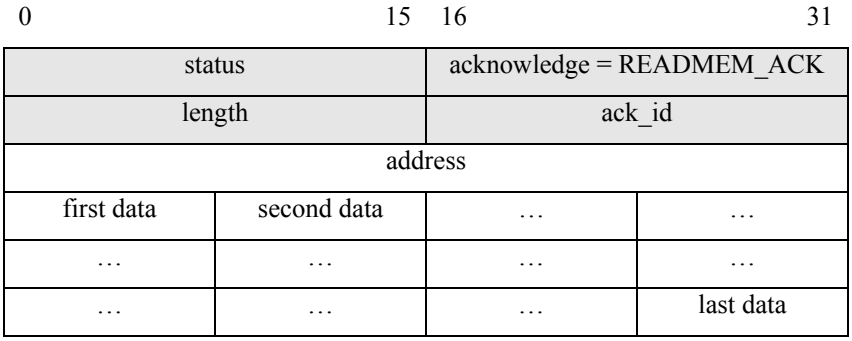


Figure 14-10: READMEM_ACK message

READMEM_ACK	
address	Address where the data was read from. It must be aligned on a 32-bit boundary (bit 30 and 31 must be cleared)
data	8-bit data retrieved from the device registers. Data is copied byte by byte from device memory into this register.

[R14-43cd] READMEM is considered successful only if all requested data has been read. Otherwise the device MUST set an appropriate status code and the "length" field of the header MUST be set to 0 (no data payload).

14.6 WRITEMEM

The WRITEMEM command may optionally be supported by a device.

Note: Even though WRITEMEM is optional, it is **highly recommended** for a device to support it for devices being used through a GenICam interface.

[O14-44ca] An application SHOULD check bit 30 of the "GVCP Capability register" at address 0x0934 to check if WRITEMEM is supported by the device.

The application can issue a WRITEMEM message to write to consecutive 8-bit locations on the device.

[CR14-45c] If WRITEMEM is supported, the number of 8-bit locations written to by this message MUST be a multiple of 4.

[CR14-46c] If WRITEMEM is supported, the address specified MUST be aligned on a 32-bit boundary.

The device does not perform any data re-ordering due to little or big-endian conversion since all data is considered as byte. It is up to the application to perform this conversion to correctly interpret the data when it is multi-byte. Endianess of the device can be retrieved from the Device Mode register (at address 0x0004).

The maximum size of memory that can be written by a single WRITEMEM command is 536 bytes (36 bytes are used by the various headers, 4 bytes by the address).

Only the primary application is allowed to send a WRITEMEM message with the exception of primary application switchover requests.

[CR14-47cd] A device MUST answer WRITEMEM messages coming from the primary application if the command is supported.

[CR14-48cd] If WRITEMEM is supported, a device MUST return a GEV_STATUS_ACCESS_DENIED status code to any secondary application trying to execute this command unless the device is under control access with switchover enabled and another application, with the right credentials, requests control over the device.

14.6.1 WRITEMEM_CMD

[CR14-49c] If supported, WRITEMEM_CMD MUST follow the layout of Figure 14-11.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

0	15	16	31
0x42	flag	command = WRITEMEM_CMD	
length		req_id	
address			
first data	second data	…	…
…	…	…	…
…	…	…	last data

Figure 14-11: WRITEMEM_CMD message

WRITEMEM_CMD	
address	Address of the first data being written. Address is automatically incremented for successive data. It must be aligned on a 32-bit boundary (bit 30 and 31 must be clear)
data	Value of the 8-bit data to write. The number of data to write must be a multiple of 4 bytes.

14.6.2 WRITEMEM_ACK

[CR14-50c] If supported, WRITEMEM_ACK MUST follow the layout of Figure 14-12.

0	15	16	31
status		acknowledge = WRITEMEM_ACK	
length		ack_id	
reserved		index	

Figure 14-12: WRITEMEM_ACK message

WRITEMEM _ACK	
reserved	Always 0
index	On success, this field indicates the number of write operations (in bytes) successfully completed. On failure, this field indicates the index of the data in the list (from 0 to 535) where the error occurred.

[CR14-51cd] If WRITEMEM_CMD is supported, when an error occurs during a memory write operation, the acknowledge message MUST put the 0-based index of the data where the write error occurred (this index is between 0 and 535); otherwise it is set to the number of write operations successfully completed.

Write operations before the error are correctly completed by the device. All subsequent write operations after the index in this request are discarded.

[CR14-52cd] If WRITEMEM_CMD is supported and an error occurs, an appropriate status code indicating the cause of failure MUST be put in the status field of the acknowledge message by the device.

14.7 PACKETRESEND

Packet resend can be used (when supported) to request the retransmission of packets that have been lost. This command is different from other GVCP commands since it can be fired asynchronously by the application. That is the application can issue this command even if it is waiting for the acknowledge message from the previous command. This allows fast retransmission of stream data.

[O14-53cd] The PACKETRESEND command SHOULD be supported by a device.

[O14-54ca] An application SHOULD check bit 29 of the "GVCP Capability register" at address 0x0934 to check if PACKETRESEND is supported by the device before using it.

Any application (primary and secondary) associated to a GVSP receiver can issue a PACKETRESEND message at any time to tell the device associated to a GVSP transmitter to resend a packet of the data stream, typically when this packet was not received by the GVSP receiver. An application can request any number of sequential packets to be retransmitted for a given block_id. An application cannot request an acknowledge for this command: the actual stream packet is used to validate reception and processing of the resend request.

[R14-55ca] An application MUST clear the ACKNOWLEDGE bit of the GVCP header when requesting a PACKETRESEND. That is the application cannot request an acknowledge. The stream packet is used to validate execution of this command.

[CR14-56cd] A device MUST answer PACKETRESEND messages coming from any application, primary and secondary, if the command is supported.

It is up to secondary applications (associated to GVSP receivers) to decide if they want to issue PACKETRESEND or not. This is left as a quality of implementation. A typical scenario is when GVSP transmitters use multicast. The main danger is that many applications might request the same packet to be resent, possibly overloading the device associated to a GVSP transmitter. In this case, the device might receive a large number of resend requests. It is up to the device to decide how to handle duplicated resend requests for the same packet coming from different applications.

[CO14-57cd] A device SHOULD return GEV_STATUS_NOT_IMPLEMENTED in the stream packet to a GVSP receiver if the command is not supported.

An application is not required to synchronize the "req_id" field with the other GVCP commands. It can use any req_id.

[R14-58cd] A device MUST NOT consider PACKETRESEND_CMD to reset the control channel heartbeat.

14.7.1 PACKETRESEND_CMD

[CR14-59c] If supported, PACKETRESEND_CMD MUST follow the layout of Figure 14-13.

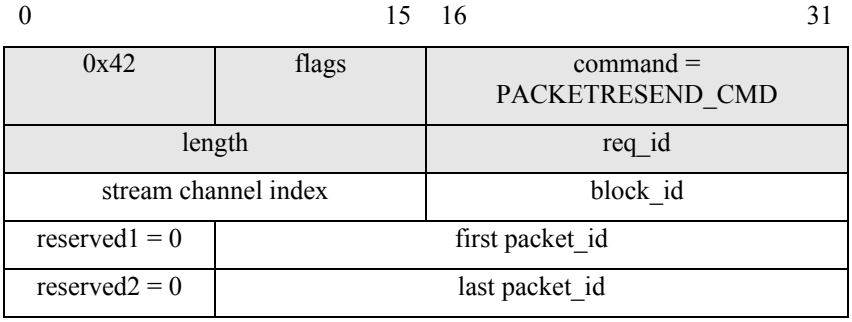


Figure 14-13: PACKETRESEND_CMD message

PACKETRESEND_CMD	
stream channel index	Index of the stream channel (0..511)
block_id	The data block ID. For example, this could represent the “frame” index. 0 is reserved for the test packet and cannot be used.
reserved1	Always 0
first packet_id	The ID of first packet to resend
reserved2	Always 0
last packet_id	The ID of the last packet to resend. It must be equal or greater than first_packet_id. If equal, only one packet is resent. When last packet_id is equal to 0xFFFFFF, this indicates to resend all packets up to (and including) the data trailer. This might be useful for variable size block.

14.7.2 PACKETRESEND_ACK

Since the application associated to a GVSP receiver is not allowed to request a packet resend acknowledge on the control channel, PACKETRESEND_ACK does not exist. The validation of execution of packet resend is performed directly on the stream channel.

[CR14-60cd] If PACKETRESEND_CMD is supported, then the device MUST never issue a PACKETRESEND_ACK on the control channel.

[CO14-63cd] If PACKETRESEND is supported, when the device resends the requested packet or packets, it should use the GEV_STATUS_PACKET_RESEND status code provided that the application has enabled it to use extended status codes.

Version 1.0 of the specification asked for GEV_STATUS_SUCCESS to be used in the above scenario. So both implementations are acceptable.

[CR14-61cd] When PACKETRESEND is supported, if the GVSP transmitter is unable to resend the packet data (it might not be available anymore from device memory or the packet_id might be out of range for the given block_id), then it MUST send a streaming data packet with only the header (no data) and status code set to GEV_STATUS_PACKET_UNAVAILABLE (when the packet data has been discarded), GEV_STATUS_PACKET_NOT_YET_AVAILABLE (when the packet has yet to be acquired), GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY (when the requested packet and all previous ones are not available anymore and have been discarded from the GVSP transmitter memory) or GEV_STATUS_PACKET_REMOVED_FROM_MEMORY (when the requested packet is not available anymore and has been discarded from the GVSP transmitter memory) for each applicable packet_id. GEV_STATUS_PACKET_NOT_YET_AVAILABLE, GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY and GEV_STATUS_PACKET_REMOVED_FROM_MEMORY can only be used after the application has enabled GVSP transmitters to use extended status codes.

Note: A GVSP transmitter can use the GEV_STATUS_PACKET_UNAVAILABLE, GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY or GEV_STATUS_PACKET_REMOVED_FROM_MEMORY to notify a GVSP receiver that the packet requested is not available anymore. Using GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY or GEV_STATUS_PACKET_REMOVED_FROM_MEMORY over GEV_STATUS_PACKET_UNAVAILABLE enables a GVSP transmitter to provide better feedback to a receiver with respect to the reason why a packet is not available anymore. The exact status code to use is left as a quality of implementation and depends of the capabilities of the GVSP transmitter.

Note that “last packet_id” can take a special value (0xFFFFFF) indicating to retransmit all packets from the specified “first packet_id” up to the data trailer (included). The GEV_STATUS_PACKET_UNAVAILABLE, GEV_STATUS_PACKET_NOT_YET_AVAILABLE, GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY or GEV_STATUS_PACKET_REMOVED_FROM_MEMORY error code must be returned for any packet in that range that cannot be resent.

Version 1.0 of the specification asked for the GEV_STATUS_PACKET_UNAVAILABLE status code to be used when the GVSP transmitter is unable to resend packet data. It didn’t make any distinction between the various causes that could have led to a GVSP transmitter not being able to resend a packet. Therefore, a GEV transmitter should use GEV_STATUS_PACKET_UNAVAILABLE in any of the situations where extended resend status codes are neither supported nor enabled.

14.7.3 Packet Resend handling on the GVSP receiver side

Since the amount of memory on the GVSP transmitter side is limited, it is important for the application associated to a GVSP receiver to send any PACKETRESEND command as fast as possible to ensure the

missing data is still available from the GVSP transmitter memory. Otherwise, the GVSP transmitter will return GEV_STATUS_PACKET_UNAVAILABLE, GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY or GEV_STATUS_PACKET_REMOVED_FROM_MEMORY because the data has been lost (possibly overwritten by newer data).

There is a special case for linescan cameras which might be triggered at a variable rate. This specification provides the GEV_STATUS_PACKET_NOT_YET_AVAILABLE error code to announce to GVSP receivers that a particular data packet has yet to be acquired and made available for transmission. This might be the case when the line trigger rate is momentarily slower than the timeout used by the packet resend logic on the GVSP receivers side.

Handling of PACKETRESEND message by the GVSP receiver and its application is left as a quality of implementation. One must remember that UDP packets may arrive out of order at their destination. Therefore, simply looking for consecutive packet_id is not a bulletproof solution to handles all situations.

Some network topologies guaranty that UDP packet will always arrive in order. This is the case if the GVSP transmitter is directly connected to the PC hosting a GVSP receiver using a cross-over cable. Other more complex network topologies, especially those offering multiple routes for the UDP packet to travel from source to destination (using gateways and routers), cannot guaranty UDP packets will arrive in sequential order.

When UDP packets are guarantied to arrive in order, a GVSP receiver and its application can use the packet_id to track down the sequence of packets. If a packet_id is skipped, then GVSP receiver can command its application to request right away the missing packet. A timeout can be used to detect if the data trailer is missing.

When UDP packets are not guarantied to arrive in order, a GVSP receiver and its application cannot assume packet_id values are sequential. Therefore the packet resend mechanism cannot react as fast as for the other case. In this situation, many schemes can be implemented, some using timeouts. It is left as a quality of implementation for a GVSP receiver and its application to offer a mechanism that suits the uncertainty introduced by packet re-ordering.

14.8 PENDING

The PENDING acknowledge is an optional message introduced with version 1.1 of this specification.

When PENDING acknowledge generation is enabled, it is issued to indicate the processing of a command will take a longer period to execute. The application can therefore adjust its acknowledge timeout value. The Pending Timeout bootstrap register (at address 0x0958) indicates the maximum amount of time command execution can take before the PENDING_ACK is issued.

[CR14-64cd] If PENDING message is supported, PENDING_ACK MUST not be issued for DISCOVERY_CMD, FORCEIP_CMD and PACKETRESEND_CMD.

[CR14-65cd] If PENDING message is supported, PENDING_ACK MUST be issued by the device before command execution exceeds the value reported by the Pending Timeout register.

PENDING_ACK will actually extend execution time for a given command. It is important to ensure this does not impact the device enumeration process performed by other applications on the network.

[CR14-66cd] If PENDING message is supported, a device MUST answer DISCOVERY_CMD even if a PENDING_ACK is in progress.

It is recommended for the device to issue the PENDING_ACK as soon as it knows command execution is going to exceed the value indicated by the Pending Timeout register. This enables better command timeout management on the application side.

[CR14-67cd] If PENDING message is supported, PENDING_ACK will temporarily suspend the heartbeat timeout counter between the time PENDING_ACK is issued and the time the command ACK is sent.

The previous conditional requirement is necessary since the application cannot issue the next command before the PENDING_ACK timeout has expired. This PENDING_ACK timeout might be longer than the heartbeat timeout period.

14.8.1 PENDING_ACK

The PENDING_ACK should be issued as soon as the device knows the request execution exceeds the maximum reported value indicated by the Pending Timeout register.

[CR14-68c] If PENDING message is supported, PENDING_ACK MUST follow the layout of Figure 14-14.

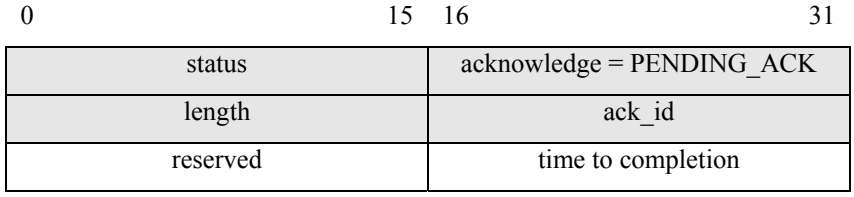


Figure 14-14: PENDING_ACK message

PENDING_ACK	
reserved	Always 0
time to completion	Number of ms to complete the pending request. The application ACK timeout should be set to a value larger or equal to this value.

[CR14-69cd] If PENDING_ACK is supported, the ack_id field MUST be identical to the req_id field of the command that generated the PENDING_ACK.

[CR14-70cd] If PENDING_ACK is supported, the time to completion field MUST indicate the time in ms to complete command execution. If the device is unable to complete execution within this time, it is allowed to issue another PENDING_ACK to extend the command ACK timeout even further. There is no limit to the number of PENDING_ACK re-issue.

The "time to completion" field is 16-bit, leading to a maximal timeout of 65 seconds. Upon reception of the PENDING_ACK, the application resets the acknowledge timeout to the value indicated by this message, plus some margin to allow for network latencies (quality of implementation).

14.9 ACTION

The ACTION command may optionally be supported by a device.

[O14-71ca] An application SHOULD check bit 25 of the "GVCP Capability register" at address 0x0934 to check if ACTION is supported by the device.

ACTION command can be unicasted or broadcasted to a device in order to trigger various actions. This can be used to simultaneously issue the same action to multiple devices residing on the same subnet. The information provided in the request provides mechanism for the device to determine if it is part of the list of devices required to act on this request. If the device must react, the device will decide which action(s) to trigger in accord with the characteristics of the request.

[CR14-72cd] If ACTION message is supported, then the device MUST process ACTION_CMD requests coming from the primary or any secondary application.

14.9.1 ACTION_CMD

[CR14-73c] If supported, ACTION_CMD must follow the layout of Figure 14-15.

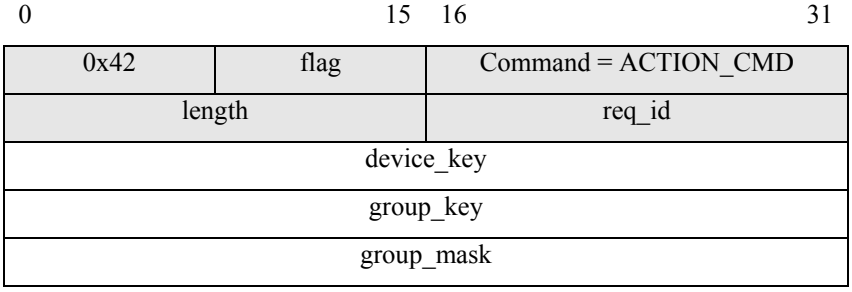


Figure 14-15: ACTION_CMD message

ACTION_CMD	
device_key	Key each sender of an ACTION_CMD has to embed into the packet to prevent erroneous execution of the associated action The device_key has to match the internal Action Device Key register (at address 0x090C) to let a device accept the ACTION_CMD
group_key	Key defining a group of devices on which actions have to be executed. group_mask can be used to filter out some of these devices from the group.
group_mask	The group_mask is compared against the mask associated to the given action number (ACTION_GROUP_MASKx) • 0x00000000 - reserved • 0xFFFFFFFF - override configured mask in all devices. Indicates that any device with a non nil ACTION_GROUP_MASKx for the given action will assert the signal provided that requirements of Table 9-1 are also met. • Any other value – to match against the configured value in the device

This command can be unicasted or broadcasted to the device by the primary or any secondary applications.

14.9.2 ACTION_ACK

The ACTION_ACK is only sent back if the 4 conditions to execute the ACTION are met. Otherwise, the device shall ignore the ACTION_CMD request.

[CR14-74cd] If ACTION_CMD is supported, then a device MUST return an ACTION_ACK message only if an acknowledge is requested and the 4 conditions provided in Table 9-1 are respected by the ACTION_CMD packet.

[CR14-75c] If supported, the ACTION_ACK message must follow the layout of Figure 14-16.

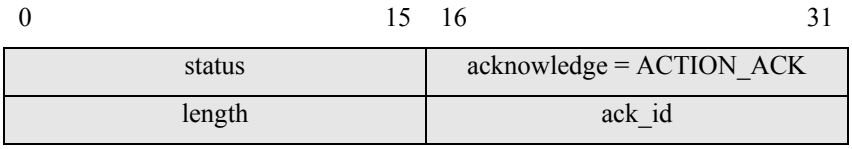


Figure 14-16: ACTION_ACK message

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

15 Message Channel Dictionary

Messages in this category are sent on the Message Channel (if one exists). An application can verify if a message channel is available by reading the "Number of Message Channels register" at address 0x0900. The device always initiates the transaction on the message channel.

[O15-15ca] An application SHOULD check bits 27 and 28 of the "GVCP Capability register" at address 0x0934 to check if EVENT or EVENTDATA commands are supported by the device before opening a message channel.

[CR15-1cd] If Message Channel is available, a device MUST offer a way to enable/disable the generation of event messages.

This is typically done using enable flag. The exact way to do this is left as a quality of implementation. But this specification suggests that each event or group of events could be enabled or disabled individually by writing into enable registers. Each bit of an enable register could represent one event or a group of related events.

[CO15-2cd] The registers used to enable event generation SHOULD be provided in the XML device description file.

15.1 EVENT

[O15-3cd] The EVENT message SHOULD be supported by a device.

~~[O15-4ca] An application SHOULD check bit 28 of the "GVCP Capability register" at address 0x0934 to check if EVENT command is supported by the device before using it.~~

The device may use EVENT messages to notify the application that asynchronous events have occurred. Multiple events can be concatenated in one message, as long as the total packet size is lower or equal to 576 bytes. Therefore, the maximum number of events in one packet is 33. It is up to the device to decide if multiple events are concatenated or sent as separate packets (this is independent of the concatenation flag, bit 31 of "GVCP Capability" register at address 0x0934). The maximum number of events in the message is equal to the header "length field / 16".

[CR15-5cd] If EVENT message is supported, each EVENT command MUST be tagged with a 64-bit timestamp representing the time at which the event was generated on the device. The frequency of this timestamp is available from the "Timestamp Tick Frequency" registers at address 0x093C and 0x0940. No timestamp is available if this register is set to 0. In this case, the timestamp field of the message MUST be set to 0.

[CO15-6cd] This protocol reserves event identifier from 0 to 36863 for GigE Vision usage. Other event identifiers are available for device-specific events and SHOULD be described in the XML device description file when EVENT message is supported.

Note that events from 32769 to 36863 directly maps to standard GigE Vision status code. This can be used by a device to asynchronously report an error.

15.1.1 EVENT_CMD

[CR15-7c] If EVENT message is supported, EVENT_CMD MUST follow the layout of Figure 15-1.

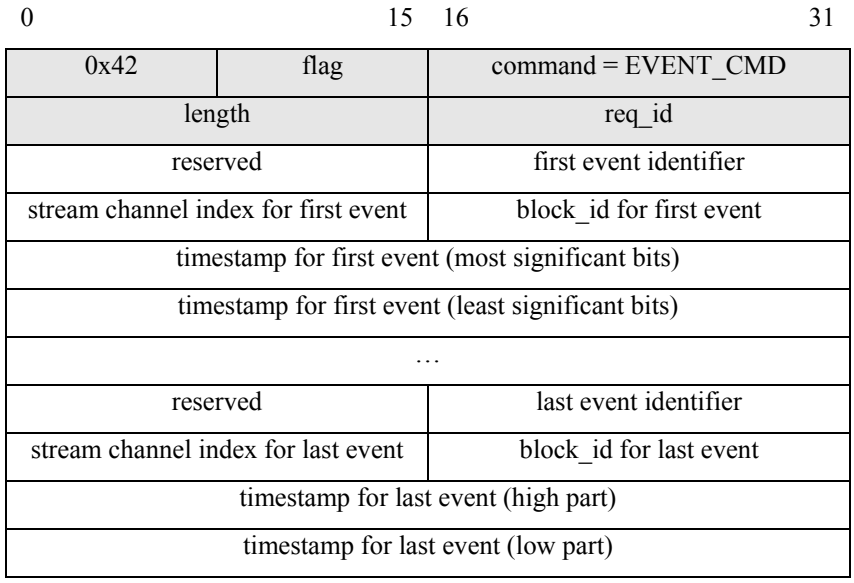


Figure 15-1: EVENT_CMD message

EVENT_CMD	
reserved	Always 0
event identifier	event number as defined by the GigE Vision specification for value between 0 and 36863 or the XML device description file for value between 36864 and 65535
stream channel index	Index of stream channel associated with this event. 0xFFFF if no stream channel involved.
block_id	ID of the data block associated to this event. 0 if no block_id associated to this event.
timestamp (high part)	32 most significant bits of the 64-bit timestamp generated by the device to indicate when this event happened. 0 if timestamp is not supported.
timestamp (low part)	32 least significant bits of the 64-bit timestamp generated by the device to indicate when this event happened. 0 if timestamp is not supported.

15.1.2 EVENT_ACK

A device is not required to request an acknowledge message.

[CR15-8c] If EVENT message is supported, EVENT_ACK MUST follow the layout of Figure 15-2.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

0	15	16	31
status		acknowledge = EVENT_ACK	
length		ack_id	

Figure 15-2: EVENT_ACK message

15.2 EVENTDATA

The EVENTDATA message may be supported by a device.

~~[O15-9ca] An application SHOULD check bit 27 of the "GVCP Capability register" at address 0x0934 to check if the EVENTDATA command is supported by the device before using it.~~

The device may use EVENTDATA messages to notify the application that asynchronous events have occurred. The main difference from the EVENT message is that device-specific data can be attached to this message. The total packet size must be lower or equal to 576 bytes. Therefore, device-specific data is limited to 540 bytes.

[CR15-10cd] If EVENTDATA message is supported, each EVENTDATA command MUST be tagged with a 64-bit timestamp representing the time at which the event was generated on the device. The frequency of this timestamp is available from the "Timestamp Tick Frequency" registers at address 0x093C and 0x0940. No timestamp is available if this register is set to 0. In this case, the timestamp field of the message is set to 0.

[CR15-11cd] It is not possible to concatenate multiple events into a EVENTDATA command when this command is supported. A device MUST only put a single event in the EVENTDATA message when this message is supported.

The event identifiers are the same as the ones associated to the EVENT message. This protocol reserves event identifier from 0 to 36863 for GigE Vision usage. Other event identifiers are available for device-specific events and should be described in the XML device description file.

[CO15-12cd] The content of the device-specific data associated to EVENTDATA SHOULD be specified in the XML device description file when this command is supported.

15.2.1 EVENTDATA_CMD

[CR15-13c] If EVENTDATA message is supported, EVENTDATA_CMD MUST follow the layout of Figure 15-3.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

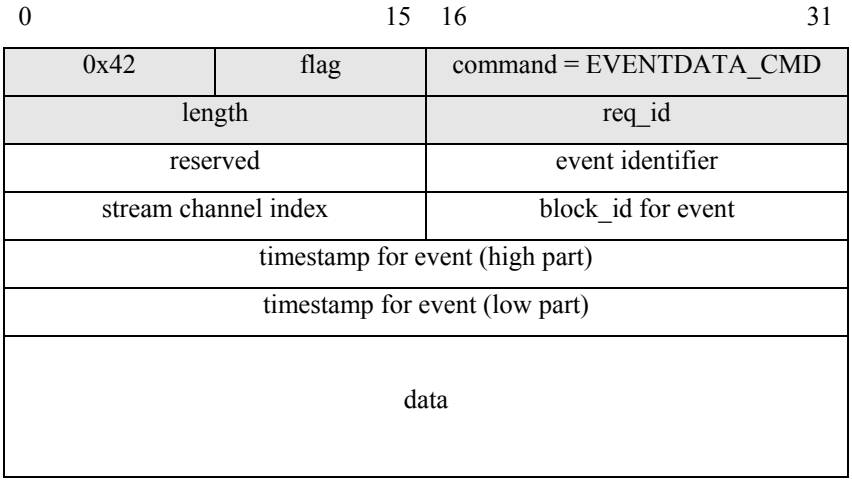


Figure 15-3: EVENTDATA_CMD message

EVENTDATA_CMD	
reserved	Always 0
event identifier	event number as defined by the GigE Vision specification for value between 0 and 36863 or the XML device description file for value between 36864 and 65535
stream channel index	Index of stream channel associated with this event. 0xFFFF if no stream channel involved.
block_id	ID of the data block associated to this event. 0 if no block_id associated to this event.
timestamp (high part)	32 most significant bits of the 64-bit timestamp generated by the device to indicate when this event happened. 0 if timestamp is not supported.
timestamp (low part)	32 least significant bits of the 64-bit timestamp generated by the device to indicate when this event happened. 0 if timestamp is not supported.
data	Raw data. Specified by the XML device description file.

15.2.2 EVENTDATA_ACK

A device is not required to request an acknowledge message.

[CR15-14c] If EVENTDATA message is supported, EVENTDATA_ACK MUST follow the layout of Figure 15-4.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

0	15	16	31
status		acknowledge = EVENTDATA_ACK	
length		ack_id	

Figure 15-4: EVENTDATA_ACK message

16 Command and Acknowledge Values

The following table lists the numerical value associated with each message defined in this specification. The support column indicates if the message is (**M**)andatory, (**R**)ecommended or (**O**)ptional. The channel type indicates on which channel the message is transmitted.

This specification reserves command and acknowledge values from 0 to 32767 for GVCP (most-significant bit cleared). Other values, from 32768 to 65535 (most-significant bit set), are available for device-specific messages.

Note that the acknowledge message associated to a command message always have the value of the corresponding command message plus one. Therefore, command messages have their least-significant bit cleared while acknowledge messages have their least-significant bit set.

[R16-1c] Device and application MUST use the following value to represent the various messages. Notice the value of the acknowledge message is always the associated command message + 1 (when such a command message exists).

Message	Support	Channel	Value
Discovery Protocol Control			
DISCOVERY_CMD	M	Control	0x0002
DISCOVERY_ACK	M	Control	0x0003
FORCEIP_CMD	M	Control	0x0004
FORCEIP_ACK	M	Control	0x0005
Streaming Protocol Control			
PACKETRESEND_CMD	R	Control	0x0040
Device Memory Access			
READREG_CMD	M	Control	0x0080
READREG_ACK	M	Control	0x0081
WRITEREG_CMD	M	Control	0x0082
WRITEREG_ACK	M	Control	0x0083
READMEM_CMD	M	Control	0x0084
READMEM_ACK	M	Control	0x0085
WRITEMEM_CMD	O	Control	0x0086
WRITEMEM_ACK	O	Control	0x0087
PENDING_ACK	O	Control	0x0089
Asynchronous Events			
EVENT_CMD	R	Message	0x00C0
EVENT_ACK	R	Message	0x00C1
EVENTDATA_CMD	O	Message	0x00C2

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

EVENTDATA_ACK	O	Message	0x00C3
Miscellaneous			
ACTION_CMD	O	Control	0x0100
ACTION_ACK	O	Control	0x0101

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

17 Retrieving the XML Device Configuration File

The GigE Vision specification uses the GenICam specification to define the device capabilities. The bootstrap registers provide sufficient information for an application to establish communication and to retrieve the XML device configuration file.

[R17-1cd] A GigE Vision device MUST have a XML device description file with the syntax described in the GenApi module of the GenICam standard.

[CR17-2cd] For a camera device, the names and types of the device's mandatory features MUST respect the standard feature list given in Section 28.

Because of the large size of the XML file (which can be a few hundred kilobytes), this specification supports file compression. The file extension indicates the type of compression used by the device. It is up to the application to decompress the file after the download. Note the device only stores a copy of the XML file. As such, it does not need to have specific knowledge about file compression (the XML file might have been compressed prior to be stored in device non-volatile memory).

[R17-3ca] Application MUST support uncompressed and compressed (ZIP) XML file.

1. Uncompressed (xml): In this case, the file MUST be stored with the standard .XML extension. The file is an uncompressed text file.
2. Zipped (zip): The file is compressed using the standard .ZIP file format. The file extension MUST be .ZIP. Only the DEFLATE and STORE compression methods of the .ZIP file format specification are permitted by this specification. It is up to the application to uncompressed the file after the download.

Compression might be useful to minimize the amount of non-volatile memory used to store the file and to minimize the file download time.

Note: The .ZIP algorithm was developed by Phil Katz for PKZIP in January 1989. DEFLATE is a lossless data compression algorithm that uses a combination of the LZ77 algorithm and Huffman coding. It was originally defined by Phil Katz for version 2 of his PKZIP archiving tool.

Note: The specification of the .ZIP is available from PKWARE at http://www.pkware.com/documents/casestudies/APPNOTE.TXT. Free versions of the ZIP compressor and decompressor utilities are available at http://www.info-zip.org.

The XML device configuration file can be stored in 3 different locations:

1. Device non-volatile memory
2. Vendor web site
3. A local directory on the application machine

Two bootstrap registers are used to indicate the location of the file:

1. First choice of URL (address 0x0200)
2. Second choice of URL (address 0x0400)

These 2 registers store a case insensitive NULL-terminated string of up to 512 bytes.

[R17-4ca] When looking for the XML file, the application MUST try to use the first URL choice, and then move to the second choice of URL if unsuccessful in using the first choice.

These URL can be read each using a single call to READMEM.

[R17-5cd] A device MUST align the start of the XML file to a 32-bit boundary to ease the retrieve process using READMEM.

17.1 Device Non-Volatile Memory

When the URL is in the form "Local:*Filename.Extension;Address;Length*", this indicates the XML device configuration file is available from on-board non-volatile memory (entries in italic must be replaced with actual character values).

Field	Description
Local	Indicates the XML device configuration file is available from on-board non-volatile memory.
Filename	Name of the file. It is recommended to put the vendor, device and revision information in the filename separated by underscore. For instance: acme_titan_rev1 is suited to revision 1 of the Titan device produced by the Acme company.
Extension	Indicates the type of file. xml: uncompressed text file zip: compressed zip file
Address	Starting address of the file in device memory map. It must be expressed in hexadecimal form.
Length	Length in byte of the file. It must be expressed in hexadecimal form.

For example "Local:acme_titan_rev1.zip;1C400;A000" indicates "acme_titan_rev1" is a .ZIP file located at address 0x1C400 in device memory map with a length of 0xA000 bytes. Having the filename might be handy to compare with another version of the file that might be available locally on the application machine.

[O17-6ca] When the XML device description file is stored locally on the device, the application SHOULD use the READMEM message of GVCP to retrieve it.

Because READMEM message is a multiple of 32-bit, the read request might ask for slightly more data than really required to allow for alignment and length restriction.

The way READMEM operates is very similar to TFTP (Trivial file transfer protocol, RFC1350): the file is divided into small blocks of one GVCP packet, and an ACK is required for each block before the application sends a request for the next block.

READMEM is a mandatory command of GVCP. This way, an application can always use this command to get a local XML device description file.

17.2 Vendor Web Site

When the URL is in the form "http://www.*manufacturer*.com/*filename.extension*, this indicates the XML device configuration file is available from the vendor web site on the Internet. This is a standard URL that can be used directly in a web browser. All fields are not case sensitive.

Field	Description
http	Indicates the file is available on the Internet.
www.*manufacturer*.com	This is the vendor web site.
Filename	Name of the file. It is recommended to put the vendor, device and revision information in the filename separated by underscore. For instance: acme_titan_rev1 is suited to revision 1 of the Titan device produced by the Acme company.
Extension	Indicates the type of file. xml: uncompressed text file zip: compressed zip file

For example "http://www.acme.com/camera/acme_titan_rev1.xml" indicates the XML device description file is available on Acme web site. The XML file extension indicates the file is an uncompressed text file. Note that URL with extensions different than ".com" are also acceptable.

17.3 Local Directory

When the URL is in the form "File:*filename.extension*", this indicates the XML device description file is available on the machine running the application. The file is typically shipped on a CD coming with the device. In this case, the user must put this file in a pre-defined directory where the application stores device description files. It is up to the application software to correctly locate the file that matches the filename.

Field	Description
File	Indicates the XML device configuration file must be retrieved from a directory on the application machine.
Filename	Name of the file. It is recommended to put the vendor, device and revision information in the filename separated by underscore. For instance: acme_titan_rev1 is suited to revision 1 of the Titan device produced by the Acme company.
Extension	Indicates the type of file. xml: uncompressed text file zip: compressed zip file

For example, “File:acme_titan_rev1.zip” indicates the XML device description file must be available in the device description folder of the application under the filename “acme_titan_rev1.zip”. Because the file extension is .ZIP, the file is in ZIP compressed format.

17.4 Manifest Table

A GigE Vision device may support multiple versions of the GenApi schema and thus supply multiple XML files.

A manifest table register block shows a list of entries which provide information about the referred XML files. Each entry in the Manifest Table contains the version number of the XML file itself as well as the version number of the GenApi schema the XML file is based on. In addition the entry contains the address of a pair of URL registers which names and locates the document.

The Manifest Table only supports GenICam XML files. However, a GenICam file may provide a mechanism to reference other documents (such as datasheets).

When the Manifest Table feature is not supported by a device, then the application must revert back to the original URL locations described above in this section. A capability bit is provided in the GVCP capability register (at address 0x0934) to indicate if the Manifest Table feature is supported.

18 Status Code

This section lists the various status codes that can be returned in an acknowledge message or a GVSP header. This specification defines two categories of status code:

1. Standard status code
2. Device-specific status code

Standard status codes are defined in this specification.

[O18-1cd] A device SHOULD return the status code that provides as much information as possible about the source of error.

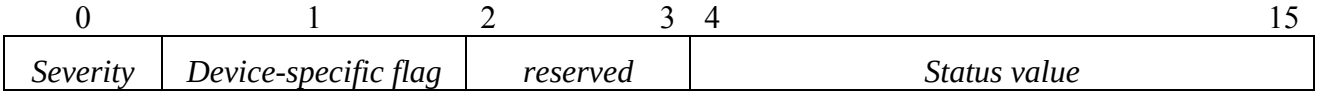
[R18-2cd] If a device cannot return a more descriptive status code, the device MUST at least return the GEV_STATUS_ERROR status code.

[O18-3st] A GVSP transmitter SHOULD return the status code that provides as much information as possible about the status of streaming activities. For instance, a memory or data overrun error in the GVSP transmitter should lead to a data trailer packet sent with a status field set to GEV_STATUS_DATA_OVERRUN.

[R18-4st] If a GVSP transmitter cannot return a more descriptive status code upon the occurrence of a streaming error, it MUST at least return the GEV_STATUS_ERROR status code.

A device or GVSP transmitter is not required to support all status codes. Some status codes are suited to the application.

Status register is mapped as follows:



Severity	0 info, 1 error
Device-specific flag	0 for standard GigE Vision status codes, 1 for device-specific status codes (from XML device description file)
Status value	actual value of status code
Reserved	Always 0

Table 18-1: List of Standard Status Codes

Status Value	Description	Value (hexadecimal)
GEV_STATUS_SUCCESS	Command executed successfully	0x0000
GEV_STATUS_PACKET_RESEND[1]	Only applies to packet being resent. This flag is preferred over the GEV_STATUS_SUCCESS when the GVSP transmitter sends a resent packet. This can be used by a GVSP receiver to better monitor packet resend.	0x0100
GEV_STATUS_NOT_IMPLEMENTED	Command is not supported by the device	0x8001
GEV_STATUS_INVALID_PARAMETER	At least one parameter provided in the command is invalid (or out of range) for the device	0x8002
GEV_STATUS_INVALID_ADDRESS	An attempt was made to access a non existent address space location.	0x8003
GEV_STATUS_WRITE_PROTECT	The addressed register cannot be written to	0x8004
GEV_STATUS_BAD_ALIGNMENT	A badly aligned address offset or data size was specified.	0x8005
GEV_STATUS_ACCESS_DENIED	An attempt was made to access an address location which is currently/momentary not accessible. This depends on the current state of the device, in particular the current privilege of the application.	0x8006
GEV_STATUS_BUSY	A required resource to service the request is not currently available. The request may be retried at a later time.	0x8007
GEV_STATUS_LOCAL_PROBLEM	An internal problem in the device implementation occurred while processing the request. Optionally the device provides a mechanism for looking up a detailed description of the problem. (Log files, Event log, 'Get last error' mechanics). This error is intended to report problems from underlying services (operating system, 3rd party library) in the device to the client side without translating every possible error code into a GigE Vision equivalent.	0x8008
GEV_STATUS_MSG_MISMATCH	Message mismatch (request and acknowledge do not match)	0x8009
GEV_STATUS_INVALID_PROTOCOL	This version of the GVCP protocol is not supported	0x800A
GEV_STATUS_NO_MSG	Timeout, no message received	0x800B
GEV_STATUS_PACKET_UNAVAILABLE	The requested packet is not available anymore.	0x800C
GEV_STATUS_DATA_OVERRUN	Internal memory of GVSP transmitter overrun (typically for image acquisition)	0x800D
GEV_STATUS_INVALID_HEADER	The message header is not valid. Some of its fields do not match the specification.	0x800E

[1] This extended status code must be used only when extended status codes are enabled (GVCP Configuration register, bit 29).

Status Value	Description	Value (hexadecimal)
GEV_STATUS_WRONG_CONFIG	The device current configuration does not allow the request to be executed due to parameters consistency issues.	0x800F
GEV_STATUS_PACKET_NOT_YET_AVAILABLE[1]	The requested packet has not yet been acquired. Can be used for linescan cameras device when line trigger rate is slower than application timeout.	0x8010
GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY[1]	The requested packet and all previous ones are not available anymore and have been discarded from the GVSP transmitter memory. An application associated to a GVSP receiver should not request retransmission of these packets again.	0x8011
GEV_STATUS_PACKET_REMOVED_FROM_MEMORY[1]	The requested packet is not available anymore and has been discarded from the GVSP transmitter memory. However, applications associated to GVSP receivers can still continue using their internal resend algorithm on earlier packets that are still outstanding. This does not necessarily indicate than any previous data is actually available, just that the application should not just assume everything earlier is no longer available.	0x8012
GEV_STATUS_ERROR	Generic error. Try to avoid and use a more descriptive status code from list above.	0x8FFF

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

19 Events

The following table lists the standard events defined by this specification.

[CR19-1cd] When a message channel is supported, event identifier with data MUST be sent using EVENTDATA command (if this command is supported).

Note: Events related to image acquisition have been deprecated in version 1.1 of this specification. The proper definition of events is managed by the GenICam Standard Features Naming Convention for GigE Vision.

Table 19-1: List of Events

Event Identifier	Description	Data	Value (hexadecimal)
GEV_EVENT_TRIGGER (deprecated)	The device has been triggered.	None	0x0002
GEV_EVENT_START_OF_EXPOSURE (deprecated)	Start of sensor exposure	None	0x0003
GEV_EVENT_END_OF_EXPOSURE (deprecated)	End of sensor exposure	None	0x0004
GEV_EVENT_START_OF_TRANSFER (deprecated)	Start of transfer on the stream channels	None	0x0005
GEV_EVENT_END_OF_TRANSFER (deprecated)	End of transfer on the stream channels	None	0x0006
GEV_EVENT_PRIMARY_APP_SWITCH	Primary application switchover has been granted.	None	0x0007
Reserved	-	-	0x0008 to 0x8000
GEV_EVENT_ERROR_xxx	An asynchronous error occurred on the device. The value of the event identifier represents the status code for the asynchronous error. Refer to the list of status code	None	0x8001 to 0x8FFF
Device-specific Event	Refer to the XML device description file.	Check device documentation	0x9000 to 0xFFFF

20 ICMP

[R20-1cd] A device MUST minimally support the subset of ICMP provided in the Table 20-1 on all its supported network interfaces.

Each ICMP message is defined to be either mandatory or optional. Only "Echo Reply", "Echo" and "Destination Unreachable" are mandatory.

For instance, a device must support the 'ping' command for packet size up to 576 bytes. Therefore, the device must correctly receive an ICMP Echo message and must correctly transmit an ICMP Echo Reply message.

Note: The Destination Unreachable ICMP message must be managed according to the recommendation of the RFC. This message might result from a routing transient, so it is a hint, not a proof, that the specified destination is unreachable. RFC1122 provides additional details about Destination Unreachable. This RFC provides recommendation on the action to take upon the reception of this message based on the error code embedded into it. For instance, a connection must not be aborted if a soft error is received while it should in the case of the reception of hard errors. If a device decides to close its channels due to a communication error, it must follow [R9-12cd].

Table 20-1: ICMP Messages

Message code	Message description	Device reception	Device transmission
0	Echo Reply	Optional	Mandatory (for packets size up to 576 bytes), not required for packet size larger than 576.
3	Destination Unreachable	Mandatory	Optional
4	Source Quench	Optional	Optional
5	Redirect	Optional	Optional
8	Echo	Mandatory (for packets size up to 576 bytes), not required for packet size larger than 576.	Optional
9	Router Advertisement	Optional	Optional
10	Router Solicitation	Optional	Optional
11	Time Exceeded	Optional	Optional
12	Parameter Problem	Optional	Optional
13	Timestamp	Optional	Optional
14	Timestamp Reply	Optional	Optional

15	Information Request	Optional	Optional
16	Information Reply	Optional	Optional
17	Address Mask Request	Optional	Optional
18	Address Mask Reply	Optional	Optional

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

PART 3 – GVSP

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

21 GVSP Summary

21.1 Overview

GVSP is an application layer protocol relying on the UDP transport layer protocol. It allows a GVSP receiver to receive image data, image information and other information from a GVSP transmitter.

GVSP packets always travel from a GVSP transmitter to a receiver.

The current version on the specification uses UDP IPv4 as the transport layer protocol. Because UDP is unreliable, GVSP provides mechanisms to guaranty the reliability of packet transmission (through GVCP) and to ensure minimal flow control.

21.2 Goals

The goals for GigE Vision Streaming Protocol (GVSP) are:

- Define a mechanism for a GVSP transmitter to send image data, image status and other data to a GVSP receiver.
- Support self-describing data.
- Minimize IP stack complexity in the GigE Vision entities.
- Minimize the network bandwidth overhead.

21.3 Scope

This section provides:

- Specification of the stream protocol.
- Headers for the various packet types.
- List of pixel formats.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

22 GVSP Transport Protocol Considerations

[R22-1s] The stream protocol MUST use UDP with IPv4 transport protocol.

[R22-2s] GVSP transmitters and receivers MUST NOT use any IP options in the IP datagram for GVSP. This way, the IP header size is fixed at 20 bytes. This is to allow efficient hardware implementation of UDP/IP offload engines.

22.1 UDP

UDP is a connectionless protocol that adds no reliability, flow-control or error recovery to IP. Because of this lack of functionality, GVSP imposes restrictions on how stream channels are handled.

[R22-3sr] GVSP receivers MUST be able to accommodate packets arriving out of order because UDP cannot guaranty the order of packet delivery.

22.1.1 Fragmentation

Since a stream channel always transmits from a GVSP transmitter to a GVSP receiver, it is up to the receiver to decide if it can support IP fragmentation. The SCPSx of GVSP transmitters provides a “do not fragment” bit that is copied into the IP header of each packet of the corresponding stream channel. The SCCx capability register can be used by GVSP receivers to advertise if IP reassembly is supported. It is up to the management entity of the system to make sure that GVSP transmitters are configured according to the capabilities of the GVSP receivers. In case of doubts, IP fragmentation should not be used.

22.1.2 Packet Size Requirements

Contrary to the GVCP, the GVSP packet size is not required to be a multiple of 32 bits: it has a byte resolution. The packet size to use can be determined by firing test packets to find the MTU of the transmission medium. Refer to the SCPSx bootstrap register.

Packet size requested through SCPSx refers to data payload packets, not to data leader and data trailer. The data leader and data trailer must be contained within a 576 bytes packet.

22.1.3 Reliability and Error Recovery

UDP is inherently an unreliable transport protocol. GVSP provides reliability by monitoring the packet_id field and validating that all packets composing a block have been received. If packets are missing, then the primary application associated to GVSP receivers can request them using the PACKETRESEND command (when this command is supported by the GVSP transmitter). packet_id is incremented by one for successive packets.

UDP checksum support is not mandatory.

The following figure shows a typical packet exchange when a packet is lost. Note the time it takes for the GVSP receiver and its application to react to a missing packet is implementation dependent. Remember that UDP does not guaranty packets are delivered in the same order they were transmitted by the GVSP transmitter.

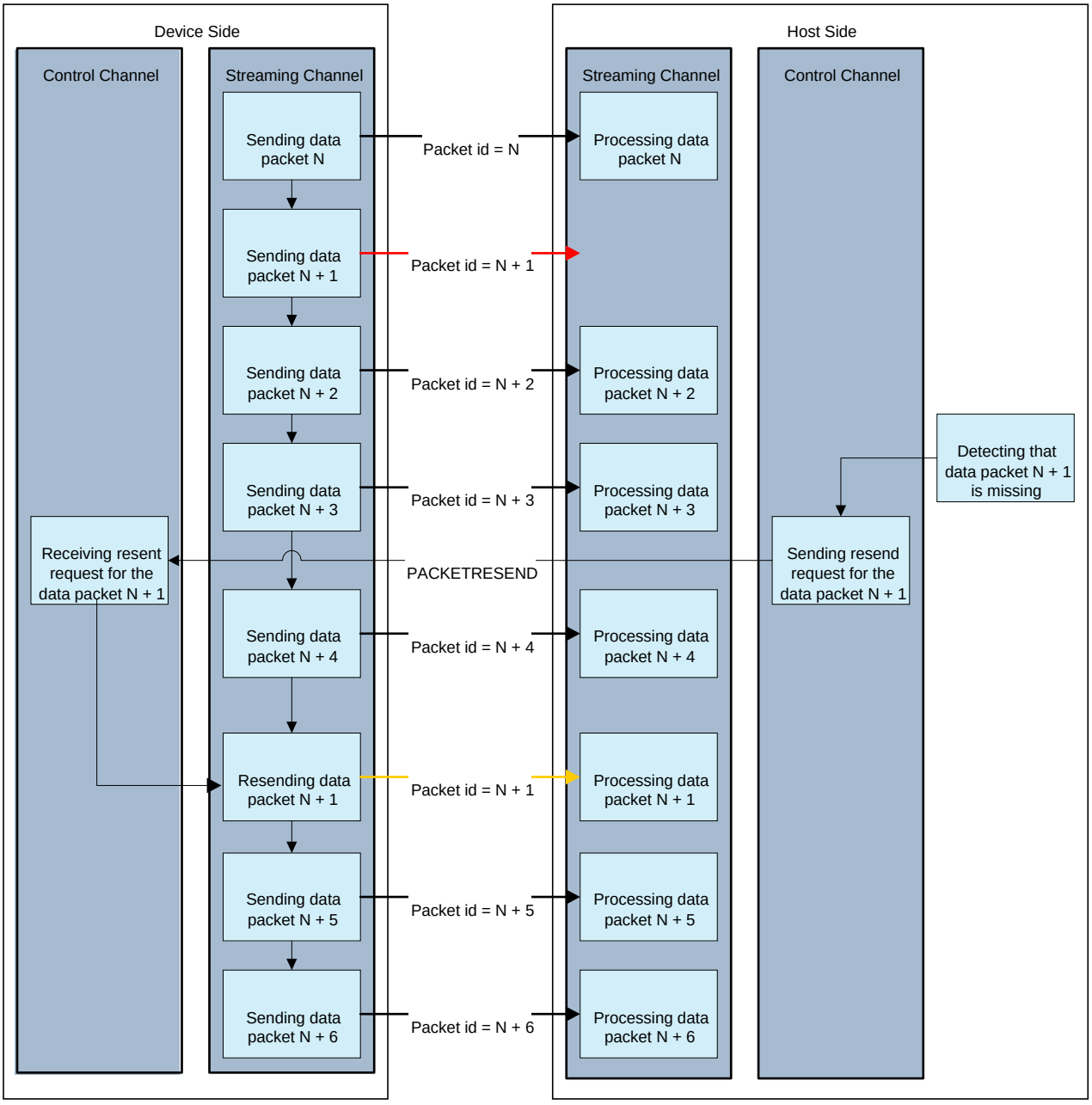


Figure 22-1: Data Resend Flowchart

22.1.4 Flow Control

There is no acknowledge associated to a stream packet. Therefore, GVSP offers a crude flow-control method based on an inter-packet delay register (SCPDx). This register indicates the minimal amount of time (in timestamp counter unit) to wait between the transmission of two streaming packets on the given stream channel for a GVSP transmitter. This amount of time is measured from the end of the first packet to the start of the next packet.

The timestamp counter is typically the timer with the finest granularity on the device. If there is no timestamp counter, then SCPDx cannot be used to manage the flow control.

22.1.5 End-to-End Connection

Connections are managed using GVCP. GVCP heartbeat monitors the presence of device and application.

GVSP provides no provision for end-to-end connection. If the application is disconnected, the GVCP heartbeat timer will expire and automatically will reset the streaming connection of GVSP transmitters unless they have been configured to continue to stream as per Section 10.8. By default, if the device is disconnected, the GVSP receivers will stop to receive streaming packets from the GVSP transmitters associated to that device.

22.1.6 Device error handling during acquisition

[R22-4st] If an acquisition error occurs in the GVSP transmitter, the transmitter might not be able to send all data packets for that data block. It MUST however, send the corresponding data trailer with an appropriate status code.

The GVSP transmitter must send one data trailer for each discarded block of data, even if a single packet is lost in the acquisition section. This ensures a GVSP receiver and its application will not ask for packet resend pointlessly.

For example a memory or data overrun error in the GVSP transmitter should lead to a data trailer packet sent with a status field set to GEV_STATUS_DATA_OVERRUN. This is to ensure that a GVSP receiver and its application are notified of corrupted image data and to prevent the delivery of corrupted images to the user.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

23 Data Block

Information passed on a stream channel is divided into blocks. For instance, a GigE Vision camera fits each captured image into a data block. A GVSP receiver would thus be able to track images by looking at the block_id associated with each data block.

[R23-1s] The data block MUST contain three elements:

1. **Data Leader**: one packet used to signal the beginning of a new data block
2. **Data Payload**: one or more packets containing the actual information to be streamed for the current data block.
3. **Data Trailer**: one packet used to signal the end of the data block

A typical data block is illustrated below.

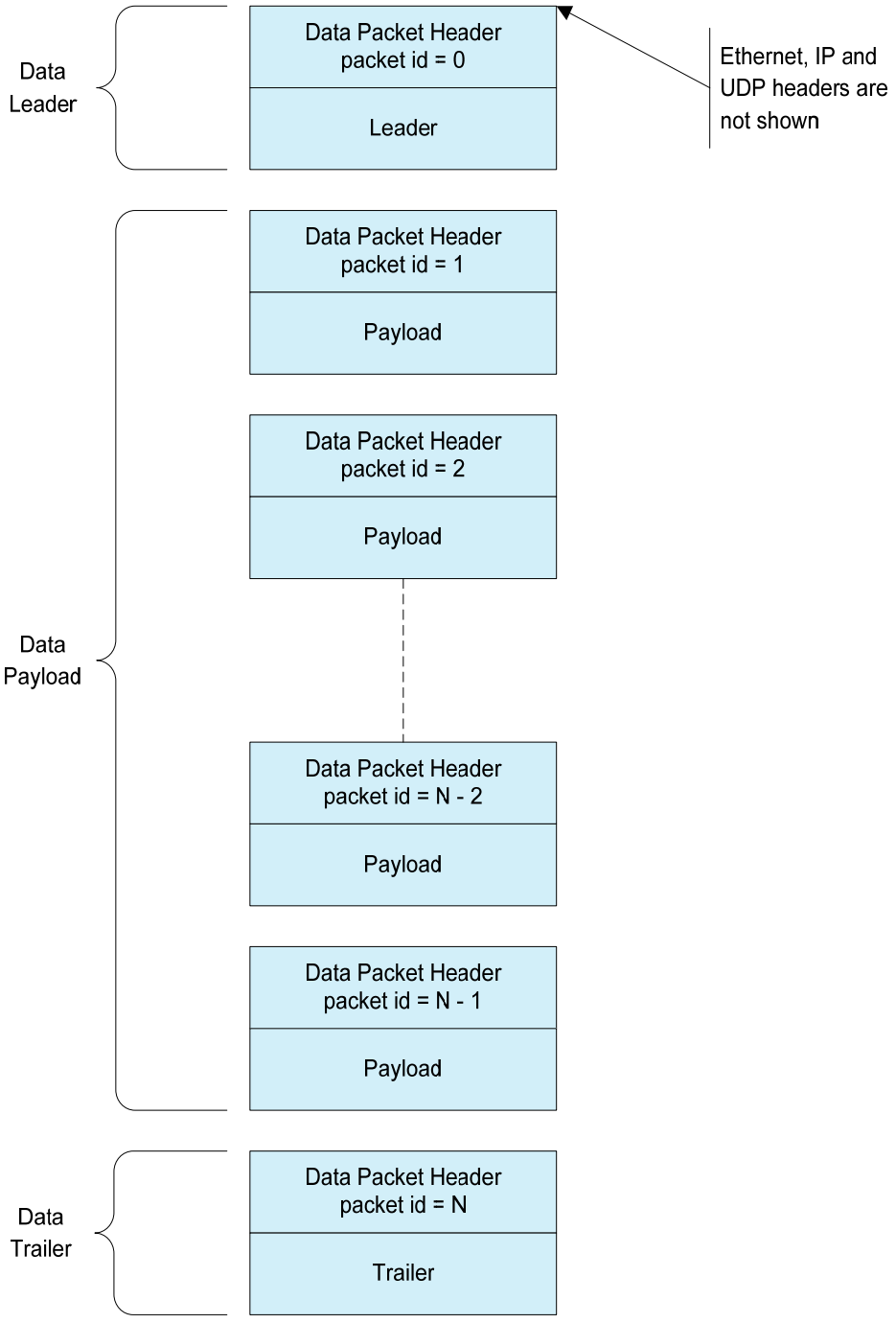


Figure 23-1: Data Block

Data blocks are sent sequentially on a given stream channel.

[R23-2st] A GVSP transmitter MUST send data blocks sequentially. It is not allowed to send the data leader of the next block before the data trailer of the current block has been transferred. Only exception is for PACKETRESEND which might request any packet in the current or in a previous data block.

To efficiently transport information, GVSP defines various payload types that can be streamed out of a GVSP transmitter:

1. Image
2. Raw data
3. File
4. Chunk data
5. Extended chunk data
6. Device-specific

23.1 Image Payload Type

[CR23-3st] If Image payload supported, a stream using the image payload type MUST output data in raster-scan format.

This means the image is reconstructed in the GVSP transmitter memory before being transmitted from left to right, then top to bottom. This is typical with single-tap sensor.

The position of the first pixel in a given data packet can be obtained by multiplying packet size (SCPSx register) with “packet_id – 1”. That means there is no gap in the image data.

[CR23-4st] If Image payload is supported, the GVSP transmitter MUST send data packet sequentially (from 1 to N-1 as per Figure 23-1).

If packets are lost, a GVSP receiver and its application can use the PACKETRESEND command (when supported by the GVSP transmitter) to ask for a group of consecutive packets.

[CR23-5st] If Image payload is supported, for multi-tap data to be sent on a single stream channel, the GVSP transmitter MUST reconstruct the image locally so it can be sent using sequential packet_id.

[CO23-6st] If Image payload is supported, when a multi-tap transmitter cannot reconstruct the image before transmission, then it SHOULD use a separate stream channel for each tap to transmit.

It is then up to the GVSP receiver to reconstruct the image (at the expense of increased processing time and resources). The tap configuration information and the way they are associated to stream channels should be available in the XML device description file or in the product documentation.

This specification recommends that a product transmitting images reconstructs the image in raster-scan format before transmission.

23.2 Raw Data Payload Type

This payload type is used to stream raw data from a GVSP transmitter to GVSP receivers. For instance, this can be used to send acquisition statistics.

[CR23-7st] If Raw Data payload is supported, the amount of information to transfer MUST be provided in the data leader. It can be variable from one data block to the next.

23.3 File Payload Type

This payload type is used to transfer files that can be directly saved to a GVSP receiver hard disk. For instance, a GVSP transmitter could use JPEG compression to deliver compressed files to a GVSP receiver.

[CR23-8st] If File payload is supported, the amount of information to transfer MUST provided in the data leader. It can be variable from one data block to the next.

23.4 Chunk Data Payload Type

This payload type is used to stream chunks. Chunks are tagged blocks of data. Examples for chunks are:

- image
- data extracted from image
- AOI / pixel format
- state of io-pins
- exposure number

Each chunk consists of the chunk data and a trailing tag. The tag contains:

- A unique chunk identifier, which identifies the structure of the chunk data and the chunk feature associated with this chunk
- The length of the chunk data

As the chunk tags (CID and Length fields) are headers embedded in the payload of chunk format block, their byte order is big endian.

[CR23-12s] If Chunk Data Payload is supported, then all chunk tags MUST use network byte order (Big-Endian).

[CR23-13s] If Chunk Data Payload is supported, then in order to minimize processing time on the GVSP receiver side, chunk data using multiple-byte in the payload data packet MUST be little-endian by default.

A GVSP transmitter or receiver may implement a converter to translate chunk data from little endian to big endian. This converter can be activated using the SCPSx bootstrap register when supported. The SCCx register indicates if big endian is supported by the given stream channel.

The table below describes the general structure of any chunk

Table 23-1: Chunk Data Content

Position	Format	Description
0	Data [K Bytes]	The data that the chunk is transporting. This section must be a multiple of 4 bytes. If it is not, the data has to be padded with zeros to a multiple of four bytes such that K is a multiple of 4 bytes. This ensures CID and Length fields are 32-bit aligned within the chunk.
K	CID [4 Bytes]	The chunk identifier
K+4	Length [4 Bytes]	The length of the data (in bytes, must be a multiple of 4)

The chunk structure with the tag behind the data allows GVSP transmitters to output data in chunk format as a stream, even in case of variable length data (such as in line scan applications).

A chunk data block consists of a sequence of chunks in chunk data format as depicted below:

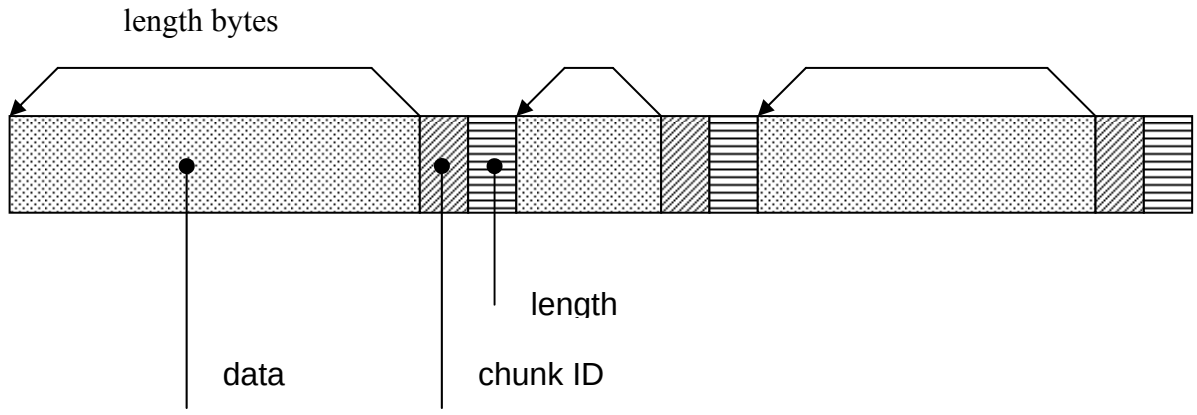


Figure 23-2: Chunk Data Diagram

[CR23-9st] If Chunk Data Payload is supported, a stream using the chunk data payload type MUST output data in the chunk data format

A block in chunk format is decoded with the help of a GenICam chunk parser. To let the chunk parser distinguish between the chunks added by multiple enabled chunk features, each chunk carries a chunk ID (CID). The CID for each chunk is transferred just before the chunk's length information.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

The chunk parser decodes a block in chunk format beginning with the last received data walking to the beginning of the block. The chunk parser exposes the chunk data as GenICam read-only features.

[CO23-10st] If Chunk Data Payload is supported, the chunks SHOULD be defined in the XML device description file.

23.4.1 Byte Ordering Example for Chunk Data Payload

The following figure shows the breakdown of the payload of a specific GVSP block in chunk format into byte fields.

Assume the following parameters:

- SCPSx is set to little endian (default)
- Image with a 1 x 1 AOI
  - ChunkOffsetX: 3
  - ChunkOffsetY: 6
  - Mono8 gray value: 0x83
- and a 32-bit frame counter with the value 0x00000008.

	Byte 0	Byte 1	Byte 2	Byte 3	Description
Image chunk	0x83	0x00	0x00	0x00	Image data + padding bytes
	0x61	0x7d	0x18	0xdb	Chunk id for image data (big endian)
	0x00	0x00	0x00	0x04	Length of the chunk (big endian)
Frame counter chunk	0x08	0x00	0x00	0x00	Frame counter (little endian)
	0x8c	0x0f	0x1c	0xd8	Chunk id for frame counter (big endian)
	0x00	0x00	0x00	0x04	Length of the chunk (big endian)
Image info chunk	0x03	0x00	0x06	0x00	ChunkOffsetX, ChunkOffsetY
	0x01	0x00	0x01	0x00	ChunkWidth, ChunkHeight
	0x23	0x0f	0x1c	0x23	Chunk id for image info (big endian)
	0x00	0x00	0x00	0x08	Length of the chunk (big endian)

23.4.2 GenICam Chunk Definition Example

The GenICam fragment to define the ImageInfo chunk in the above example is:

```
<RegisterDescription>
…
                <MaskedIntReg Name="ChunkOffsetXValue">
                        <Address>0x00</Address>
```

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

```
                <Length>4</Length>
                <AccessMode>RO</AccessMode>
                <pPort>ImageInfoPort</pPort>
                <LSB>16</LSB>
                <MSB>31</MSB>
                <Sign>Unsigned</Sign>
                <Endianess>LittleEndian</Endianess>
            </MaskedIntReg>
            <MaskedIntReg Name="ChunkOffsetYValue">
                <Address>0x00</Address>
                <Length>4</Length>
                <AccessMode>RO</AccessMode>
                <pPort>ImageInfoPort</pPort>
                <LSB>0</LSB>
                <MSB>15</MSB>
                <Sign>Unsigned</Sign>
                <Endianess>LittleEndian</Endianess>
            </MaskedIntReg>
            <MaskedIntReg Name="ChunkWidthValue">
                <Address>0x04</Address>
                <Length>4</Length>
                <AccessMode>RO</AccessMode>
                <pPort>ImageInfoPort</pPort>
                <LSB>16</LSB>
                <MSB>31</MSB>
                <Sign>Unsigned</Sign>
                <Endianess>LittleEndian</Endianess>
            </MaskedIntReg>
            <MaskedIntReg Name="ChunkHeightValue">
                <Address>0x04</Address>
                <Length>4</Length>
                <AccessMode>RO</AccessMode>
                <pPort>ImageInfoPort</pPort>
                <LSB>0</LSB>
                <MSB>15</MSB>
                <Sign>Unsigned</Sign>
                <Endianess>LittleEndian</Endianess>
            </MaskedIntReg>
        <Port Name="ImageInfoPort">
            <ChunkID>230f1c23</ChunkID>
        </Port>
</RegisterDescription>
```

Refer to the GenICam specification for additional information about Chunk Data.

23.5 Extended Chunk Data Payload Type

The chunk data payload type is frequently used to carry image data with additional information, i.e. it is often used as an extension of the image payload type. However, it misses some of the features of the image payload type which makes it practically impossible for a GVSP receiver to de-encapsulate the information without using the GenApi module of the GenICam standard. This is not ideal for GVSP receivers interested solely in the image itself given the fact that they have no way to know, at the GVSP level, whether the payload contains image or not and if so, where in the block the image is located.

A second issue with the chunk data payload type is that it is not possible for GVSP transmitters to notify GVSP receivers when the layout of the chunk data layout changes; for instance when a chunk is not available in a given block. In other words, GVSP receivers don't know when they need to re-parse the chunk layout.

This version of the specification addresses the issue by introducing a new payload type called extended chunk data. This payload type enables GVSP transmitters to append additional information to a fixed position image chunk so that a GVSP receiver interested only in the image data can extract the image data without using the GenApi module of GenICam. It also enables a GVSP transmitter to notify receivers that the layout of the data payload format has changed between 2 blocks.

[CR23-14s] When the Extended Chunk Data payload type is used and an image is present in the block, the image data chunk MUST be the first chunk in the data payload of the block. It MUST be aligned, with an offset of zero, with the beginning of the data payload of the block.

23.6 Device-specific Payload Type

This payload type can be used to transfer device-specific information.

~~[CO23-11d] If Device-specific payload is supported, its content SHOULD be defined in the XML device description file or in documentation provided with the device.~~

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

24 Data Packet Headers

[R24-1s] The data packet header illustrated in Figure 24-1 MUST be available on all GVSP packets.

The protocol header does not include the length field: GVSP receivers use the UDP length information to determine the packet size.

With the exception of the last data packet, which may be smaller to match the data payload size, all data payload packets are the same size. However, it is also possible to pad the last data packet so that all data payload packets are exactly the same size. The management entity of the system is responsible to optimize packet size. This could be achieved using the test packet for instance. This facilitates the positioning of information by the GVSP receiver in the buffer if a packet is missing.

[R24-2s] All GVSP headers MUST use network byte order (Big-endian).

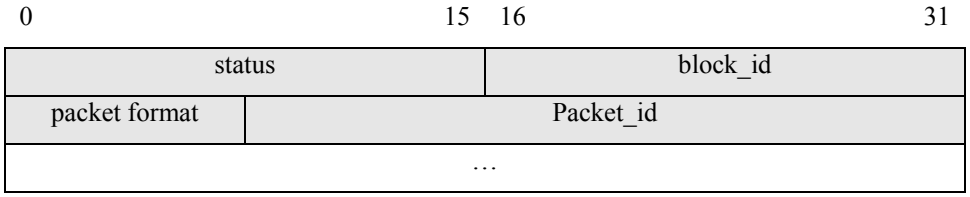


Figure 24-1: Data Packet Header

DATA PACKET HEADER	
status	Status of the streaming operation (see status code section)
block_id	ID of the data block. Sequential and incrementing starting at 1. A block_id of 0 is reserved for the test packet. Block_id wraps-around to 1 when it reaches the limit of the 16-bit.
packet format	• DATA_LEADER_FORMAT (1): The packet contains a data leader. • DATA_TRAILER_FORMAT (2): The packet contains a data trailer. • DATA_PAYLOAD_FORMAT (3): The packet contains a part of the data payload. • All other values are reserved.
packet_id	ID of packet in the block. The packet_id is reset to 0 at the start of each data block. Packet_id 0 is thus the data leader for the current block_id.

24.1 Data Leader

[R24-3st] The data leader MUST be the first packet of the block.

[R24-4st] The data leader MUST be sent in a separate packet with packet_id set to 0. It MUST fit in one packet of 576 bytes at most (including the IP, UDP and GVSP header).

The data leader follows the template of Figure 24-2. For different payload type, other information might be appended.

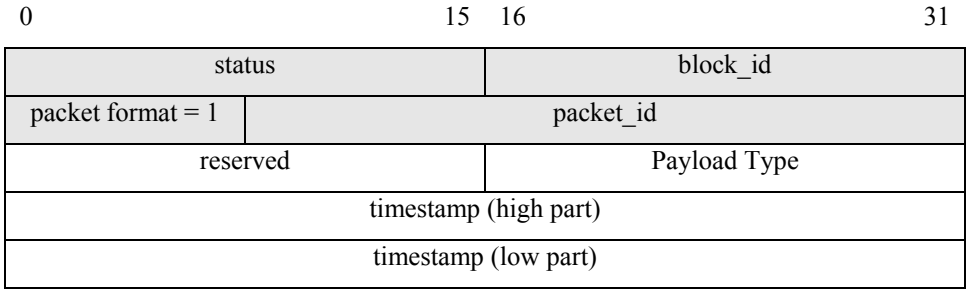


Figure 24-2: Generic Data Leader

DATA LEADER	
Reserved	Always 0
Payload Type	Unique ID identifying the type of data block we can expect to receive. • 0x0001: image • 0x0002: raw data • 0x0003: file • 0x0004: chunk data • 0x0005: extended chunk data • 0x8000 and above are device-specific
timestamp	64-bit timestamp when the block of data was generated. Timestamps are optional. This field should be 0 when timestamps are not supported. Multiple ROI's from the same frame (exposure) should have the same timestamp.

The following sections depict the data leader header associated to each payload type.

24.1.1 Payload Type = Image

[CR24-5s] If Image payload is supported, its Data Leader MUST follow the layout of Figure 24-3.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

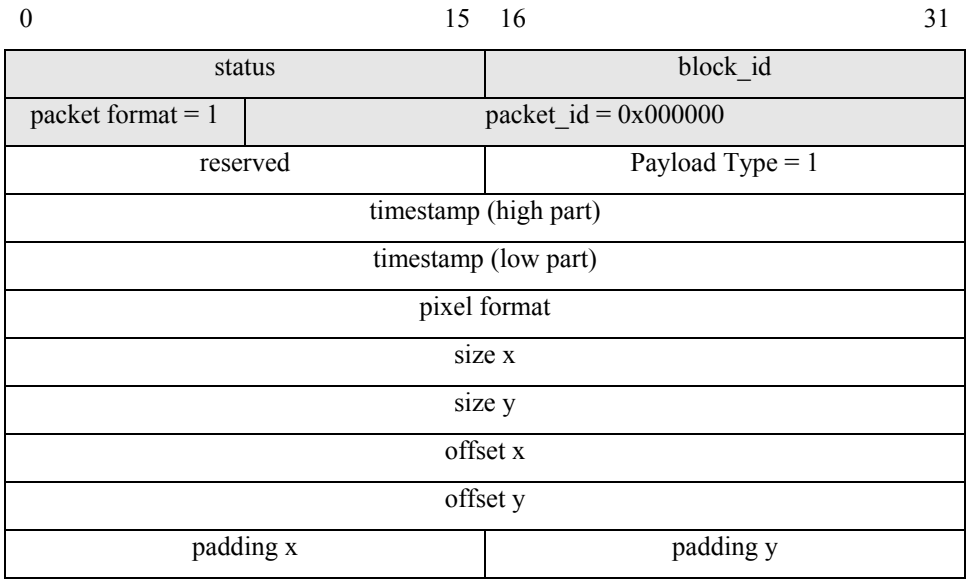


Figure 24-3: Image Data Leader

IMAGES DATA LEADER	
pixel format	This field gives the pixel format of payload data. Refer to Pixel Layout section.
size x	Width in pixels. The actual width received can be lower than the maximum supported by the image source when a ROI is set. When no ROI is defined, this field must be set to the maximum number of pixels supported by the source.
size y	Height in lines. The actual number of lines received can be lower than the maximum supported by the image source when a ROI is set. When no ROI is defined, this field must be set to the maximum number of lines supported by the source.
offset x	Offset in pixels from image origin. Used for ROI support. When no ROI is defined this field must be set to 0.
offset y	Offset in lines from image origin. Used for ROI support. When no ROI is defined this field must be set to 0.
padding x	Horizontal padding expressed in bytes. Number of extra bytes transmitted at the end of each line to facilitate image alignment in buffers. This can typically used to have 32-bit aligned image lines. This is similar to the horizontal invalid (or horizontal blanking) in analog cameras. Set to 0 when no horizontal padding is used.
padding y	Vertical padding expressed in bytes. Number of extra bytes transmitted at the end of the image to facilitate image alignment in buffers. This could be used to align buffers to certain block size (for instance 4 KB). This is similar to the vertical invalid (or vertical blanking) in analog cameras. Set to 0 when no vertical padding is used.

Note: When they are transmitted on the same stream channel, each ROI must be transmitted with a different block_id so the data leader correctly reflects ROI position within the original image. When they are transmitted on different stream channels (one ROI per stream channel), then they might use the same block_id value to facilitate matching.

24.1.2 Payload Type = Raw Data

[CR24-6s] If Raw Data payload is supported, its Data Leader MUST follow the layout of Figure 24-4.

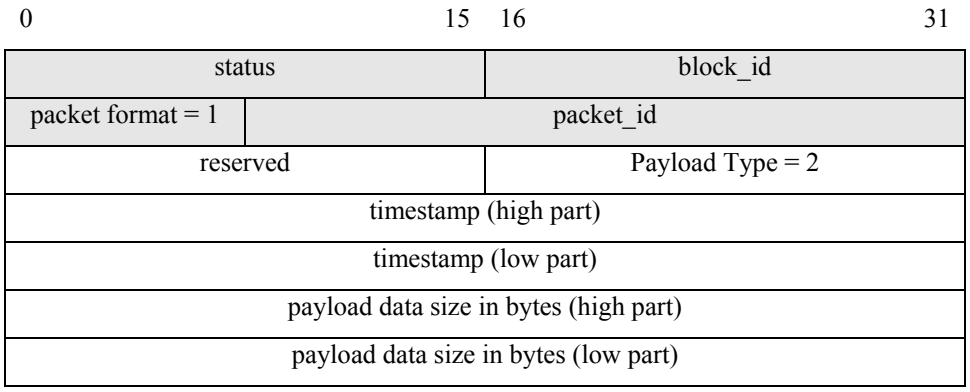


Figure 24-4: Raw Data Leader

RAW DATA LEADER	
Payload data size	Total size of payload data in bytes

24.1.3 Payload Type = File

[CR24-7s] If File payload is supported, its Data Leader MUST follow the layout of Figure 24-5.

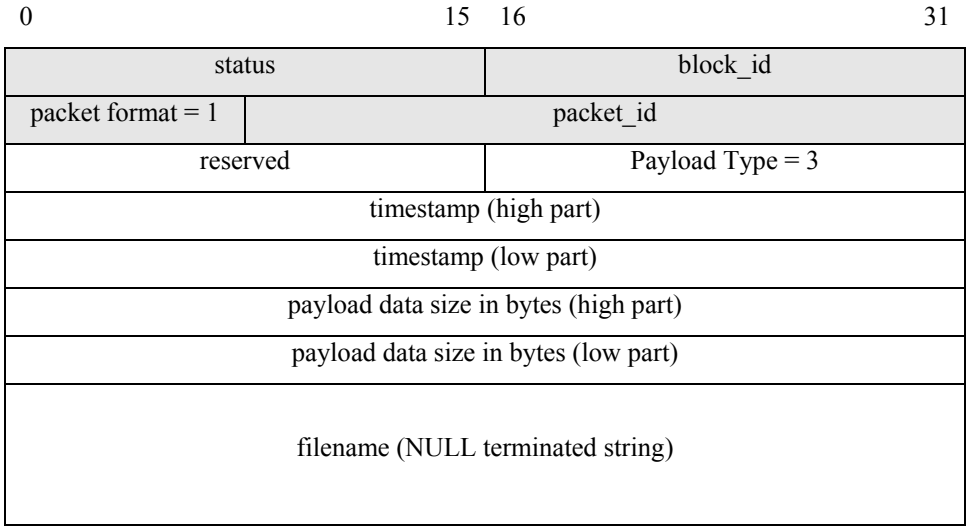


Figure 24-5: File Data Leader

FILE DATA LEADER	
Payload data size	Total size of payload data in bytes
Filename	NULL terminated string using the current character set.

24.1.4 Payload Type = Chunk Data

[CR24-8s] If Chunk Data payload is supported, its Data Leader MUST follow the layout of Figure 24-6

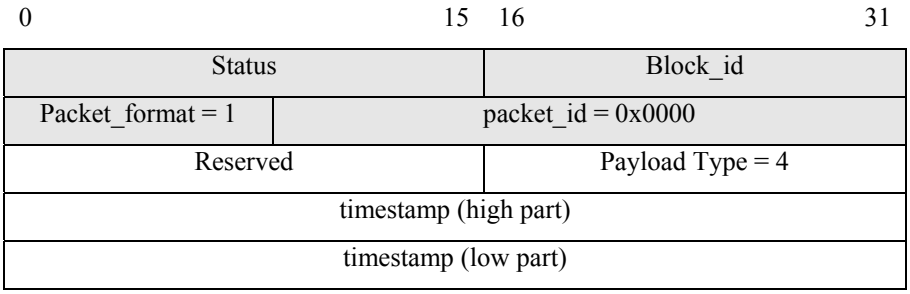


Figure 24-6: Chunk Data Leader

24.1.5 Payload Type = Extended Chunk Data

[CR24-31s] If Extended Chunk Data payload is supported, its Data Leader MUST follow the layout of Figure 24-7.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

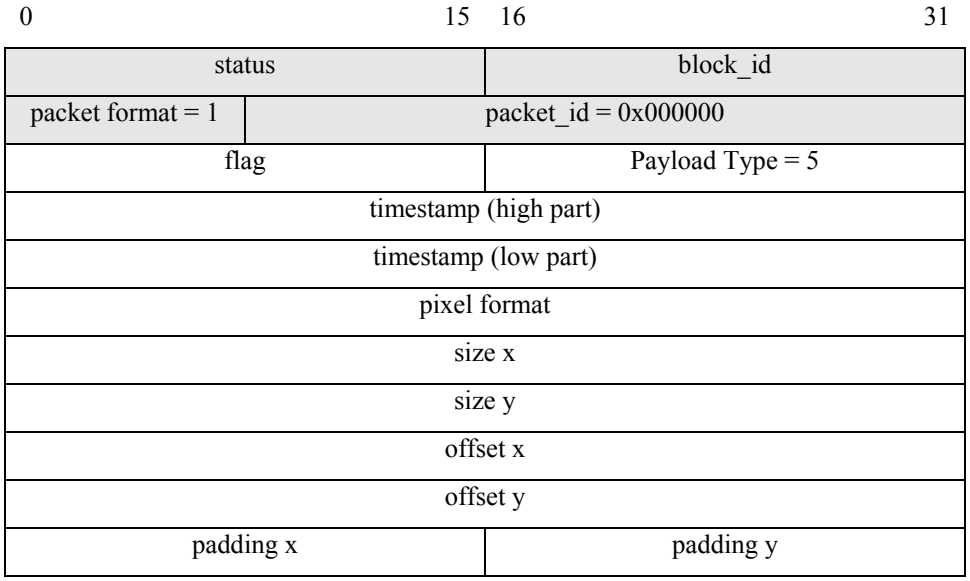


Figure 24-7: Extended Chunk Data Leader

flag	16 bits	bit 15 (least significant bit) – Image Chunk: When set, indicates if an image chunk is present in the block. bits 0 to 14 – Reserved, set to 0.

The pixel format, size, offset and padding fields of the leader are define as per Section 24.1.1 when the image chunk bit of the flag field is set. Otherwise, they are set to 0.

24.1.6 Payload Type = Device-specific

[CR24-9s] If Device-specific payload is supported, its Data Leader MUST follow the layout of Figure 24-8.

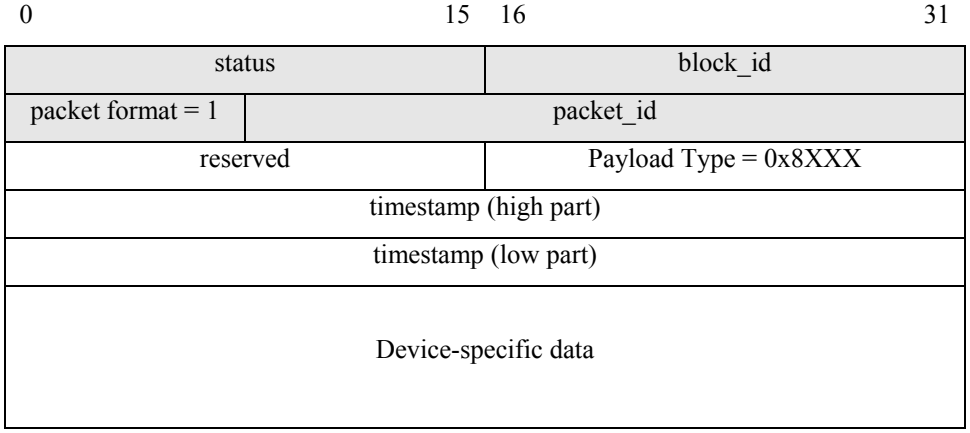


Figure 24-8: Device-specific Data Leader

24.2 Data Payload

Data payload packets transport the actual information to a GVSP receiver. There might be up to 16 millions data payload packets per block (24-bit packet_id field).

For images, multiple ROIs from the same frame can be provided either as a sequence on a single stream channel (each with a different block_id), or on different stream channels.

[CR24-10st] If an image source supports multiple ROI, then overlapping data from multiple ROIs MUST be retransmitted with each ROI so that each image is provided completely in its block.

[CR24-11st] If a camera device supports multiple ROI, then different ROIs from the same exposure MUST use the same timestamp.

The last data payload packet of a block might have a smaller size than the other payload packets since total amount of data to transfer is not necessarily a multiple of the packet size.

[R24-12st] All data payload packets except the last one MUST use the requested packet_size.

This way, a GVSP receiver can quickly compute the location in the image of the first pixel in the packet by multiplying packet_size with "packet_id – 1".

The following sections depict the data leader header associated to each payload type.

24.2.1 Payload Type = Image

[CR24-13s] If Image payload is supported, its Data Payload MUST follow layout of Figure 24-9

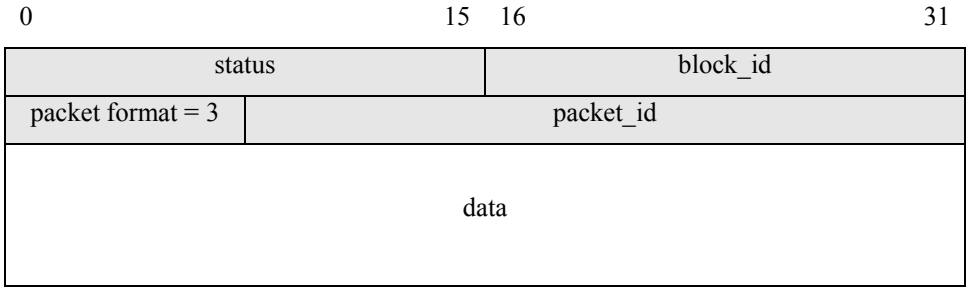


Figure 24-9: Image Data Payload

DATA PAYLOAD	
data	Image data formatted as specified in the Data Leader pixel format field. For Data Payload packets, the IP Header + UDP Header + GVSP Header + data must be equal to the packet size specified in the Stream Channel Packet Size register of the GVSP transmitter (See GVCP). The only exception is the last Data Payload packet which may be smaller than the specified packet size. However, it is also possible to pad the last data packet so that all data payload packets are exactly the same size.

24.2.2 Payload Type = Raw data

[CR24-14s] If Raw payload is supported, its Data Payload MUST follow layout of Figure 24-10.

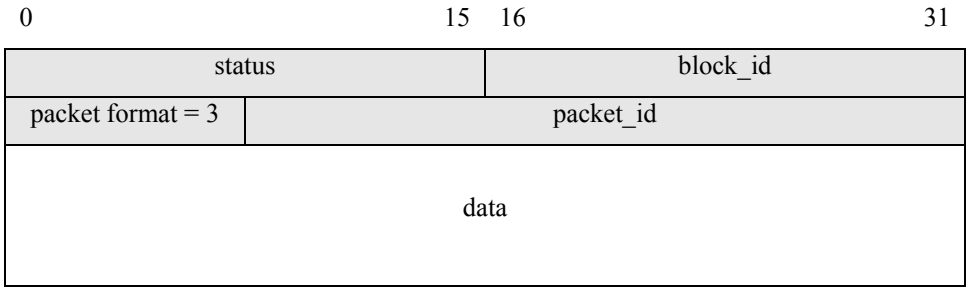


Figure 24-10: Raw Data Payload

DATA PAYLOAD	
data	Image data. For Data Payload packets, the IP Header + UDP Header + GVSP Header + data must be equal to the packet size specified in the Stream Channel Packet Size register of the GVSP transmitter (See GVCP). The only exception is the last Data Payload packet which may be smaller than the specified packet size. However, it is also possible to pad the last data packet so that all data payload packets are exactly the same size.

24.2.3 Payload Type = File

[CR24-15s] If File payload is supported, its Data Payload MUST follow layout of Figure 24-11.

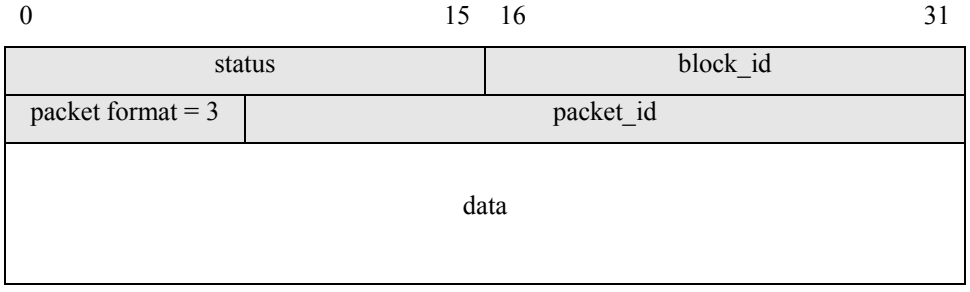


Figure 24-11: File Data Payload

DATA PAYLOAD	
data	Image data. For Data Payload packets, the IP Header + UDP Header + GVSP Header + data must be equal to the packet size specified in the Stream Channel Packet Size register of the GVSP transmitter (See GVCP). The only exception is the last Data Payload packet which may be smaller than the specified packet size. However, it is also possible to pad the last data packet so that all data payload packets are exactly the same size.

24.2.4 Payload Type = Chunk Data or Extended Chunk Data

[CR24-16s] If Chunk Data or Extended Chunk Data payload is supported, its Data Payload MUST follow the layout of Figure 24-12.

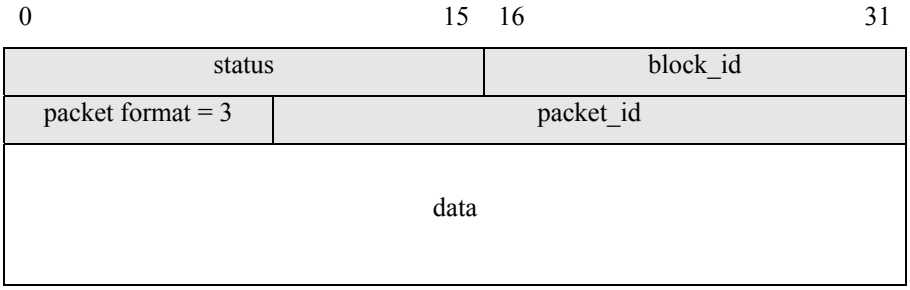


Figure 24-12: Chunk Data and Extended Chunk Data Payload

DATA PAYLOAD	
data	Chunk data. For chunk data or extended chunk data payload packets, the IP Header + UDP Header + GVSP Header + data must be equal to the packet size specified in the Stream Channel Packet Size register of the GVSP transmitter (See GVCP). The only exception is the last Data Payload packet which may be smaller than the specified packet size. However, it is also possible to pad the last data packet so that all data payload packets are exactly the same size.

24.2.5 Payload Type = Device-specific

[CR24-17s] If Device-specific payload is supported, its Data Payload MUST follow the layout of Figure 24-13.

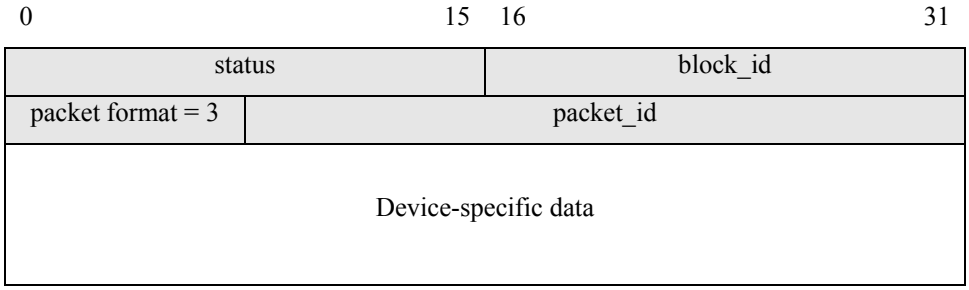


Figure 24-13: Device-specific Data Payload

DATA PAYLOAD	
data	Image data. For Data Payload packets, the IP Header + UDP Header + GVSP Header + data must be equal to the packet size specified in the Stream Channel Packet Size register of the GVSP transmitter (See GVCP). The only exception is the last Data Payload packet which may be smaller than the specified packet size. However, it is also possible to pad the last data packet so that all data payload packets are exactly the same size.

24.3 Data Trailer

[R24-18st] The data trailer MUST be the last packet of the block.

[R24-19st] The data trailer MUST be sent as a separate packet that fits in one packet of 576 bytes at most (including the IP, UDP and GVSP header).

[R24-20st] After the data trailer packet is sent, the block_id MUST be incremented and the packet_id MUST be reset to 0 for the next block.

[R24-21s] The Data Trailer MUST follow the layout of Figure 24-14.

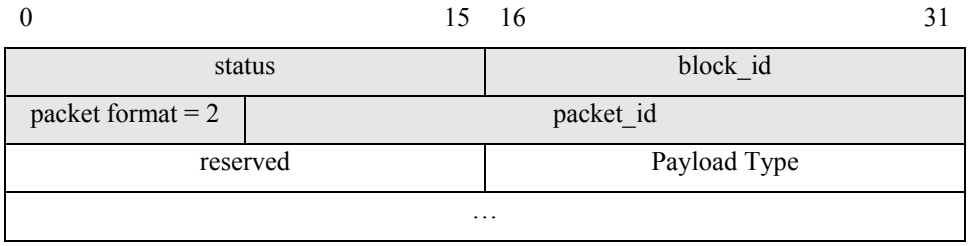


Figure 24-14: Generic Data Trailer

DATA TRAILER	
Payload Type	Unique ID identifying the type of data block we can expect to receive. • 0x0001: Image • 0x0002: Raw data • 0x0003: File • 0x0004: Chunk data • 0x0005: Extended chunk data • 0x8000 and above are device-specific

24.3.1 Payload Type = Image

[CR24-22s] If Image payload is supported, its Data Trailer MUST follow the layout of Figure 24-15.

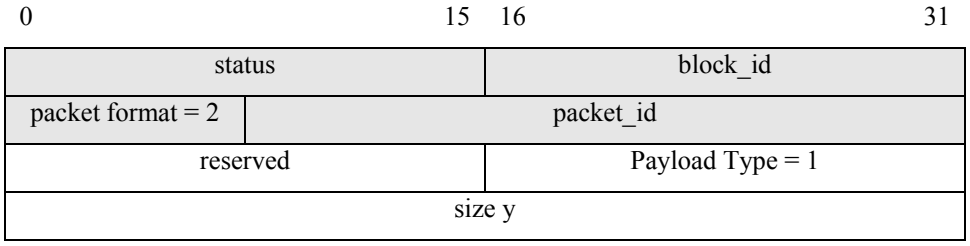


Figure 24-15: Image Data Trailer

IMAGES DATA TRAILER	
size y	The size y is put again in the data trailer. This is done to support variable frame size image sources. Note that the size y specified in the data leader is the maximum supported by the source (when no ROI is defined). This field is the actual image height, in lines, for this particular block.

24.3.2 Payload Type = Raw data

[CR24-23s] If Raw payload is supported, its Data Trailer MUST follow the layout of Figure 24-16.

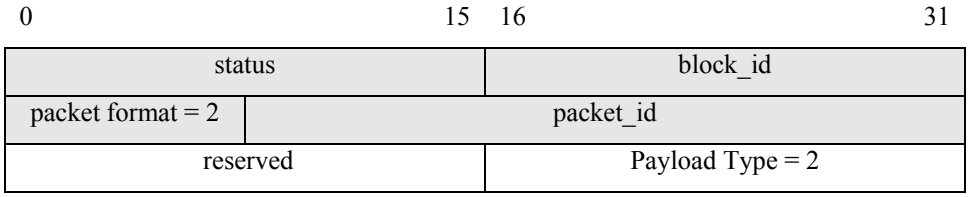


Figure 24-16: Raw Data Trailer

24.3.3 Payload Type = File

[CR24-24s] If File payload is supported, its Data Trailer MUST follow the layout of Figure 24-17.

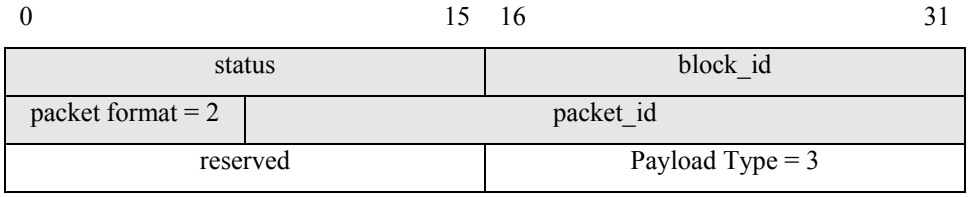


Figure 24-17: File Data Trailer

24.3.4 Payload Type = Chunk Data

[CR24-25s] If Chunk Data Payload is supported, its Data Trailer MUST follow the layout of Figure 24-18.

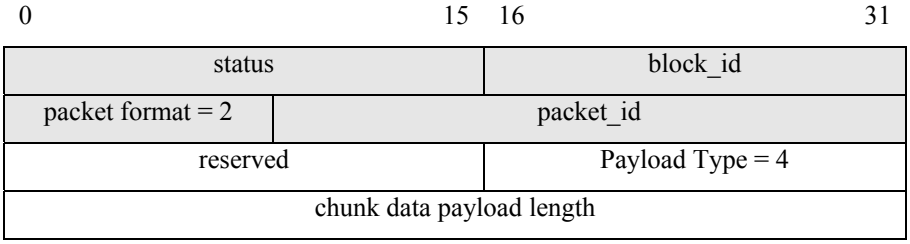


Figure 24-18: Chunk Data Trailer

CHUNK DATA TRAILER	
chunk data payload length	Total length of the chunk data payload in bytes. It must be a multiple of 4 bytes (bit 0 and 1 must be cleared). This field is put here to help GVSP receivers finding the end of the chunk data payload.

24.3.5 Payload Type = Extended Chunk Data

[CR24-32s] If Extended Chunk Data Payload is supported, its Data Trailer MUST follow the layout of Figure 24-19.

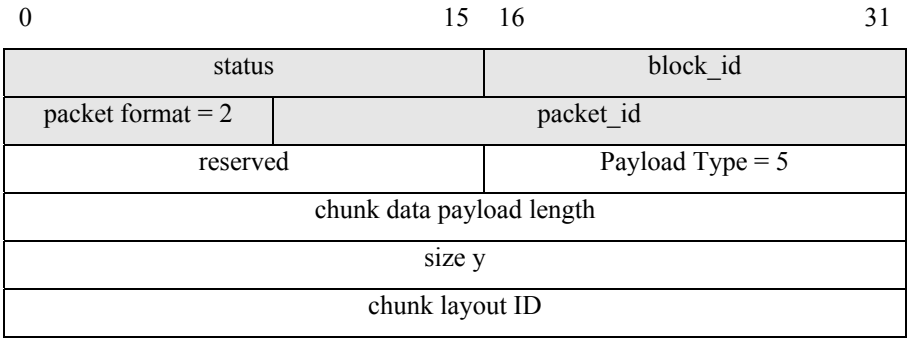


Figure 24-19: Extended Chunk Data Trailer

CHUNK DATA TRAILER	
chunk data payload length	As per corresponding field in the chunk data trailer (Section 24.3.4).
size y	When an image chunk is present in the block, this field is set as per its corresponding field in the image data trailer (Section 24.3.1). Otherwise, it is set to 0. Please refer to Section 24.1.5 for details on how to determine if an image chunk is present in the block.
chunk layout ID	This field serves as an indicator that the chunk layout has changed and the GVSP receiver should re-parse the chunk layout in the buffer. When a chunk layout (availability or position of individual chunks) changes since the last block, the GVSP transmitter MUST change the chunk layout ID. As long as the chunk layout remains the same, the GVSP transmitter MUST keep the chunk layout ID identical. When switching back to a layout, which was already used before, the GVSP transmitter can use the same ID that was used then or use a new ID. A chunk layout ID value of 0 is invalid. It is reserved to be used by GVSP transmitters not supporting the layout ID functionality. If the GVSP transmitter does not support the chunk layout ID functionality, it MUST set this field to 0 for all blocks. The algorithm used to compute the chunk layout ID is left as quality of implementation.

24.3.6 Payload Type = Device-specific

[CR24-26s] If Device-specific payload is supported, its Data Trailer MUST follow the layout of Figure 24-20.

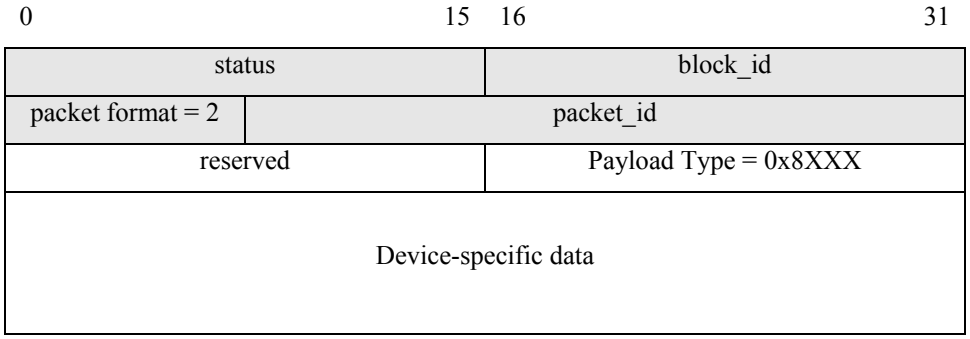


Figure 24-20: Device-specific Data Trailer

24.4 Test packet

A management entity can use test packets to discover the MTU of the connection. This is done on each network interfaces associated to a GVSP transmitter using the stream channel registers.

[R24-27st] In a test packet, the block_id MUST be 0. All other information in the test packet header (status, packet format and packet_id fields) is “don’t care”.

[CR24-29st] If the GVSP transmitter supports the "Test Data" capability, it MUST fill the payload section of the test packet with the output of the linear feedback shift register (LFSR) generator provided below. Otherwise, the payload data is "don't care". This is required for all the network interfaces associated to a GVSP transmitter.

[CO24-30sr] If the GVSP transmitter supports the "Test Data" capability for the test packet and it fires a test packet, then a GVSP receiver and its application requesting this test packet SHOULD compare the content of the payload against the data generated by the LFSR generator provided below.

Note the data provided by the LFSR generator does not apply to the header of the test packet.

A GVSP transmitter could create a test packet by clearing the GVSP header and data fields of the packet. Obviously, Ethernet, IP and UDP headers must have valid values (they are not cleared).

[R24-28st] The Test Packet MUST follow the layout of Figure 24-21.

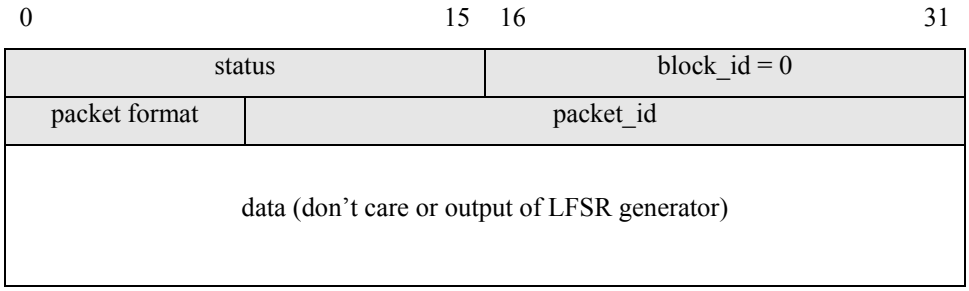


Figure 24-21: Test Packet

Test packet	
data	Filled with don't care data or output of LFSR generator. Total data must be such that IP Header + UDP Header + GVSP Header + data is equal to the packet size specified in the Stream Channel Packet Size register of the GVSP transmitter (See GVCP). The don't fragment bit must also be set in the IP Header (the don't fragment bit is only used for the test packet; all other stream channel packets do not use this feature).

24.4.1 LFSR Generator

The data for the test packet is generated with a maximum sequence 16-bit LFSR. This allows a GVSP transmitter implementation using minimum resources. The period of such an LFSR is (2^16 -1). The LFSR is clocked once for every byte of output data generated. The byte is filled with the LSB byte of the shift register.

The generator has the following properties:

- 16-bit, right-shifted
- initial value = 0xFFFF
- Polynomial (feedback connection) = 0x8016

Here is a possible VHDL pseudo-implementation:

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity NumberGenerator is
	port(
		iClock  : in std_logic;
		iStart  : in std_logic;
		iEnable : in std_logic;
		oData   : out std_logic_vector(7 downto 0)
	);
end NumberGenerator;

architecture behavior of NumberGenerator is
	constant POLYNOMIAL  : std_logic_vector(15 downto 0) := X"8016";
	constant INIT_VALUE  : std_logic_vector(15 downto 0) := X"FFFF";
	signal ShiftRegister : std_logic_vector(15 downto 0);
begin
	process(iClock)
		variable Temp    : std_logic_vector(15 downto 0);
	begin
		if rising_edge(iClock) then
			if iStart = '1' then
				ShiftRegister <= INIT_VALUE;
			elsif iEnable = '1' then
				Temp := ShiftRegister;
			     if Temp(0) = '1' then
					Temp := '0' & Temp(15 downto 1);
					Temp := Temp xor POLYNOMIAL;
				else
					Temp := '0' & Temp(15 downto 1);
				end if;
				ShiftRegister <= Temp;
			end if;
		end if;
	end process;

	oData <= ShiftRegister(7 downto 0);

end behavior;
```

Here is a possible C pseudo-implementation:

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

```
unsigned short POLYNOMIAL = 0x8016;
unsigned short INIT_VALUE = 0xFFFF;

// reset to init value
unsigned short lfsr = INIT_VALUE;
// the output byte
unsigned char  databyte;

// loop kernel for each databyte
        databyte = (lfsr & 0xff); /// LSB byte of lfsr
        // calculate next lfsr state
      lfsr = ((lfsr >> 1) ^ (-(lfsr & 1) & POLYNOMIAL));
```

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

25 Pixel Layouts

This section describes the various pixel and image layouts supported by GigE Vision.

[O25-1st] GVSP transmitters providing pixel depths greater than 8 bits SHOULD also provide an 8 bit transfer mode to facilitate display of images by GVSP receivers. This mode is realized by removing the least significant bits.

The mechanism to select the pixel depth is supplied in the XML device description file.

25.1 Image Formats

The figures in this section illustrate the upper-left portion of the image coming out of an image source. This clearly indicates the location of the various pixels in an image as seen by a video source. Row number starts at 1. Column number starts at 1.

For a given pixel, first index represents the row, second index represents the column. Ex: Y24 is the pixel located on the second row, fourth column.

- Y represents Luminance.
- R represents Red color component
- G represents Green color component
- B represents Blue color component
- U represents first chrominance in YUV format
- V represents second chrominance in YUV format

25.1.1 Mono

If signed, then MSB is the sign bit (2nd complement).

	1	2	3	4	…
1	Y11	Y12	Y13	Y14	…
2	Y21	Y22	Y23	Y24	…
3	Y31	Y32	Y33	Y34	…
4	Y41	Y42	Y43	Y44	…
…	…	…	…	…	…

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

25.1.2 RGB

	1	2	3	4	…
1	RGB11	RGB12	RGB13	RGB14	…
2	RGB21	RGB22	RGB23	RGB24	…
3	RGB31	RGB32	RGB33	RGB34	…
4	RGB41	RGB42	RGB43	RGB44	…
…	…	…	…	…	…

25.1.3 BGR

	1	2	3	4	…
1	BGR11	BGR12	BGR13	BGR14	…
2	BGR21	BGR22	BGR23	BGR24	…
3	BGR31	BGR32	BGR33	BGR34	…
4	BGR41	BGR42	BGR43	BGR44	…
…	…	…	…	…	…

25.1.4 BayerGR

	1	2	3	4	…
1	G11	R12	G13	R14	…
2	B21	G22	B23	G24	…
3	G31	R32	G33	R34	…
4	B41	G42	B43	G44	…
…	…	…	…	…	…

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

25.1.5 BayerRG

	1	2	3	4	…
1	R11	G12	R13	G14	…
2	G21	B22	G23	B24	…
3	R31	G32	R33	G34	…
4	G41	B42	G43	B44	…
…	…	…	…	…	…

25.1.6 BayerGB

	1	2	3	4	…
1	G11	B12	G13	B14	…
2	R21	G22	R23	G24	…
3	G31	B32	G33	B34	…
4	R41	G42	R43	G44	…
…	…	…	…	…	…

25.1.7 BayerBG

	1	2	3	4	…
1	B11	G12	B13	G14	…
2	G21	R22	G23	R24	…
3	B31	G32	B33	G34	…
4	G41	R42	G43	R44	…
…	…	…	…	…	…

25.1.8 YUV411

	1	2	3	4	5	…
1	YUV11	Y12	Y13	Y14	YUV15	…
2	YUV21	Y22	Y23	Y24	YUV25	…
3	YUV31	Y32	Y33	Y34	YUV35	…
4	YUV41	Y42	Y43	Y44	YUV45	…
…	…	…	…	…	…	…

25.1.9 YUV422

	1	2	3	4	…
1	YUV11	Y12	YUV13	Y14	…
2	YUV21	Y22	YUV23	Y24	…
3	YUV31	Y32	YUV33	Y34	…
4	YUV41	Y42	YUV43	Y44	…
…	…	…	…	…	…

25.1.10 YUV444

	1	2	3	4	…
1	YUV11	YUV12	YUV13	YUV14	…
2	YUV21	YUV22	YUV23	YUV24	…
3	YUV31	YUV32	YUV33	YUV34	…
4	YUV41	YUV42	YUV43	YUV44	…
…	…	…	…	…	…

25.2 Pixel Alignment

[R25-2st] In order to minimize pixel processing time on the GVSP receiver side, pixel data using multiple-byte in the payload data packets MUST be little-endian by default.

A GVSP transmitter may implement a converter to translate pixels from little-endian to big-endian. This converter can be activated using the SCPSx bootstrap register when supported. The SCCx register indicates if this converter is supported.

Note that in the following figures, the least significant bit is on the left side. The most significant bit is on the right side. This is the little-endian convention which is different from the big-endian convention used for GVCP and for the headers of GVSP.

In the following figures, Byte 0 is sent first on the wire, followed by Byte 1 and so on for little-endian mode.

25.2.1 Align_Mono8

8-bit data into 8 bits

Description : 8-bit monochrome unsigned
Pixel Size : 1 byte
Value range : 0 to 255

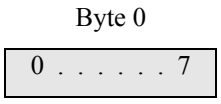


Figure 25-1: Align_Mono8

25.2.2 Align_Mono8Signed

8-bit data into 8 bits (with sign)

Description : 8-bit monochrome signed
Pixel Size : 1 byte
Value Range : -128 to 127

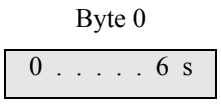


Figure 25-2: Align_Mono8Signed

25.2.3 Align_Mono10

10-bit data into 16-bits

Description : 10-bit monochrome
Pixel Size : 2 bytes
Value range : 0 to 1023

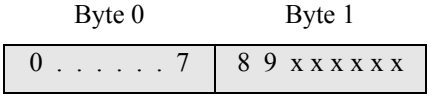


Figure 25-3: Align_Mono10

25.2.4 Align_Mono10Packed

2 x 10-bit data packed into 24 bits (Y1, Y2)

Description : 10-bit packed monochrome unsigned
Pixel Size : 3 bytes for two pixels
Value range : 0 to 1023

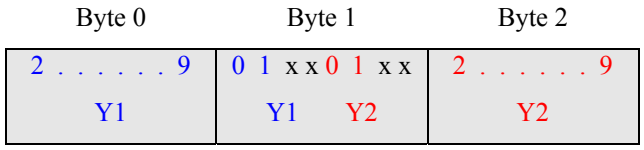


Figure 25-4: Align_Mono10Packed

25.2.5 Align_Mono12

12- bit data into 16 bits

Description : 12-bit monochrome unsigned
Pixel Size : 2 bytes
Value range : 0 to 4095

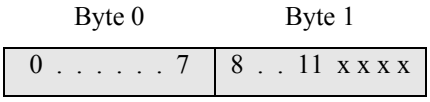


Figure 25-5: Align_Mono12

25.2.6 Align_Mono12Packed

2 x 12-bit data packed into 24 bits (Y1, Y2)

Description : 12-bit packed monochrome unsigned
Pixel Size : 3 bytes for two pixels
Value range : 0 to 1023

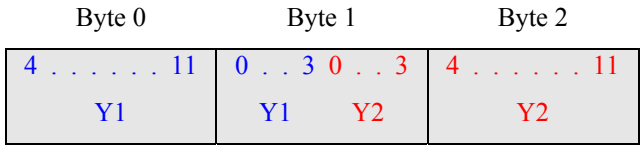
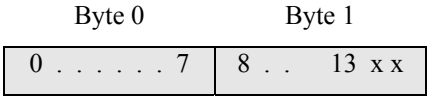


Figure 25-6: Align_Mono12Packed

25.2.7 Align_Mono14

14-bit data into 16 bits.

Description : 14-bit monochrome unsigned
Pixel Size : 2 bytes
Value range : 0 to 16383



25.2.8 Align_Mono16

16-bit data into 16 bits

Description : 16-bit monochrome unsigned
Pixel Size : 2 bytes
Value range : 0 to 65535

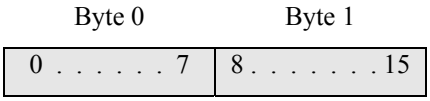


Figure 25-8: Align_Mono16

25.2.9 Align_ABC10Packed_V1

3 x 10-bit data packed into 32 bits (A, B, C)

Description : 3 x 10-bit packed unsigned version 1
Pixel Size : 4 bytes
Value range : 0 to 1023

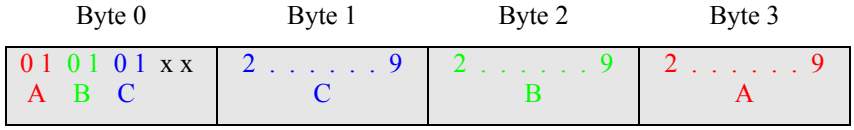


Figure 25-9: Align_ABC10Packed_V1

25.2.10 Align_ABC10Packed_V2

3 x 10-bit data packed into 32 bits (A, B, C)

Description : 3 x 10-bit packed unsigned version 2
Pixel Size : 4 bytes
Value range : 0 to 1023

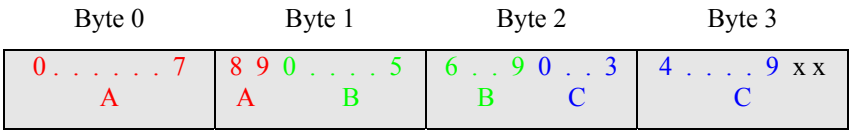


Figure 25-10: Align_ABC10Packed_V2

25.2.11 Align_ABC565Packed

A combination of 3 color components packed into 16 bits, with 5 bits for the first and third components and 6 bits for the second component.

Description : 3 color components (A, B, C) packed in 16-bit unsigned
Pixel Size : 2 bytes
Value range : 0 to 31 for the 5-bit components, 0 to 63 for the 6-bit component

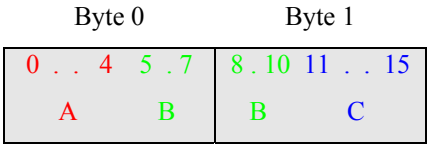


Figure 25-11: Align_565Packed

25.3 Pixel Formats

This section illustrates the various pixel formats natively supported by GVSP.

[O25-3s] A GVSP transmitter or a GVSP receiver SHOULD support a subset of the pixel formats listed in this section.

Note: Device-specific pixel formats are supported by setting bit 31 of the pixel format value.

25.3.1 GVSP_PIX_MONO8

[CR25-4s] If supported, GVSP_PIX_MONO8 MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono8	
Image Format	Mono	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x00000001
Pixel Format:	GVSP_PIX_MONO8	0x01080001

Byte 0
Y11
Y12
Y13
Y14
…
Y21
…

25.3.2 GVSP_PIX_MONO8_SIGNED

[CR25-5s] If supported, GVSP_PIX_MONO8_SIGNED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono8Signed	
Image Format	Mono	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x00000002
Pixel Format:	**GVSP_PIX_MONO8 SIGNED**	0x01080002

Byte 0
Y11
Y12
Y13
Y14
…
Y21
…

25.3.3 GVSP_PIX_MONO10

[CR25-6s] If supported, GVSP_PIX_MONO10 MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono10	
Image Format	Mono	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000003
Pixel Format:	**GVSP_PIX_MONO10**	0x01100003

Byte 0 Byte1
Y11
Y12
Y13
Y14
…
Y21
…

25.3.4 GVSP_PIX_MONO10_PACKED

[CR25-7s] If supported, GVSP_PIX_MONO10_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono10Packed	
Image Format	Mono	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x00000004
Pixel Format:	**GVSP_PIX_MONO10_PACKED**	0x010C0004

Byte 0	Byte1	Byte2
Y11/Y12		
Y13/Y14		
…		

Note that the first pixel of the second line will not start aligned on a 3-byte boundary when the image width contains an odd number of pixels.

25.3.5 GVSP_PIX_MONO12

[CR25-8s] If supported, GVSP_PIX_MONO12 MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono12	
Image Format	Mono	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000005
Pixel Format:	**GVSP_PIX_MONO12**	0x01100005

Byte 0	Byte1
Y11	
Y12	
Y13	
Y14	
…	
Y21	
…	

25.3.6 GVSP_PIX_MONO12_PACKED

[CR25-9s] If supported, GVSP_PIX_MONO12_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono12Packed	
Image Format	Mono	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x00000006
Pixel Format:	**GVSP_PIX_MONO12_PACKED**	0x010C0006

Byte 0	Byte1	Byte2
Y11/Y12		
Y13/Y14		
…		

Note that the first pixel of the second line will not start aligned on a 3-byte boundary when the image width contains an odd number of pixels.

25.3.7 GVSP_PIX_MONO14

[CR25-41s] If supported, GVSP_PIX_MONO14 MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono14	
Image Format	Mono	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000025
Pixel Format:	**GVSP_PIX_MONO14**	0x01100025

Byte 0 Byte1
Y11
Y12
Y13
Y14
…
Y21
…

25.3.8 GVSP_PIX_MONO16

[CR25-10s] If supported, GVSP_PIX_MONO16 MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono16	
Image Format	Mono	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000007
Pixel Format:	**GVSP_PIX_MONO16**	0x01100007

Byte 0 Byte1
Y11
Y12
Y13
Y14
…
Y21
…

25.3.9 GVSP_PIX_BAYGR8

[CR25-11st] All Bayer pixel formats MUST represent the Bayer pattern of the sensor.

[CR25-12s] If supported, GVSP_PIX_BAYGR8 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Image Format	BayerGR	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x00000008
Pixel Format:	**GVSP_PIX_BAYGR8**	0x01080008

Byte 0
G11
R12
G13
R14
…
B21
G22
…

25.3.10 GVSP_PIX_BAYRG8

[CR25-13s] If supported, GVSP_PIX_BAYRG8 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Image Format	BayerRG	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x00000009
Pixel Format:	**GVSP_PIX_BAYRG8**	0x01080009

Byte 0
R11
G12
R13
G14
…
G21
B22
…

25.3.11 GVSP_PIX_BAYGB8

[CR25-14s] If supported, GVSP_PIX_BAYGB8 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Image Format	BayerGB	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x0000000A
Pixel Format:	**GVSP_PIX_BAYGB8**	0x0108000A

Byte 0
G11
B12
G13
B14
…
R21
G22
…

25.3.12 GVSP_PIX_BAYBG8

[CR25-15s] If supported, GVSP_PIX_BAYBG8 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Image Format	BayerBG	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x0000000B
Pixel Format:	**GVSP_PIX_BAYBG8**	0x0108000B

Byte 0
B11
G12
B13
G14
…
G21
R22
…

25.3.13 GVSP_PIX_BAYGR10

[CR25-16s] If supported, GVSP_PIX_BAYGR10 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Image Format	BayerGR	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000000C
Pixel Format:	**GVSP_PIX_BAYGR10**	0x0110000C

Byte 0 Byte1
G11
R12
G13
R14
…
B21
G22
…

25.3.14 GVSP_PIX_BAYRG10

[CR25-17s] If supported, GVSP_PIX_BAYRG10 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Image Format	BayerRG	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000000D
Pixel Format:	**GVSP_PIX_BAYRG10**	0x0110000D

Byte 0 Byte1
R11
G12
R13
G14
…
G21
B22
…

25.3.15 GVSP_PIX_BAYGB10

[CR25-18s] If supported, GVSP_PIX_BAYGB10 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Image Format	BayerGB	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000000E
Pixel Format:	**GVSP_PIX_BAYGB10**	0x0110000E

Byte 0	Byte1
G11	
B12	
G13	
B14	
…	
R21	
G22	
…	

25.3.16 GVSP_PIX_BAYBG10

[CR25-19s] If supported, GVSP_PIX_BAYBG10 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Image Format	BayerBG	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000000F
Pixel Format:	**GVSP_PIX_BAYBG10**	0x0110000F

Byte 0 Byte1
B11
G12
B13
G14
…
G21
R22
…

25.3.17 GVSP_PIX_BAYGR12

[CR25-20s] If supported, GVSP_PIX_BAYGR12 MUST follow the layout below.

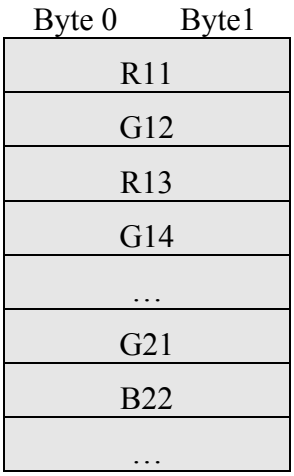
	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Image Format	BayerGR	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000010
Pixel Format:	**GVSP_PIX_BAYGR12**	0x01100010

Byte 0 Byte1
G11
R12
G13
R14
…
B21
G22
…

25.3.18 GVSP_PIX_BAYRG12

[CR25-21s] If supported, GVSP_PIX_BAYRG12 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Image Format	BayerRG	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000011
Pixel Format:	**GVSP_PIX_BAYRG12**	0x01100011



25.3.19 GVSP_PIX_BAYGB12

[CR25-22s] If supported, GVSP_PIX_BAYGB12 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Image Format	BayerGB	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000012
Pixel Format:	**GVSP_PIX_BAYGB12**	0x01100012

Byte 0 Byte1
G11
B12
G13
B14
…
R21
G22
…

25.3.20 GVSP_PIX_BAYBG12

[CR25-23s] If supported, GVSP_PIX_BAYBG12 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Image Format	BayerBG	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000013
Pixel Format:	**GVSP_PIX_BAYBG12**	0x01100013

Byte 0 Byte1
B11
G12
B13
G14
…
G21
R22
…

25.3.21 GVSP_PIX_BAYGR10_PACKED

[CR25-42s] If supported, GVSP_PIX_BAYGR10_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono10Packed	
Image Format	BayerGR	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x00000026
Pixel Format:	**GVSP_PIX_BAYGR10_PACKED**	0x010C0026

The GVSP_PIX_BAYGR10_PACKED pixel format compresses a stream of raw 10-bit Bayer pixels. For a top/left aligned AOI the stream starts with:

Byte 0	Byte1	Byte2
G11/R12		
G13/R14		
…		
B21/G22		
…		

25.3.22 GVSP_PIX_BAYRG10_PACKED

[CR25-43s] If supported, GVSP_PIX_BAYRG10_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono10Packed	
Image Format	BayerRG	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x00000027
Pixel Format:	**GVSP_PIX_BAYRG10_PACKED**	0x010C0027

The GVSP_PIX_BAYRG10_PACKED pixel format compresses a stream of raw 10-bit Bayer pixels. For a top/left aligned AOI the stream starts with:

Byte 0	Byte1	Byte2
R11/G12		
R13/G14		
…		
G21/B22		
…		

25.3.23 GVSP_PIX_BAYGB10_PACKED

[CR25-44s] If supported, GVSP_PIX_BAYGB10_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono10Packed	
Image Format	BayerGB	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x00000028
Pixel Format:	**GVSP_PIX_BAYGB10_PACKED**	0x010C0028

The GVSP_PIX_BAYGB10_PACKED pixel format compresses a stream of raw 10-bit Bayer pixels. For a top/left aligned AOI the stream starts with:

Byte 0	Byte1	Byte2
G11/B12		
G13/B14		
…		
R21/G22		
…		

25.3.24 GVSP_PIX_BAYBG10_PACKED

[CR25-45s] If supported, GVSP_PIX_BAYBG10_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono10Packed	
Image Format	BayerBG	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x00000029
Pixel Format:	**GVSP_PIX_BAYBG10_PACKED**	0x010C0029

The GVSP_PIX_BAYBG10_PACKED pixel format compresses a stream of raw 10-bit Bayer pixels. For a top/left aligned AOI the stream starts with:

Byte 0	Byte1	Byte2
B11/G12		
B13/G14		
…		
G21/R22		
…		

25.3.25 GVSP_PIX_BAYGR12_PACKED

[CR25-46s] If supported, GVSP_PIX_BAYGR12_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono12Packed	
Image Format	BayerGR	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x0000002A
Pixel Format:	**GVSP_PIX_BAYGR12_PACKED**	0x010C002A

The GVSP_PIX_BAYGR12_PACKED pixel format compresses a stream of raw 12-bit Bayer pixels. For a top/left aligned AOI the stream starts with:

Byte 0	Byte1	Byte2
G11/R12		
G13/R14		
…		
B21/G22		
…		

25.3.26 GVSP_PIX_BAYRG12_PACKED

[CR25-47s] If supported, GVSP_PIX_BAYRG12_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono12Packed	
Image Format	BayerRG	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x0000002B
Pixel Format:	**GVSP_PIX_BAYRG12_PACKED**	0x010C002B

The GVSP_PIX_BAYRG12_PACKED pixel format compresses a stream of raw 12-bit Bayer pixels. For a top/left aligned AOI the stream starts with:

Byte 0	Byte1	Byte2
R11/G12		
R13/G14		
…		
G21/B22		
…		

25.3.27 GVSP_PIX_BAYGB12_PACKED

[CR25-48s] If supported, GVSP_PIX_BAYGB12_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono12Packed	
Image Format	BayerGB	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x0000002C
Pixel Format:	**GVSP_PIX_BAYGB12_PACKED**	0x010C002C

The GVSP_PIX_BAYGB12_PACKED pixel format compresses a stream of raw 12-bit Bayer pixels. For a top/left aligned AOI the stream starts with:

Byte 0	Byte1	Byte2
G11/B12		
G13/B14		
…		
R21/G22		
…		

25.3.28 GVSP_PIX_BAYBG12_PACKED

[CR25-49s] If supported, GVSP_PIX_BAYBG12_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono12Packed	
Image Format	BayerBG	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x0000002D
Pixel Format:	**GVSP_PIX_BAYBG12_PACKED**	0x010C002D

The GVSP_PIX_BAYBG12_PACKED pixel format compresses a stream of raw 12-bit Bayer pixels. For a top/left aligned AOI the stream starts with:

Byte 0	Byte1	Byte2
B11/G12		
B13/G14		
…		
G21/R22		
…		

25.3.29 GVSP_PIX_BAYGR16

[CR25-50s] If supported, GVSP_PIX_BAYGR16 MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono16	
Image Format	BayerGR	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000002E
Pixel Format:	**GVSP_PIX_BAYGR16**	0x0110002E

Byte 0	Byte1
G11	
R12	
G13	
R14	
…	
B21	
G22	
…	

25.3.30 GVSP_PIX_BAYRG16

[CR25-51s] If supported, GVSP_PIX_BAYRG16 MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono16	
Image Format	BayerRG	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000002F
Pixel Format:	**GVSP_PIX_BAYRG16**	0x0110002F

Byte 0 Byte1
R11
G12
R13
G14
…
G21
B22
…

25.3.31 GVSP_PIX_BAYGB16

[CR25-52s] If supported, GVSP_PIX_BAYGB16 MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono16	
Image Format	BayerGB	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000030
Pixel Format:	**GVSP_PIX_BAYGB16**	0x01100030

Byte 0	Byte1
G11	
B12	
G13	
B14	
…	
R21	
G22	
…	

25.3.32 GVSP_PIX_BAYBG16

[CR25-53s] If supported, GVSP_PIX_BAYBG16 MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono16	
Image Format	BayerBG	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000031
Pixel Format:	**GVSP_PIX_BAYBG16**	0x01100031

Byte 0 Byte1
B11
G12
B13
G14
…
G21
R22
…

25.3.33 GVSP_PIX_RGB8_PACKED

[CR25-24s] If supported, GVSP_PIX_RGB8_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	24 Bit	0x00180000
Pixel ID	Id of pixel	0x00000014
Pixel Format:	**GVSP_PIX_RGB8_PACKED**	0x02180014

Byte 0
R11
G11
B11
R12
G12
B12
…
R21
G21
B21
…

25.3.34 GVSP_PIX_BGR8_PACKED

[CR25-25s] If supported, GVSP_PIX_BGR8_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	B = G = R = Align_Mono8	
Image Format	BGR	
Color/Mono	COLOR	0x02000000
Bits per pixel	24 Bit	0x00180000
Pixel ID	Id of pixel	0x00000015
Pixel Format:	**GVSP_PIX_BGR8_PACKED**	0x02180015

Byte 0
B11
G11
R11
B12
G12
R12
…
B21
G21
R21
…

25.3.35 GVSP_PIX_RGBA8_PACKED

[CR25-26s] If supported, GVSP_PIX_RGBA8_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = a = Align_Mono8	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	32 Bit	0x00200000
Pixel ID	Id of pixel	0x00000016
Pixel Format:	**GVSP_PIX_RGBA8_PACKED**	0x02200016

Byte 0
R11
G11
B11
α11
R12
G12
B12
α12
…
R21
G21
B21
α21
…

25.3.36 GVSP_PIX_BGRA8_PACKED

[CR25-27s] If supported, GVSP_PIX_BGRA8_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	B = G = R = a = Align_Mono8	
Image Format	BGR	
Color/Mono	COLOR	0x02000000
Bits per pixel	32 Bit	0x00200000
Pixel ID	Id of pixel	0x00000017
Pixel Format:	**GVSP_PIX_BGRA8_PACKED**	0x02200017

Byte 0
B11
G11
R11
α11
B12
G12
R12
α12
…
B21
G21
R21
α21
…

25.3.37 GVSP_PIX_RGB10_PACKED

[CR25-28s] If supported, GVSP_PIX_RGB10_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000018
Pixel Format:	**GVSP_PIX_RGB10_PACKED**	0x02300018

Byte 0 Byte 1
R11
G11
B11
R12
G12
B12
…
R21
G21
B21
…

25.3.38 GVSP_PIX_BGR10_PACKED

[CR25-29s] If supported, GVSP_PIX_BGR10_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	B = G = R = Align_Mono10	
Image Format	BGR	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000019
Pixel Format:	**GVSP_PIX_BGR10_PACKED**	0x02300019

Byte 0 Byte 1
B11
G11
R11
B12
G12
R12
…
B21
G21
R21
…

25.3.39 GVSP_PIX_RGB12_PACKED

[CR25-30s] If supported, GVSP_PIX_RGB12_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x0000001A
Pixel Format:	**GVSP_PIX_RGB12_PACKED**	0x0230001A

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Byte 0 Byte 1
R11
G11
B11
R12
G12
B12
…
R21
G21
B21
…

25.3.40 GVSP_PIX_BGR12_PACKED

[CR25-31s] If supported, GVSP_PIX_BGR12_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	B = G = R = Align_Mono12	
Image Format	BGR	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x0000001B
Pixel Format:	**GVSP_PIX_BGR12_PACKED**	0x0230001B

Byte 0	Byte 1
B11	
G11	
R11	
B12	
G12	
R12	
…	
B21	
G21	
R21	
…	

25.3.41 GVSP_PIX_RGB16_PACKED

[CR25-54s] If supported, GVSP_PIX_RGB16_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono16	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000033
Pixel Format:	GVSP_PIX_RGB16_PACKED	0x02300033

Byte 0 Byte 1
R11
G11
B11
R12
G12
B12
…
R21
G21
B21
…

25.3.42 GVSP_PIX_BGR10V1_PACKED

[CR25-32s] If supported, GVSP_PIX_BGR10V1_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Align_ABC10Packed_V1 B is position A G is position B R is position C	
Image Format	BGR	
Color/Mono	COLOR	0x02000000
Bits per pixel	32 Bit	0x00200000
Pixel ID	Id of pixel	0x0000001C
Pixel Format:	**GVSP_PIX_BGR10V1_PACKED**	0x0220001C

Byte 0 Byte1 Byte2 Byte 3
B11/G11/R11
B12/G12/R12
…
B21/G21/R21
…

25.3.43 GVSP_PIX_BGR10V2_PACKED

[CR25-33s] If supported, GVSP_PIX_BGR10V2_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Align_ABC10Packed_V2 B is position A G is position B R is position C	
Image Format	BGR	
Color/Mono	COLOR	0x02000000
Bits per pixel	32 Bit	0x00200000
Pixel ID	Id of pixel	0x0000001D
Pixel Format:	**GVSP_PIX_BGR10V2_PACKED**	0x0220001D

Byte 0	Byte1	Byte2	Byte 3
B11/G11/R11			
B12/G12/R12			
…			
B21/G21/R21			
…			

25.3.44 GVSP_PIX_RGB12V1_PACKED

[CR25-55s] If supported, GVSP_PIX_RGB12V1_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R=G=B = Align_Mono12Packed	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	36 Bit	0x00240000
Pixel ID	Id of pixel	0x00000034
Pixel Format:	GVSP_PIX_RGB12V1_PACKED	0x02240034

The GVSP_PIX_RGB12V1_PACKED pixel format compresses a stream of 12-bit RGB pixels.

Byte 0	Byte1	Byte2
R11/G11		
B11/R12		
G12/B12		
…		

25.3.45 GVSP_PIX_RGB565_PACKED

[CR25-57s] If supported, GVSP_PIX_RGB565_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Align_ABC565Packed R is position A G is position B B is position C	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000035
Pixel Format:	**GVSP_PIX_RGB565_PACKED**	0x02100035

Byte 0	Byte 1
RGB11	
RGB12	
…	
RGB21	
…	

25.3.46 GVSP_PIX_BGR565_PACKED

[CR25-58s] If supported, GVSP_PIX_BGR565_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Align_ABC565Packed B is position A G is position B R is position C	
Image Format	BGR	
Color/Mono	COLOR	0x02000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000036
Pixel Format:	**GVSP_PIX_BGR565_PACKED**	0x02100036

Byte 0	Byte 1
BGR11	
BGR12	
…	
BGR21	
…	

25.3.47 GVSP_PIX_YUV411_PACKED

[CR25-34s] If supported, GVSP_PIX_YUV411_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = U = V = Align_Mono8	
Image Format	YUV411	
Color/Mono	COLOR	0x02000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x0000001E
Pixel Format:	**GVSP_PIX_YUV411_PACKED**	0x020C001E

Byte 0
U11
Y11
Y12
V11
Y13
Y14
U15
Y15
Y16
V15
…
U21
Y21
Y22
V21
…

25.3.48 GVSP_PIX_YUV422_PACKED

[CR25-35s] If supported, GVSP_PIX_YUV422_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = U = V = Align_Mono8	
Image Format	YUV422	
Color/Mono	COLOR	0x02000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000001F
Pixel Format:	**GVSP_PIX_YUV422_PACKED**	0x0210001F

Byte 0
U11
Y11
V11
Y12
U13
Y13
V13
Y14
…
U21
Y21
V21
Y22
…

25.3.49 GVSP_PIX_YUV422_YUYV_PACKED

[CR25-56s] If supported, GVSP_PIX_YUV422_YUYV_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y=U=V= Align_Mono8	
Image Format	YUV422	
Color/Mono	COLOR	0x02000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000032
Pixel Format:	**GVSP_PIX_YUV422_YUYV_PACKED**	0x02100032

Byte 0
Y11
U11
Y12
V11
Y13
U13
Y14
V13
…
Y21
U21
Y22
V21
…

25.3.50 GVSP_PIX_YUV444_PACKED

[CR25-36s] If supported, GVSP_PIX_YUV444_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = U = V = Align_Mono8	
Image Format	YUV444	
Color/Mono	COLOR	0x02000000
Bits per pixel	24 Bit	0x00180000
Pixel ID	Id of pixel	0x00000020
Pixel Format:	**GVSP_PIX_YUV444_PACKED**	0x02180020

Byte 0
U11
Y11
V11
U12
Y12
V12
…
U21
Y21
V21
…

25.3.51 GVSP_PIX_RGB8_PLANAR

[CR25-37s] If supported, GVSP_PIX_RGB8_PLANAR MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	24 Bit	0x00180000
Pixel ID	Id of pixel	0x00000021
Pixel Format:	**GVSP_PIX_RGB8_PLANAR**	0x02180021

Red plane (stream channel 0):

Byte 0
R11
R12
R13
R14
…
R21
…

Green plane (stream channel 1):

Byte 0
G11
G12
G13
G14
…
G21
…

Blue plane (stream channel 2):

Byte 0
B11
B12
B13
B14
…
B21
…

25.3.52 GVSP_PIX_RGB10_PLANAR

[CR25-38s] If supported, GVSP_PIX_RGB10_PLANAR MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000022
Pixel Format:	**GVSP_PIX_RGB10_PLANAR**	0x02300022

Red plane (stream channel 0):

Byte 0 Byte1
R11
R12
R13
R14
…
R21
…

Green plane (stream channel 1):

Byte 0 Byte1
G11
G12
G13
G14
…
G21
…

Blue plane (stream channel 2):

Byte 0 Byte1
B11
B12
B13
B14
…
B21
…

25.3.53 GVSP_PIX_RGB12_PLANAR

[CR25-39s] If supported, GVSP_PIX_RGB12_PLANAR MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000023
Pixel Format:	**GVSP_PIX_RGB12_PLANAR**	0x02300023

Red plane (stream channel 0):

Byte 0 Byte1
R11
R12
R13
R14
…
R21
…

Green plane (stream channel 1):

Byte 0 Byte1
G11
G12
G13
G14
…
G21
…

Blue plane (stream channel 2):

Byte 0 Byte1
B11
B12
B13
B14
…
B21
…

25.3.54 GVSP_PIX_RGB16_PLANAR

[CR25-40s] If supported, GVSP_PIX_RGB16_PLANAR MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono16	
Image Format	RGB	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000024
Pixel Format:	**GVSP_PIX_RGB16_PLANAR**	0x02300024

Red plane (stream channel 0):

Byte 0 Byte1
R11
R12
R13
R14
…
R21
…

Green plane (stream channel 1):

Byte 0 Byte1
G11
G12
G13
G14
…
G21
…

Blue plane (stream channel 2):

Byte 0 Byte1
B11
B12
B13
B14
…
B21
…

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

26 Pixel Format Defines

The following provides #define for the various pixel formats supported by GVSP. Each pixel format is represented by a 32-bit value. The upper 8-bit indicates the color. The second upper 8-bit indicates the number of bit occupied by a pixel (including any padding). This can be used to quickly compute the amount of memory required to store an image using this pixel format.

```
//===================================================
// PIXEL FORMATS
//===================================================
// Indicate if pixel is monochrome or RGB
#define GVSP_PIX_MONO                           0x01000000
#define GVSP_PIX_RGB                            0x02000000 // deprecated in version 1.1
#define GVSP_PIX_COLOR                          0x02000000
#define GVSP_PIX_CUSTOM                         0x80000000
#define GVSP_PIX_COLOR_MASK                     0xFF000000

// Indicate effective number of bits occupied by the pixel (including padding).
// This can be used to compute amount of memory required to store an image.
#define GVSP_PIX_OCCUPY8BIT                     0x00080000
#define GVSP_PIX_OCCUPY12BIT                    0x000C0000
#define GVSP_PIX_OCCUPY16BIT                    0x00100000
#define GVSP_PIX_OCCUPY24BIT                    0x00180000
#define GVSP_PIX_OCCUPY32BIT                    0x00200000
#define GVSP_PIX_OCCUPY36BIT                    0x00240000
#define GVSP_PIX_OCCUPY48BIT                    0x00300000
#define GVSP_PIX_EFFECTIVE_PIXEL_SIZE_MASK      0x00FF0000
#define GVSP_PIX_EFFECTIVE_PIXEL_SIZE_SHIFT     16

// Pixel ID: lower 16-bit of the pixel formats
#define GVSP_PIX_ID_MASK                        0x0000FFFF
```

26.1 Mono buffer format defines

```
#define GVSP_PIX_MONO8              (GVSP_PIX_MONO | GVSP_PIX_OCCUPY8BIT   | 0x0001)
#define GVSP_PIX_MONO8_SIGNED       (GVSP_PIX_MONO | GVSP_PIX_OCCUPY8BIT   | 0x0002)
#define GVSP_PIX_MONO10             (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x0003)
#define GVSP_PIX_MONO10_PACKED      (GVSP_PIX_MONO | GVSP_PIX_OCCUPY12BIT  | 0x0004)
#define GVSP_PIX_MONO12             (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x0005)
#define GVSP_PIX_MONO12_PACKED      (GVSP_PIX_MONO | GVSP_PIX_OCCUPY12BIT  | 0x0006)
#define GVSP_PIX_MONO14             (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x0025)
#define GVSP_PIX_MONO16             (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x0007)
```

26.2 Bayer buffer format defines

```
#define GVSP_PIX_BAYGR8             (GVSP_PIX_MONO | GVSP_PIX_OCCUPY8BIT   | 0x0008)
#define GVSP_PIX_BAYRG8             (GVSP_PIX_MONO | GVSP_PIX_OCCUPY8BIT   | 0x0009)
#define GVSP_PIX_BAYGB8             (GVSP_PIX_MONO | GVSP_PIX_OCCUPY8BIT   | 0x000A)
#define GVSP_PIX_BAYBG8             (GVSP_PIX_MONO | GVSP_PIX_OCCUPY8BIT   | 0x000B)
#define GVSP_PIX_BAYGR10            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x000C)
#define GVSP_PIX_BAYRG10            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x000D)
#define GVSP_PIX_BAYGB10            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x000E)
```

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

```
#define GVSP_PIX_BAYBG10            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x000F)
#define GVSP_PIX_BAYGR12            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x0010)
#define GVSP_PIX_BAYRG12            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x0011)
#define GVSP_PIX_BAYGB12            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x0012)
#define GVSP_PIX_BAYBG12            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x0013)
#define GVSP_PIX_BAYGR10_PACKED     (GVSP_PIX_MONO | GVSP_PIX_OCCUPY12BIT  | 0x0026)
#define GVSP_PIX_BAYRG10_PACKED     (GVSP_PIX_MONO | GVSP_PIX_OCCUPY12BIT  | 0x0027)
#define GVSP_PIX_BAYGB10_PACKED     (GVSP_PIX_MONO | GVSP_PIX_OCCUPY12BIT  | 0x0028)
#define GVSP_PIX_BAYBG10_PACKED     (GVSP_PIX_MONO | GVSP_PIX_OCCUPY12BIT  | 0x0029)
#define GVSP_PIX_BAYGR12_PACKED     (GVSP_PIX_MONO | GVSP_PIX_OCCUPY12BIT  | 0x002A)
#define GVSP_PIX_BAYRG12_PACKED     (GVSP_PIX_MONO | GVSP_PIX_OCCUPY12BIT  | 0x002B)
#define GVSP_PIX_BAYGB12_PACKED     (GVSP_PIX_MONO | GVSP_PIX_OCCUPY12BIT  | 0x002C)
#define GVSP_PIX_BAYBG12_PACKED     (GVSP_PIX_MONO | GVSP_PIX_OCCUPY12BIT  | 0x002D)
#define GVSP_PIX_BAYGR16            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x002E)
#define GVSP_PIX_BAYRG16            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x002F)
#define GVSP_PIX_BAYGB16            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x0030)
#define GVSP_PIX_BAYBG16            (GVSP_PIX_MONO | GVSP_PIX_OCCUPY16BIT  | 0x0031)
```

26.3 RGB Packed buffer format defines

```
#define GVSP_PIX_RGB8_PACKED        (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY24BIT | 0x0014)
#define GVSP_PIX_BGR8_PACKED        (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY24BIT | 0x0015)
#define GVSP_PIX_RGBA8_PACKED       (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY32BIT | 0x0016)
#define GVSP_PIX_BGRA8_PACKED       (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY32BIT | 0x0017)
#define GVSP_PIX_RGB10_PACKED       (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY48BIT | 0x0018)
#define GVSP_PIX_BGR10_PACKED       (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY48BIT | 0x0019)
#define GVSP_PIX_RGB12_PACKED       (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY48BIT | 0x001A)
#define GVSP_PIX_BGR12_PACKED       (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY48BIT | 0x001B)
#define GVSP_PIX_RGB16_PACKED       (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY48BIT | 0x0033)
#define GVSP_PIX_RGB10V1_PACKED     (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY32BIT | 0x001C)
#define GVSP_PIX_RGB10V2_PACKED     (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY32BIT | 0x001D)
#define GVSP_PIX_RGB12V1_PACKED     (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY36BIT | 0X0034)
#define GVSP_PIX_RGB565_PACKED      (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY16BIT | 0x0035)
#define GVSP_PIX_BGR565_PACKED      (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY16BIT | 0X0036)
```

26.4 YUV Packed buffer format defines

```
#define GVSP_PIX_YUV411_PACKED      (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY12BIT | 0x001E)
#define GVSP_PIX_YUV422_PACKED      (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY16BIT | 0x001F)
#define GVSP_PIX_YUV422_YUYV_PACKED (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY16BIT | 0x0032)
#define GVSP_PIX_YUV444_PACKED      (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY24BIT | 0x0020)
```

26.5 RGB Planar buffer format defines

```
#define GVSP_PIX_RGB8_PLANAR        (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY24BIT | 0x0021)
#define GVSP_PIX_RGB10_PLANAR       (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY48BIT | 0x0022)
#define GVSP_PIX_RGB12_PLANAR       (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY48BIT | 0x0023)
#define GVSP_PIX_RGB16_PLANAR       (GVSP_PIX_COLOR | GVSP_PIX_OCCUPY48BIT | 0x0024)
```

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

PART 4 – Bootstrap Registers

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

27 Bootstrap Registers

[R27-1cd] All mandatory bootstrap registers MUST be present on all GigE Vision compliant devices (though some specific bootstrap registers are optional when indicated). These bootstrap registers are listed in Table 27-1.

[O27-37cd] Bootstrap registers SHOULD be provided in the XML device configuration file and respect the rules defined in GenICam Standard Features Naming Convention for GigE Vision.

Having the bootstrap registers in the XML file provides wider possibility for value validation since XML features may have information about their minimum, maximum and increment.

Device registers are accessed using GVCP. Device-specific registers can start after bootstrap registers memory-space.

[R27-2cd] All device registers MUST be 32-bit aligned.

[O27-3ca] Bootstrap registers SHOULD be accessed by the application using READREG and WRITEREG messages, with the exception of character strings which should be accessed using READMEM.

This is to avoid any endianess issue that could arise when accessing multi-byte data with READMEM.

[R27-4ca] When information spans multiple 32-bit registers, then the register with the lowest address MUST be accessed first, followed by the other registers sequentially until the register with the highest address is accessed.

The above requirement would facilitate migration to wider memory architecture than the current 32-bit (such as 64-bit).

[R27-5cd] All strings stored in the bootstrap registers space MUST match the character set specified by register “Device Mode” at address 0x0004 and be NULL-terminated.

Some registers might not be available on a given device. They are highlighted by the “optional” designation. In this case, the application must refer to the device capabilities indicating the number of message channels, stream channels and network interfaces. Also, optional GVCP commands are indicated by a bitfield register where each bit represents a different optional command.

[R27-6cd] All unused bits of bootstrap registers MUST be set to 0.

[O27-7cd] Trying to access an unsupported register SHOULD return GEV_STATUS_INVALID_ADDRESS.

[O27-8cd] Trying to read from a write-only register SHOULD return GEV_STATUS_ACCESS_DENIED. Trying to write to a read only register SHOULD return GEV_STATUS_WRITE_PROTECT.

[O27-9ca] An application should use the READMEM message to retrieve strings from the bootstrap registers.

Note: A device internal representation of each 32-bit register might be either big or little endian. This must be indicated in the Device Mode register. But GVCP messages must have the data in big-endian (network byte order). A little-endian device is free to perform byte swapping operation to convert its internal register to big-endian when it receives a message.

The following table lists the bootstrap registers. It indicates if a register is (M)andatory or (O)ptional. It also indicates the type of access (R)ead and (W)rite that can be performed.

Table 27-1: Bootstrap Registers

Address	Name	Support	Type	Length (bytes)	Description
0x0000	Version	M	R	4	Version of the GigE Standard with which the device is compliant The two most-significant bytes are allocated to the major number of the version, the two least-significant bytes for the minor number.
0x0004	Device Mode	M	R	4	Information about device mode of operation.
0x0008	Device MAC address – High (Network interface #0)	M	R	4	The two most-significant bytes of this area are reserved and will return 0. Upper 2 bytes of the MAC address are stored in the lower 2 significant bytes of this register.
0x000C	Device MAC address – Low (Network interface #0)	M	R	4	Lower 4 bytes of the MAC address
0x0010	Supported IP configuration (Network interface #0)	M	R	4	Bits can be OR-ed. All other bits are reserved and set to 0. DHCP and LLA bits must be on.
0x0014	Current IP configuration procedure (Network interface #0)	M	R/W	4	Bits can be OR-ed. LLA is always activated and is read-only.
0x0018	Reserved	-	-	-	
0x0024	Current IP address (Network interface #0)	M	R	4	Current IP address of this device.
0x0028	Reserved	-	-	-	
0x0034	Current subnet mask (Network interface #0)	M	R	4	Current subnet mask used by this device.
0x0038	Reserved	-	-	-	
0x0044	Current default Gateway (Network interface #0)	M	R	4	Current default gateway used by this device.
0x0048	Manufacturer name	M	R	32	Provides the name of the manufacturer of the device. String using the format in “string character set”. The string must be 0 terminated, and all bytes after the terminator must be 0.

Address	Name	Support	Type	Length (bytes)	Description
0x0068	Model name	M	R	32	Provides the model name of the device. String using the format in "string character set". The string must be 0 terminated, and all bytes after the terminator must be 0.
0x0088	Device version	M	R	32	Provides the version of the device. String using the format in "string character set". The string must be 0 terminated, and all bytes after the terminator must be 0.
0x00A8	Manufacturer specific information	M	R	48	Provides extended manufacturer information about the device. String using the format in "string character set". The string must be 0 terminated.
0x00D8	Serial number	O	R	16	When supported, this string contains the serial number of the device. It can be used to identify the device. This string is provided in the DISCOVERY_ACK message (set to all NULL when not supported). String using the format in "string character set". The string must be 0 terminated.
0x00E8	User-defined name	O	R/W	16	When supported, this string contains a user-programmable name to assign to the device. It can be used to identify the device. This string is provided in the DISCOVERY_ACK message (set to all NULL when not supported). String using the format in "string character set". The string must be 0 terminated.
0x00F8	Reserved	-	-	-	
0x0200	First choice of URL for XML device description file	M	R	512	String using the format in "string character set". The string must be NULL-terminated, and all bytes after the terminator must be 0. 3 different schemes are supported to link to the XML device description. This string indicates XML file location. 1) XML stores in non-volatile on-board memory. In this case, the device provides the address and the length where the XML file is stored. 2) URL to device manufacturer. If file extension is .XML, then file is an uncompressed text file. If file extension is .ZIP, then file is compressed using ZIP. 3) Filename of the XML device description file. This assumes the application uses a 'default' folder where XML files are stored. If file extension is .XML, then file is an uncompressed text file. If file extension is .ZIP, then file is compressed using ZIP.
0x0400	Second choice of URL for XML device description file	M	R	512	String using the format in "string character set". The string must be NULL-terminated, and all bytes after the terminator must be 0.

Address	Name	Support	Type	Length (bytes)	Description
0x0600	Number of network interfaces	M	R	4	Indicates the number of physical network interfaces on this device. A device must have at least one network interface.
0x0604	Reserved	-	-	-	
0x064C	Persistent IP address (Network interface #0)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x0650	Reserved	-	-	-	
0x065C	Persistent subnet mask (Network interface #0)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x0660	Reserved	-	-	-	
0x066C	Persistent default gateway (Network interface #0)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x0670	Link Speed (Network interface #0)	O	R	4	32-bit value indicating current Ethernet link speed in Mbits per second.
0x0674	Reserved	-	-	-	
0x0680	MAC address – High (Network interface #1)	O	R	4	The two most-significant bytes of this area are reserved and will return 0. Upper 2 bytes of the MAC address are stored in the lower 2 significant bytes of this register.
0x0684	MAC address – Low (Network interface #1)	O	R	4	Lower 4 bytes of the MAC address
0x0688	Supported IP configuration (Network interface #1)	O	R	4	Bits can be OR-ed. All other bits are reserved and set to 0. DHCP and LLA bits must be on.
0x068C	Current IP configuration procedure (Network interface #1)	O	R/W	4	Bits can be OR-ed. LLA is always activated and is read-only.
0x0690	Reserved	-	-	-	
0x069C	Current IP address (Network interface #1)	O	R	4	Current IP address of this device.
0x06A0	Reserved	-	-	-	
0x06AC	Current subnet mask (Network interface #1)	O	R	4	Current subnet mask used by this device.
0x06B0	Reserved	-	-	-	
0x06BC	Current default gateway (Network interface #1)	O	R	4	Current default gateway used by this device.
0x06C0	Reserved	-	-	-	
0x06CC	Persistent IP address (Network interface #1)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x06D0	Reserved	-	-	-	
0x06DC	Persistent subnet mask (Network interface #1)	O	R/W	4	Only available if Persistent IP is supported by the device.

Address	Name	Support	Type	Length (bytes)	Description
0x06E0	Reserved	-	-	-	
0x06EC	Persistent default gateway (Network interface #1)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x06F0	Link Speed (Network interface #1)	O	R	4	32-bit value indicating current Ethernet link speed in Mbits per second.
0x06F4	Reserved	-	-	-	
0x0700	MAC address – High (Network interface #2)	O	R	4	The two most-significant bytes of this area are reserved and will return 0. Upper 2 bytes of the MAC address are stored in the lower 2 significant bytes of this register.
0x0704	MAC address – Low (Network interface #2)	O	R	4	Lower 4 bytes of the MAC address
0x0708	Supported IP configuration (Network interface #2)	O	R	4	Bits can be OR-ed. All other bits are reserved and set to 0. DHCP and LLA bits must be on.
0x070C	Current IP configuration procedure (Network interface #2)	O	R/W	4	Bits can be OR-ed. LLA is always activated and is read-only.
0x0710	Reserved	-	-	-	
0x071C	Current IP address (Network interface #2)	O	R	4	Current IP address of this device.
0x0720	Reserved	-	-	-	
0x072C	Current subnet mask (Network interface #2)	O	R	4	Current subnet mask used by this device.
0x0730	Reserved	-	-	-	
0x073C	Current default gateway (Network interface #2)	O	R	4	Current default gateway used by this device.
0x0740	Reserved	-	-	-	
0x074C	Persistent IP address (Network interface #2)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x0750	Reserved	-	-	-	
0x075C	Persistent subnet mask (Network interface #2)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x0760	Reserved	-	-	-	
0x076C	Persistent default gateway (Network interface #2)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x0770	Link Speed (Network interface #2)	O	R	4	32-bit value indicating current Ethernet link speed in Mbits per second.
0x0774	Reserved	-	-	-	
0x0780	MAC address – High (Network interface #3)	O	R	4	The two most-significant bytes of this area are reserved and will return 0. Upper 2 bytes of the MAC address are stored in the lower 2 significant bytes of this register.

Address	Name	Support	Type	Length (bytes)	Description
0x0784	MAC address – Low (Network interface #3)	O	R	4	Lower 4 bytes of the MAC address
0x0788	Supported IP configuration (Network interface #3)	O	R	4	Bits can be OR-ed. All other bits are reserved and set to 0. DHCP and LLA bits must be on.
0x078C	Current IP configuration procedure (Network interface #3)	O	R/W	4	Bits can be OR-ed. LLA is always activated and is read-only.
0x0790	Reserved	-	-	-	
0x079C	Current IP address (Network interface #3)	O	R	4	Current IP address of this device.
0x07A0	Reserved	-	-	-	
0x07AC	Current subnet mask (Network interface #3)	O	R	4	Current subnet mask used by this device.
0x07B0	Reserved	-	-	-	
0x07BC	Current default gateway (Network interface #3)	O	R	4	Current default gateway used by this device.
0x07C0	Reserved	-	-	-	
0x07CC	Persistent IP address (Network interface #3)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x07D0	Reserved	-	-	-	
0x07DC	Persistent subnet mask (Network interface #3)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x07E0	Reserved	-	-	-	
0x07EC	Persistent default gateway (Network interface #3)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x07F0	Link Speed (Network interface #3)	O	R	4	32-bit value indicating current Ethernet link speed in Mbits per second.
0x07F4	Reserved	-	-	-	
0x0900	Number of Message channels	M	R	4	Indicates the number of message channels supported by this device. It can take two values: 0 or 1.
0x0904	Number of Stream channels	M	R	4	Indicates the number of stream channels supported by this device. It can take any value from 0 to 512.
0x0908	Number of Action Signals	O	R	4	Inidicates the number of separate action signals supported by this device. It can take any value ranging from 0 to 128.
0x090C	Action Device Key	O	W	4	Device key to check the validity of action commands.
0x0910	Reserved	-	-	-	
0x092C	Stream channels Capability	O	R	4	Indicates the capabilities of the stream channels. It lists which of the non-mandatory stream channel features are supported.

Address	Name	Support	Type	Length (bytes)	Description
0x0930	Message channel Capability	O	R	4	Indicates the capabilities of the message channel. It lists which of the non-mandatory message channel features are supported.
0x0934	GVCP Capability	M	R	4	This is a capability register indicating which one of the non-mandatory GVCP features are supported by this device.
0x0938	Heartbeat timeout	M	R/W	4	In msec, default is 3000 msec. Internally, the heartbeat is rounded according to the clock used for heartbeat. The heartbeat timeout shall have a minimum precision of 100 ms. The minimal value is 500 ms.
0x093C	Timestamp tick frequency – High	O	R	4	64-bit value indicating the number of timestamp clock tick in 1 second. This register holds the most significant bytes. Timestamp tick frequency is 0 if timestamp is not supported.
0x0940	Timestamp tick frequency – Low	O	R	4	64-bit value indicating the number of timestamp clock tick in 1 second. This register holds the least significant bytes. Timestamp tick frequency is 0 if timestamp is not supported.
0x0944	Timestamp control	O	W	4	Used to latch the current timestamp value. No need to clear to 0.
0x0948	Timestamp value (latched) – High	O	R	4	Latched value of the timestamp (most significant bytes)
0x094C	Timestamp value (latched) – Low	O	R	4	Latched value of the timestamp (least significant bytes)
0x0950	Discovery ACK delay	O	R/(W)	4	Randomized delay in ms to acknowledge a DISCOVERY_CMD.
0x0954	GVCP Configuration	O	R/W	4	Configuration of GVCP optional features
0x0958	Pending Timeout	O	R	4	Pending Timeout to report the longest GVCP command execution time before issuing a PENDING_ACK.
0x095C	Control Switchover Key	O	W	4	Key to authenticate primary application switchover requests.
0x0960	Reserved	-	-	-	
0x0A00	CCP	M	R/W	4	Control Channel Privilege register.
0x0A04	Primary Application Port	O	R	4	UDP source port of the control channel of the primary application.
0x0A08	Reserved	-	-	-	
0x0A14	Primary Application IP address	O	R	4	Source IP address of the control channel of the primary application.
0x0A18	Reserved	-	-	-	
0x0B00	MCP	O	R/W	4	Message Channel Port register.
0x0B04	Reserved	-	-	-	
0x0B10	MCDA	O	R/W	4	Message Channel Destination Address register.
0x0B14	MCTT	O	R/W	4	Message Channel Transmission Timeout in ms
0x0B18	MCRC	O	R/W	4	Message Channel Retry Count

Address	Name	Support	Type	Length (bytes)	Description
0x0B1C	MCSP	O	R	4	Message Channel Source Port
0x0B20	Reserved	-	-	-	
0x0D00	SCP0	M[2]	R/W	4	First Stream Channel Port register.
0x0D04	SCPS0	M[2]	R/W	4	First Stream Channel Packet Size register.
0x0D08	SCPD0	M[3]	R/W	4	First Stream Channel Packet Delay register.
0x0D0C	Reserved	-	-	-	
0x0D18	SCDA0	M[2]	R/W	4	First Stream Channel Destination Address register.
0x0D1C	SCSP0	O	R	4	First Stream Channel Source Port register
0x0D20	SCC0	O	R	4	First Stream Channel Capability register
0x0D24	SCCFG0	O	R	4	First Stream Channel Configuration register
0x0D28	Reserved	-	-	-	
0x0D40	SCP1	O	R/W	4	Second stream channel, if supported.
0x0D44	SCPS1	O	R/W	4	Second stream channel, if supported.
0x0D48	SCPD1	O	R/W	4	Second stream channel, if supported.
0x0D4C	Reserved	-	-	-	
0x0D58	SCDA1	O	R/W	4	Second stream channel, if supported.
0x0D5C	SCSP1	O	R	4	Second stream channel, if supported.
0x0D60	SCC1	O	R	4	Second stream channel, if supported
0x0D64	SCCFG1	O	R	4	Second stream channel, if supported
0x0D68	Reserved	-	-	-	
0x0D80	*Other stream channels registers*	O	R/W	-	Each stream channel is allocated a section of 64 bytes (0x40). Only supported channels are available.
0x8CC0	SCP511	O	R/W	4	512th stream channel, if supported.
0x8CC4	SCPS511	O	R/W	4	512th stream channel, if supported.
0x8CC8	SCPD511	O	R/W	4	512th stream channel, if supported.
0x8CCC	Reserved	-	-	-	
0x8CD8	SCDA511	O	R/W	4	512th stream channel, if supported.
0x8CDC	SCSP511	O	R	4	512th stream channel, if supported.
0x8CE0	SCC511	O	R	4	512th stream channel, if supported
0x8CE4	SCCFG511	O	R	4	512th stream channel, if supported
0x8CE8	Reserved	-	-	-	
0x9000	Manifest Table	O	R	512	Manifest Table providing information to retrieve XML documents stored in the device.
0x9200	Reserved	-	-	-	

[2] Mandatory only for transmitter, receiver and transceiver devices. Not applicable for peripherals.
[3] Mandatory only for transmitter devices. Not applicable for others.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Address	Name	Support	Type	Length (bytes)	Description
0x9800	ACTION_GROUP_KEY0	O	R/W	4	First action signal group key
0x9804	ACTION_GROUP_MASK0	O	R/W	4	First action signal group mask
0x9808	Reserved	-	-	-	
0x9810	ACTION_GROUP_KEY1	O	R/W	4	Second action signal group key
0x9814	ACTION_GROUP_MASK1	O	R/W	4	Second action signal group mask
0x9818	Reserved	-	-	-	
0x9820	*Other action signals registers*	O	R/W	-	Each action signal is allocated a section of 16 bytes (0x10). Only supported action signals are available.
0x9FF0	ACTION_GROUP_KEY127	O	R/W	4	128th action signal group key
0x9FF4	ACTION_GROUP_MASK127	O	R/W	4	128th action signal group mask
0x9FF8	Reserved	-	-	-	
0xA000	Start of manufacturer-specific register space	-	-	-	Start of device-specific registers. These are not covered by the specification.

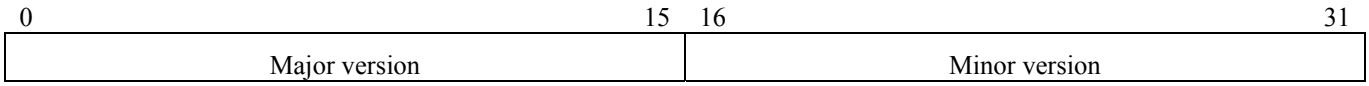
GEV V1.2 000000

27.1 Version Register

This register indicates the version of the GigE Vision specification implemented by this device. Version 1.2 of this specification shall return 0x00010002.

The version register can be used by the application to validate the device is compliant with the specified version of the GigE Vision specification.

Address	0x0000
Length	4 bytes
Access type	Read
Factory Default	0x00010002

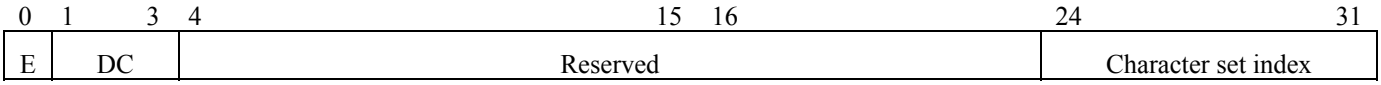


Bits	Name	Description
0 – 15	Major version	This field represents the major version of the specification. For instance, GigE Vision version **1**.2 would have the major version set to 1.
16 – 31	Minor version	This field represents the minor version of the specification. For instance, GigE Vision version 1.**2** would have the minor version set to 2.

27.2 Device Mode Register

This register indicates the character set used by the various strings present in the bootstrap registers and other device-specific information, such as the endianess of multi-byte data and the device class.

Address	0x0004
Length	4 bytes
Access type	Read
Factory Default	Device-specific



Bits	Name	Description
0	Endianess	Endianess might be used to interpret multi-byte data for READMEM and WRITEMEM commands. This represents the endianess of all device's registers (bootstrap registers and manufacturer-specific registers). 0: Little-endian device 1: Big-endian device Note this bit has no effect on the endianess of the GigE Vision protocol headers: they are always big-endian.
1 – 3	Device Class	This field represents the class of the device. It must take one of the following values 0: Transmitter 1: Receiver 2: Transceiver 3: Peripheral other: reserved
4 – 23	Reserved	Always 0
24 – 31	Character set index	This register represents the character set. It must take one of the following values 0: reserved 1: UTF-8 other: reserved

27.3 Device MAC Registers

These registers store the MAC address of the given network interface.

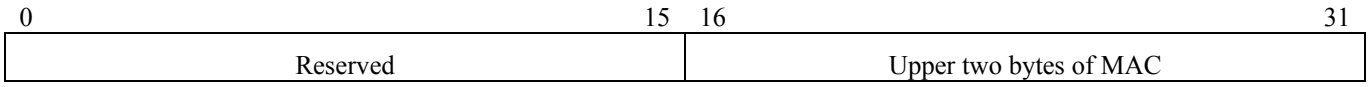
These registers are mandatory on all supported interfaces.

[R27-10cd] The device MUST support Interface #0 for the Device MAC register. Device must return an error for unsupported interfaces.

An application needs to access the high part before accessing the low part.

27.3.1 High Part

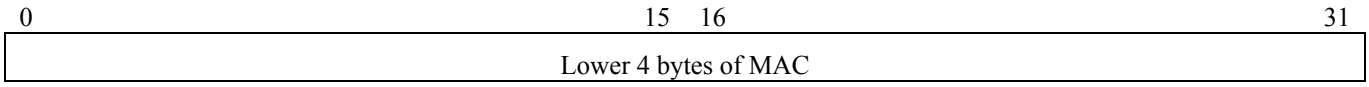
Address	0x0008 (interface #0) 0x0680 (interface #1) *[optional]* 0x0700 (interface #2) *[optional]* 0x0780 (interface #3) *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



Bits	Name	Description
0 – 15	Reserved	Always 0
16 – 31	Upper two bytes of MAC	Hold the upper two bytes of the MAC address

27.3.2 Low Part

Address	0x000C (interface #0) 0x0684 (interface #1) *[optional]* 0x0704 (interface #2) *[optional]* 0x0784 (interface #3) *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



Bits	Name	Description
0 – 31	Lower 4 bytes of MAC	Hold the lower four bytes of the MAC address

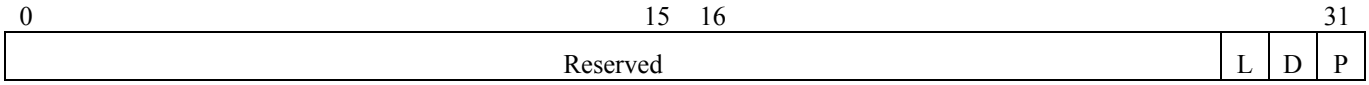
27.4 Supported IP Configuration Registers

These registers indicate the IP configuration scheme supported on the given network interface. Multiple schemes can be supported simultaneously.

These registers are mandatory on all supported interfaces.

[R27-11cd] The device MUST support Interface #0 for the Supported IP Configuration register. Device must return an error for unsupported interfaces.

Address	0x0010 (interface #0) 0x0688 (interface #1) *[optional]* 0x0708 (interface #2) *[optional]* 0x0788 (interface #3) *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



Bits	Name	Description
0 – 28	Reserved	Always 0
29	LLA	Link-local address is supported. Always 1
30	DHCP	DHCP is supported. Always 1
31	Persistent IP	1 if Persistent IP is supported, 0 otherwise.

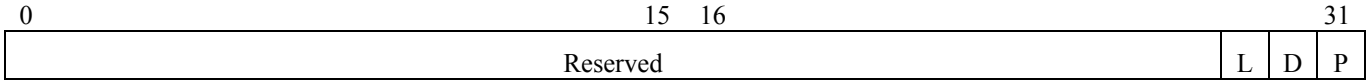
27.5 Current IP Configuration Registers

These registers indicate which IP configurations schemes are currently activated on the given network interface.

This register is mandatory on all supported interfaces.

[R27-12cd] The device MUST support Interface #0 for the Current IP Configuration register. Device must return an error for unsupported interfaces.

Address	0x0014 (interface #0) 0x068C (interface #1) *[optional]* 0x070C (interface #2) *[optional]* 0x078C (interface #3) *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0x0000006



Bits	Name	Description
0 – 28	Reserved	Always 0.
29	LLA	Link-local address is activated. Always 1.
30	DHCP	DHCP is activated on this interface.
31	Persistent IP	Persistent IP is activated on this interface.

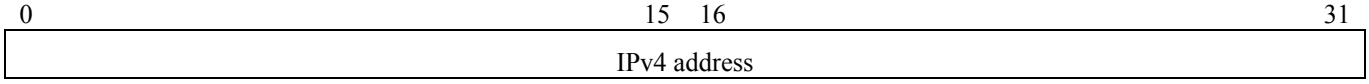
27.6 Current IP Address Registers

These registers report the IP address for the given network interface once it has been configured.

These registers are mandatory on all supported interfaces.

[R27-13cd] The device MUST support Interface #0 for the Current IP Address register. Device must return an error for unsupported interfaces.

Address	0x0024 (interface #0) 0x069C (interface #1) *[optional]* 0x071C (interface #2) *[optional]* 0x079C (interface #3) *[optional]*
Length	4 bytes
Access type	Read
Factory Default	0x00000000



Bits	Name	Description
0 – 31	IPv4 address	IP address for this network interface

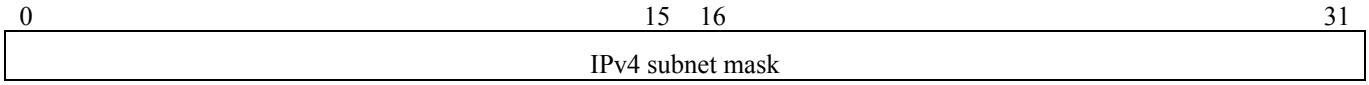
27.7 Current Subnet Mask Registers

These registers provide the subnet mask of the given interface.

These registers are mandatory on all supported interfaces.

[R27-14cd] The device MUST support Interface #0 for the Current Subnet Mask register. Device must return an error for unsupported interfaces.

Address	0x0034 (interface #0) 0x06AC (interface #1) *[optional]* 0x072C (interface #2) *[optional]* 0x07AC (interface #3) *[optional]*
Length	4 bytes
Access type	Read
Factory Default	0x00000000



Bits	Name	Description
0 – 31	IPv4 subnet mask	Subnet mask for this network interface

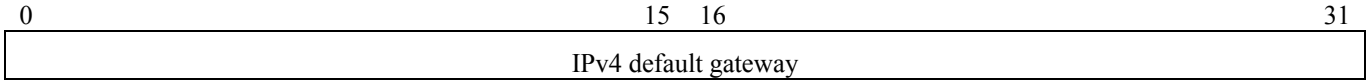
27.8 Current Default Gateway Registers

These registers indicate the default gateway IP address to be used on the given network interface.

These registers are mandatory on all supported interfaces.

[R27-15cd] The device MUST support Interface #0 for the Current Default Gateway register. Device must return an error for unsupported interfaces.

Address	0x0044 (interface #0) 0x06BC (interface #1) *[optional]* 0x073C (interface #2) *[optional]* 0x07BC (interface #3) *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



Bits	Name	Description
0 – 31	IPv4 default gateway	Default gateway for this network interface

27.9 Manufacturer Name Register

This register stores a string containing the manufacturer name. This string uses the character set indicated in the "Device Mode" register.

Address	0x0048
Length	32 bytes
Access type	Read
Factory Default	Device-specific

Bits	Name	Description
-	Manufacturer	NULL-terminated string indicating the manufacturer name.

27.10 Model Name Register

This register stores a string containing the device model name. This string uses the character set indicated in the "Device Mode" register.

Address	0x0068
Length	32 bytes
Access type	Read
Factory Default	Device-specific

Bits	Name	Description
-	Model	NULL-terminated string indicating the device model.

27.11 Device Version Register

This register stores a string containing the version of the device. This string uses the character set indicated in the "Device Mode" register. The XML device description file should also provide this information to ensure the device matches the description file.

Address	0x0088
Length	32 bytes
Access type	Read
Factory Default	Device-specific

Bits	Name	Description
-	Version	NULL-terminated string indicating the version of the device.

27.12 Manufacturer Info Register

This register stores a string containing additional manufacturer-specific information about the device. This string uses the character set indicated in the "Device Mode" register.

Address	0x00A8
Length	48 bytes
Access type	Read
Factory Default	Device-specific

Bits	Name	Description
-	Info	NULL-terminated string providing additional information about this device.

27.13 Serial Number Register

This register is optional. It can store a string containing the serial number of the device. This string uses the character set indicated in the "Device Mode" register.

An application can check bit 1 of the "GVCP Capability register" at address 0x0934 to check if the serial number register is supported by the device.

[CO27-16cd] When a device does not support the Serial Number register, READMEM SHOULD return GEV_STATUS_INVALID_ADDRESS with 'length' field of the header set to 0.

Address	0x00D8 *[optional]*
Length	16 bytes
Access type	Read
Factory Default	Device-specific

Bits	Name	Description
-	serial number	NULL-terminated string providing the serial number of this device.

27.14 User-defined Name Register

This register is optional. It can store a user-programmable string providing the name of the device. This string uses the character set indicated in the "Device Mode" register.

An application can check bit 0 of the "GVCP Capability register" at address 0x0934 to check if the user-defined name register is supported by the device.

[CO27-17cd] When a device does not support the User-defined Name register, READMEM and WRITEMEM SHOULD return GEV_STATUS_INVALID_ADDRESS. In this case, for READMEM, the 'length' field of the header is set to 0; for WRITEMEM, the 'index' field is set to 0.

[CR27-18cd] When this register is supported, the device MUST provide persistent storage to memorize the user-defined name. This name shall remain readable across power-up cycles.

Address	0x00E8 *[optional]*
Length	16 bytes
Access type	Read/Write
Factory Default	Device-specific

Bits	Name	Description
-	user-defined name	NULL-terminated string providing the device name.

27.15 First URL Register

This register stores the first URL to the XML device description file. The First URL is used as the first choice by the application to retrieve the XML device description file.

[R27-19cd] The First URL string MUST use the character set indicated in the "Device Mode" register.

Address	0x0200
Length	512 bytes
Access type	Read
Factory Default	Device-specific

Bits	Name	Description
-	First URL	NULL-terminated string providing the first URL to the XML device description file.

27.16 Second URL Register

This register stores the second URL to the XML device description file. This URL is an alternative if the application was unsuccessful to retrieve the device description file using the first URL.

[R27-20cd] The Second URL string uses the character set indicated in the "Device Mode" register.

Address	0x0400
Length	512 bytes
Access type	Read
Factory Default	Device-specific

Bits	Name	Description
-	Second URL	NULL-terminated string providing the second URL to the XML device description file.

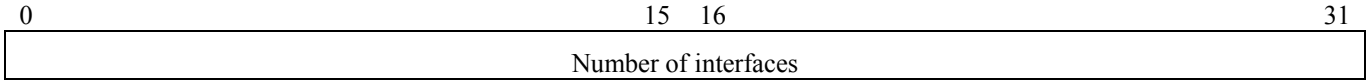
27.17 Number of Network Interfaces Register

This register indicates the number of physical network interfaces supported by this device.

A device need to support at least one network interfaces (the primary interface). A device can support at most four network interfaces.

Note that interface #0 is the only one supporting GVCP. Additional network interfaces only supports stream channels in order to increase available bandwidth out of the device.

Address	0x0600
Length	4 bytes
Access type	Read
Factory Default	Device-specific



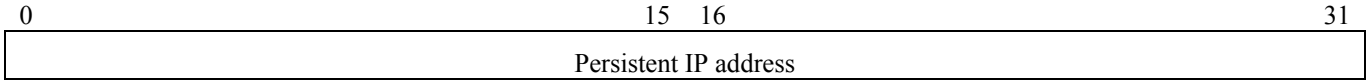
Bits	Name	Description
0 – 31	Number of interfaces	A value from 1 to 4. All other values are invalid.

27.18 Persistent IP Address Registers

These optional registers indicate the Persistent IP address for the given network interface. They are only used when the device boots with the Persistent IP configuration scheme.

[CR27-21cd] When Persistent IP is supported, the device MUST support Persistent IP Address register on all supported interfaces. Device must return an error for unsupported interfaces.

Address	0x064C (interface #0) *[optional]* 0x06CC (interface #1) *[optional]* 0x074C (interface #2) *[optional]* 0x07CC (interface #3) *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



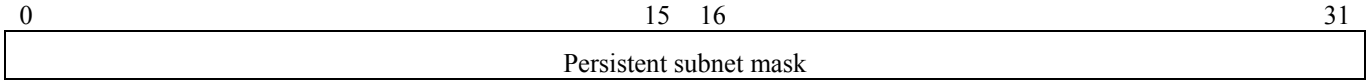
Bits	Name	Description
0 – 31	Persistent IP address	IPv4 persistent IP address for this network interface

27.19 Persistent Subnet Mask Registers

These optional registers indicate the Persistent subnet mask associated with the Persistent IP address on the given network interface. They are only used when the device boots with the Persistent IP configuration scheme.

[CR27-22cd] When Persistent IP is supported, the device MUST support Persistent Subnet Mask register on all supported interfaces. Device must return an error for unsupported interfaces.

Address	0x065C (interface #0) *[optional]* 0x06DC (interface #1) *[optional]* 0x075C (interface #2) *[optional]* 0x07DC (interface #3) *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



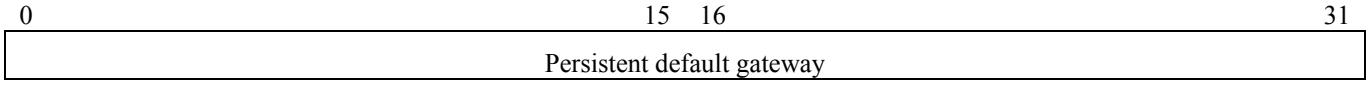
Bits	Name	Description
0 – 31	Persistent subnet mask	IPv4 persistent subnet mask for this network interface

27.20 Persistent Default Gateway Registers

These optional registers indicate the persistent default gateway for the given network interface. They are only used when the device boots with the Persistent IP configuration scheme.

[CR27-23cd] When Persistent IP is supported, the device MUST support Persistent Default Gateway register on all supported interfaces. Device must return an error for unsupported interfaces.

Address	0x066C (interface #0) *[optional]* 0x06EC (interface #1) *[optional]* 0x076C (interface #2) *[optional]* 0x07EC (interface #3) *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



Bits	Name	Description
0 – 31	Persistent default gateway	IPv4 persistent default gateway for this network interface

27.21 Link Speed Registers

These optional registers provide the Ethernet link speed (in Mbits per second) for the given network interface. This can be used to compute the transmission speed.

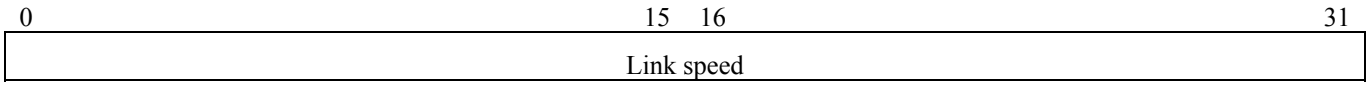
[CR27-38cd] When Link Speed registers are supported, the device MUST support a Link Speed register on all supported interfaces. Device must return an error for unsupported interfaces.

For instance, if a device only implements a single network interface, then it shall only support one link speed registers assigned to network interface #0. A link speed of 1 gigabit per second is equal to 1000 Mbits per second.

A capability bit in the GVCP capability register (bit 3 at address 0x0934) indicates if these registers are supported.

Theses registers return an error to the application if they are not supported and the application tries to access them, according to [O27-7cd].

Address	0x0670 (interface #0) *[optional]* 0x06F0 (interface #1) *[optional]* 0x0770 (interface #2) *[optional]* 0x07F0 (interface #3) *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



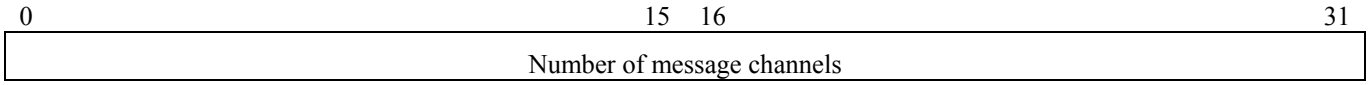
Bits	Name	Description
0 – 31	Link speed	Ethernet link speed value in Mbps. 0 if Link is down.

27.22 Number of Message Channels Register

This register reports the number of message channels supported by this device.

A device may support at most 1 message channel, but it is allowed to support no message channel.

Address	0x0900
Length	4 bytes
Access type	Read
Factory Default	Device-specific



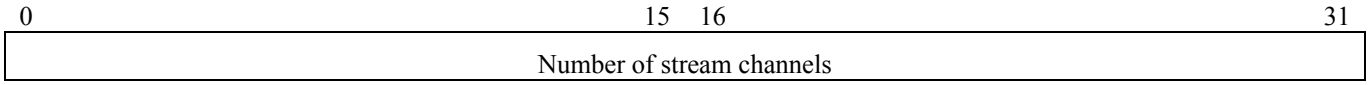
Bits	Name	Description
0 – 31	Nb Message channels	Number of message channels supported by this device. Can be 0 or 1.

27.23 Number of Stream Channels Register

This register reports the number of stream channels supported by this device.

A product can support from 0 up to 512 stream channels.

Address	0x0904
Length	4 bytes
Access type	Read
Factory Default	Device-specific



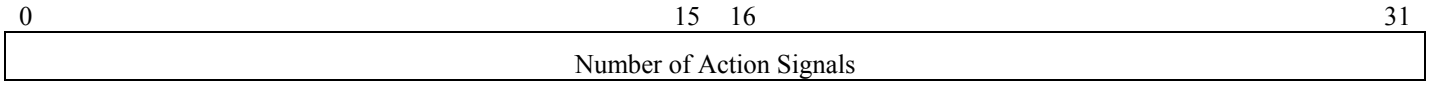
Bits	Name	Description
0 – 31	Nb Stream channels	Number of stream channels supported by this device. A value from 0 to 512.

27.24 Number of Action Signals Register

This optional register reports the number of action signal supported by this device.

This register returns an error to the application if it is not supported and the application tries to access it, according to [O27-7cd].

Address	0x0908 *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



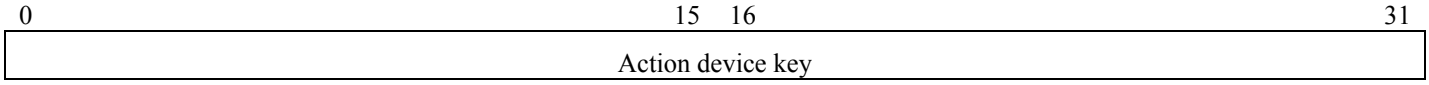
Bits	Name	Description
0 – 31	Nb Action Signals	Number of action signals supported by this device. A value from 0 to 128.

27.25 Action Device Key Register

This optional register provides the device key to check the validity of action commands. The action device key is **write-only** to hide the key if CCP is not in exclusive access mode. The intention is for the Primary Application to have absolute control. The Primary Application can send the key to a Secondary Application. This mechanism prevents a rogue application to have access to the ACTION_CMD mechanism.

This register returns an error to the application if it is not supported and the application tries to access it, according to [O27-7cd].

Address	0x090C *[optional]*
Length	4 bytes
Access type	Write-only
Factory Default	0



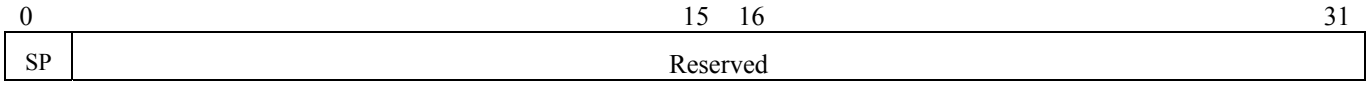
Bits	Name	Description
0-31	Action device key	Device key to check the validity of action commands

27.26 Stream Channels Capability Register

This optional register reports the optional stream channel features supported by this device. This register might not be supported by a device: in this case the device should respect [O27-7cd].

[CR27-39cd] If the device supports any of the optional features listed in the Stream Channel Capability register, then it MUST implement the Stream Channel Capability register.

Address	0x092C *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



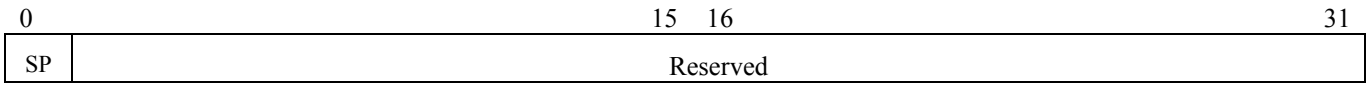
Bits	Name	Description
0	SCSPx supported (SP)	Indicates the SCSPx registers (stream channel source port) are available for all supported stream channels.

27.27 Message Channel Capability Register

This optional register reports the optional message channel features supported by this device. This register might not be supported by a device: in this case the device should respect [O27-7cd].

[CR27-40cd] If the device supports any of the optional features listed in the Message Channel Capability register, then it MUST implement the Message Channel Capability register.

Address	0x0930 *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



Bits	Name	Description
0	MCSP supported	Indicates the MCSP register (message channel source port) is available for the message channel.

27.28 GVCP Capability Register

This register reports the optional GVCP commands and bootstrap registers supported by this device on its Control Channel. When supported, some of these features are enabled through the GVCP Configuration register (at address 0x0954).

Address	0x0934
Length	4 bytes
Access type	Read
Factory Default	Device-specific

0	1	2	3	4	5	6	7	8	9	10		25	26	27	28	29	30	31
UN	SN	HD	LS	CAP	MT	TD	DD	WD	ES	PAS	Reserved	A	P	ED	E	PR	W	C

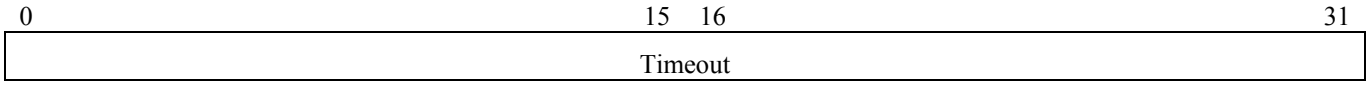
Bits	Name	Description
0	User-defined Name (UN)	User-defined name register is supported.
1	Serial Number (SN)	Serial number register is supported.
2	Heartbeat Disable (HD)	Heartbeat can be disabled.
3	Link Speed Register (LS)	Link Speed registers are supported.
4	CCP Application Port/IP Register (CAP)	CCP Application Port and IP address registers are supported.
5	Manifest Table (MT)	Manifest Table is supported. When supported, the application must use the Manifest Table to retrieve the XML device description file.
6	Test Data (TD)	Test packet is filled with data from the LFSR generator.
7	Discovery ACK Delay (DD)	Discovery ACK Delay register is supported.
8	Writable Discovery ACK Delay (WD)	When Discovery ACK Delay register is supported, this bit indicates that the application can write it. If this bit is 0, the register is read-only.
9	Extended Status Code for 1.1 (ES)	Support generation of extended status codes introduced in specification 1.1: GEV_STATUS_PACKET_RESEND, GEV_STATUS_WRONG_CONFIG, GEV_STATUS_PACKET_NOT_YET_AVAILABLE, GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY and GEV_STATUS_PACKET_REMOVED_FROM_MEMORY.
10	Primary Application Switchover (PAS)	Primary application switchover capability is supported.
11 – 24	Reserved	Always 0.
25	ACTION (A)	ACTION_CMD and ACTION_ACK are supported.
26	PENDING (P)	PENDING_ACK is supported.
27	EVENTDATA (ED)	EVENTDATA_CMD and EVENTDATA_ACK are supported.
28	EVENT (E)	EVENT_CMD and EVENT_ACK are supported.
29	PACKETRESEND (PR)	PACKETRESEND_CMD is supported.
30	WRITEMEM (W)	WRITEMEM_CMD and WRITEMEM_ACK are supported.
31	Concatenation (C)	Multiple operations in a single message are supported.

27.29 Heartbeat Timeout Register

This register indicates the current heartbeat timeout in milliseconds.

[O27-24cd] A value smaller than 500 ms SHOULD be defaulted to 500 ms by the device. In this case, the heartbeat timeout register content is changed to reflect the actual value used by the device.

Address	0x0938
Length	4 bytes
Access type	Read/Write
Factory Default	3000 = 0x0BB8



Bits	Name	Description
0 – 31	Timeout	Heartbeat timeout in milliseconds (minimum is 500 ms)

Note: Upon changing the heartbeat timeout register, it might take up to the amount of time previously specified in this register for the new value to take effect.

27.30 Timestamp Tick Frequency Registers

These optional registers indicate the number of timestamp tick during 1 second. This corresponds to the timestamp frequency in Hertz. For example, a 100 MHz clock would have a value of 100 000 000 = 0x05F5E100. They are combined to form a 64-bit value.

Note the timestamp counter is also used to compute stream channel inter-packet delay.

No timestamp is supported if both of these registers are equal to 0.

[O27-25cd] A device SHOULD support the "Timestamp Tick Frequency" registers. If its value is set to 0 or if this register is not available, then no timestamp counter is present. This means no mechanism is offered to control inter-packet delay.

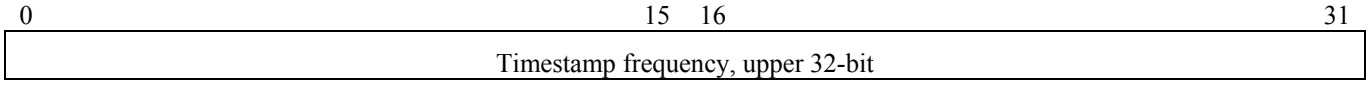
These registers return an error to the application if they are not supported and the application tries to access them, according to [O27-7cd].

An application needs to access the high part before accessing the low part, as indicated by [R27-4ca]

27.30.1 High Part

Address	0x093C *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific

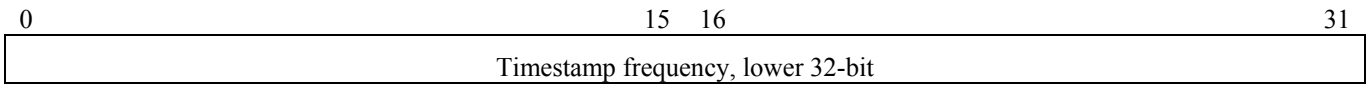
Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724



Bits	Name	Description
0 – 31	Freq - High	Timestamp frequency, upper 32-bit

27.30.2 Low Part

Address	0x0940 *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



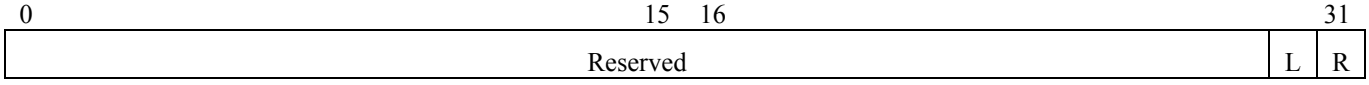
Bits	Name	Description
0 – 31	Freq – Low	Timestamp frequency, lower 32-bit

27.31 Timestamp Control Register

This optional register is used to control the timestamp counter. This register returns an error to the application if it is not supported and the application tries to access it, according to [O27-7cd].

[CR27-26ca] If a timestamp counter exists, an application MUST not attempt to read this register. This register is write-only.

Address	0x0944 *[optional]*
Length	4 bytes
Access type	Write-only
Factory Default	0



Bits	Name	Description
0 – 29	Reserved	Always 0
30	Latch	Latch current timestamp counter into "Timestamp value" register at address 0x0948.
31	Reset	Reset timestamp 64-bit counter to 0.

[CR27-27cd] If a timestamp counter exists, when the application set both the Latch and Reset bit in the same access, then the device MUST first latch the timestamp before resetting it.

Note: Writing a 1 into a bit causes the requested operation to execute. There is no need to write a 0 into the register afterwards.

27.32 Timestamp Value Registers

These optional registers report the latched value of the timestamp counter. It is necessary to latch the 64-bit timestamp value to guaranty its integrity when performing the two 32-bit read accesses to retrieve the higher and lower 32-bit portions.

[CR27-28ca] If an application wants to retrieve the 64-bit value of the timestamp counter and the timestamp counter is supported by the device, it MUST write 1 into bit 30 of Timestamp control register for the free-running timestamp counter to be copied into these registers.

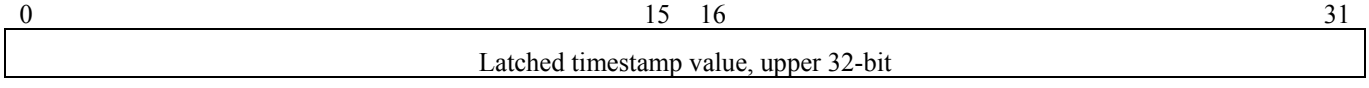
[CR27-29cd] The timestamp value MUST always be 0 if timestamp is not supported.

These registers return an error to the application if they are not supported and the application tries to access them, according to [O27-7cd].

An application needs to access the high part before accessing the low part, as indicated by [R27-4ca].

27.32.1 High Part

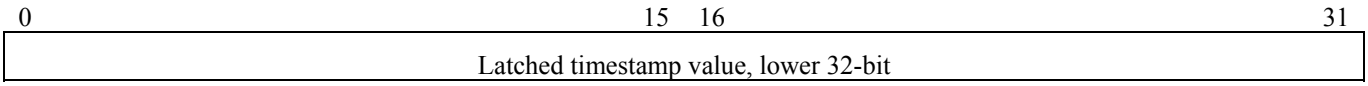
Address	0x0948 *[optional]*
Length	4 bytes
Access type	Read
Factory Default	0



Bits	Name	Description
0 – 31	Time – High	Latched timestamp value, upper 32-bit

27.32.2 Low Part

Address	0x094C *[optional]*
Length	4 bytes
Access type	Read
Factory Default	0



Bits	Name	Description
0 – 31	Time – Low	Latched timestamp value, lower 32-bit

27.33 Discovery ACK Delay Register

This optional register indicates the maximum randomized delay in milliseconds the device will wait upon reception of a DISCOVERY_CMD before it will send back the DISCOVERY_ACK. This randomized delay can take any value from 0 second up to the value indicated by this bootstrap register.

[CR27-41cd] If Discovery ACK Delay register is writable, then its value MUST be persistent and stored in non-volatile memory to be re-used on next device reset or power cycle.

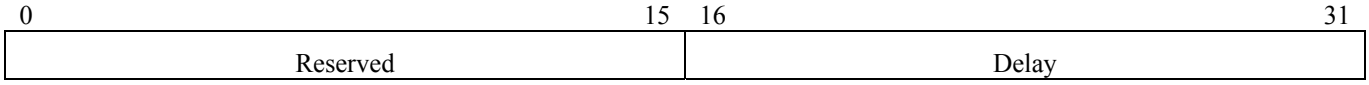
[CR27-42cd] If Discovery ACK Delay is supported, the maximum factory default value for this register MUST be lower or equal to 1000 ms (1 second).

The previous requirement is necessary to ensure backward compatibility with version 1.0 of the specification.

This register can be read-only if the device only supports a fixed value. Furthermore a device is allowed to set this register has read-only with a value of 0 if it does not support such a randomized delay. Bit 8 of the GVCP Capability register (at address 0x0934) indicates if this register is writable.

An application should retrieve information from the XML device description file to see if this register is writable and to determine the maximum value it can take.

Address	0x0950
Length	4 bytes
Access type	Read/(Write)
Factory Default	Device-specific (≤1 second)



Bits	Name	Description
0 – 15	reserved	Always 0
16 – 31	Delay	Maximum random delay in ms to wait before sending DISCOVERY_ACK upon reception of DISCOVERY_CMD.

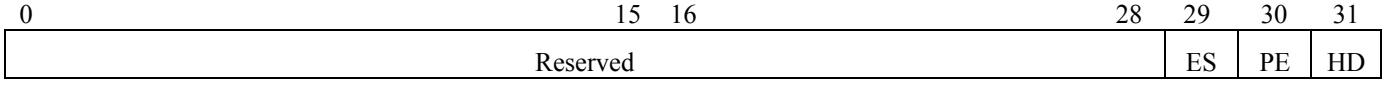
27.34 GVCP Configuration Register

This optional register provides additional control over GVCP. These additional functions must be indicated by the GVCP Capability register (at address 0x0934).

For instance, it can be used to disable Heartbeat when this capability is supported. This can be useful for debugging purposes.

This register returns an error to the application if it is not supported and the application tries to access it, according to [O27-7cd].

Address	0x0954 *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



Bits	Name	Description
0 – 28	reserved	Always 0
29	Extended Status Code for 1.1 (ES)	Enable generation of extended status codes introduced in specification 1.1: GEV_STATUS_PACKET_RESEND, GEV_STATUS_WRONG_CONFIG, GEV_STATUS_PACKET_NOT_YET_AVAILABLE, GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY and GEV_STATUS_PACKET_REMOVED_FROM_MEMORY.
30	PENDING_ACK Enable (PE)	Enable generation of PENDING_ACK by this device. Only available when the PENDING capability bit is set.
31	Heartbeat Disable (HD)	Disable heartbeat. Only available when Heartbeat Disable capability bit is set.

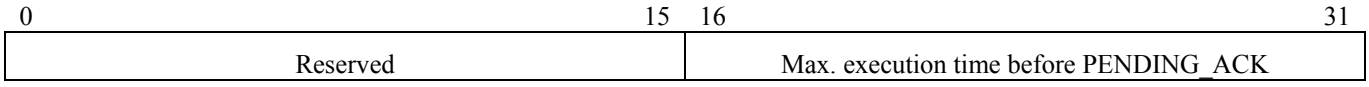
27.35 Pending Timeout Register

This optional register indicates the longest GVCP command execution time before the device returns a PENDING_ACK. The Application can use this value to deduce a suitable ACK timeout to wait for when it issues the various GVCP commands.

This register returns an error to the application if it is not supported and the application tries to access it, according to [O27-7cd].

Note the PENDING_ACK cannot be used for DISCOVERY_CMD, FORCEIP_CMD and PACKETRESEND_CMD.

Address	0x0958 *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



Bits	Name	Description
0 – 15	reserved	Must be 0.
16 – 31	Max. execution time	Provide the maximum execution time (in ms) before the device will issue a PENDING_ACK to notify the application to extend the ACK timeout for the current GVCP command.

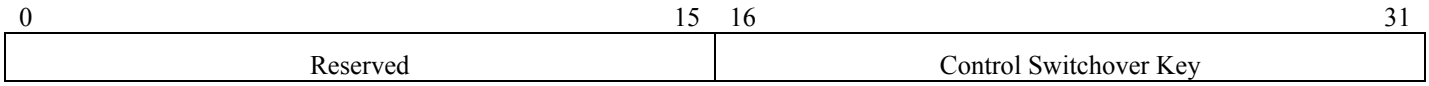
27.36 Control Switchover Key Register

This optional register provides the key to authenticate primary application switchover requests. The register is **write-only** to hide the key from secondary applications. The primary intent is to have a mechanism to

control who can take control over a device. The primary application or an higher level system management entity can send the key to an application that would request control over a device.

This register returns an error to the application if it is not supported and the application tries to access it, according to [O27-7cd].

Address	0x090C *[optional]*
Length	4 bytes
Access type	Write-only
Factory Default	0



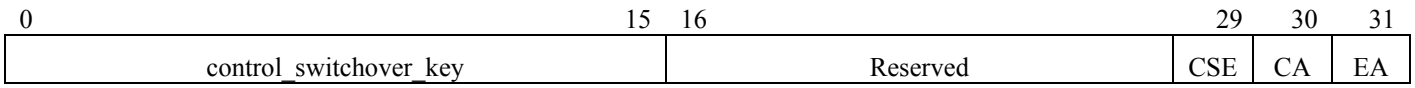
Bits	Name	Description
0 – 15	Reserved	Must be 0.
16 – 31	Control Switchover Key	Device key to check the validity of action commands

27.37 Control Channel Privilege Register (CCP)

This register is used to grant privilege to an application. Only one application is allowed to control the device. This application is able to write into device's registers. Other applications can read device's register only if the controlling application does not have the exclusive privilege.

[R27-30ca] The primary application MUST write 0 into CCP to release its privilege.

Address	0x0A00
Length	4 bytes
Access type	Read/Write
Factory Default	0



Bits	Name	Description
0 – 15	control_switchover_key	This field is applicable only when the device supports the primary application switchover capability. It is taken into account only when the control_switchover_en bit was set prior to the current transaction writing to the CCP register and the transactions comes from a different application. In this case, the primary application switchover will occur only if the value written to this field matches the value in the Control Switchover Key bootstrap register. No matter if the primary application switchover capability is supported or not, this field always reads as zero.
16 – 28	reserved	Always 0.
29	control_switchover_en (CSE)	This bit is applicable only when the device supports the primary application switchover capability. The application writing a 1 to this bit enables another application to request exclusive or control access to the device (without or with switchover enabled). This bit is used in conjunction with the control_access bit. It is "don't care" when the exclusive_access bit is set. This bit always reads as zero when the primary application switchover capability is not supported.
30	control_access (CA)	The application writing a 1 to this bit requests control access to the device (without or with switchover enabled depending of the value written to the control_switchover_en bit). If a device grants control access, no other application can control the device if the control_switchover_en bit is set to zero. Otherwise, another application can request to take over exclusive or control access of the device (without or with switchover enabled). Other applications are still able to monitor the device.
31	exclusive_access (EA)	The application writing a 1 to this bit requests exclusive access to the device. If a device grants exclusive access, no other application can control or monitor the device. Exclusive_access has priority over control_access when both bits are set.

Since exclusive access is more restrictive than control access (without or with switchover enabled), the device must be in exclusive access mode if the exclusive_access bit is set independently of the state of the control_access and control_switchover_en bits. Control access is defined according to Table 27-2.

Table 27-2: Control Access Definition

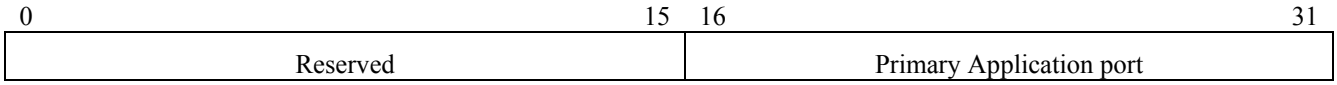
control_switchover_en	control_access	exclusive_access	Access Mode
x	0	0	Open access.
0	1	0	Control access.
1	1	0	Control access with switchover enabled.
x	x	1	Exclusive access.

27.38 Primary Application Port Register

This optional register provides UDP port information about the primary application holding the control channel privilege.

This register returns an error to the application if CCP owner registers are not supported and the application tries to access it, according to [O27-7cd].

Address	0x0A04 *[optional]*
Length	4 bytes
Access type	Read
Factory Default	0



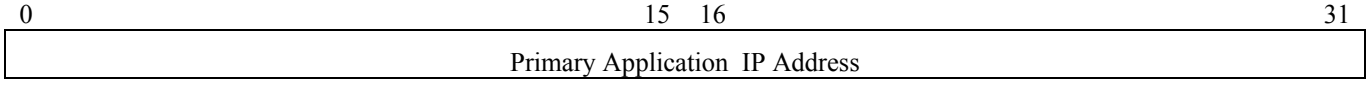
Bits	Name	Description
0 – 15	reserved	Must be 0.
16 – 31	Primary Application host port	The UDP source port of the primary application. This value must be 0 when no primary application is bound to the device (CCP register equal to 0).

27.39 Primary Application IP Address Register

This optional register provides IP address information about the primary application holding the control channel privilege.

This register returns an error to the application if CCP owner registers are not supported and the application tries to access it, according to [O27-7cd].

Address	0x0A14 *[optional]*
Length	4 bytes
Access type	Read
Factory Default	0



Bits	Name	Description
0 – 31	Primary Application IP Address	The IPv4 address of the primary application. This must be a unicast address. This value must be 0 when no primary application is bound to the device (CCP register equal to 0).

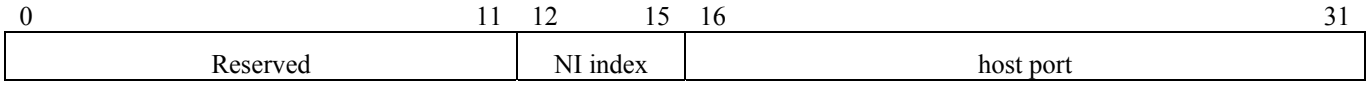
27.40 Message Channel Port Register (MCP)

This optional register provides port information about the message channel.

[CR27-31ca] When supported, an application MUST activate the message channel by writing the host_port field to a value different from 0. Otherwise, the channel is closed.

This register returns an error to the application if message channel is not supported and the application tries to access it, according to [O27-7cd].

Address	0x0B00 *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



Bits	Name	Description
0 – 11	Reserved	Must be 0
12 – 15	network interface index	Always 0 in this version. Only the primary interface (#0) supports GVCP.
16 – 31	host_port	The port to which the device must send messages. Setting this value to 0 closes the message channel.

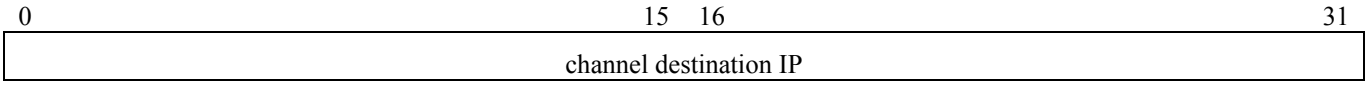
27.41 Message Channel Destination Address Register (MCDA)

This optional register indicates the destination IP address for the message channel.

This register returns an error to the application if message channel is not supported and the application tries to access it, according to [O27-7cd].

Address	0x0B10 *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724



Bits	Name	Description
0 – 31	Channel destination IP	Message channel destination IPv4 address. The destination address can be a multicast or a unicast address.

27.42 Message Channel Transmission Timeout Register (MCTT)

This optional register provides the transmission timeout value in milliseconds.

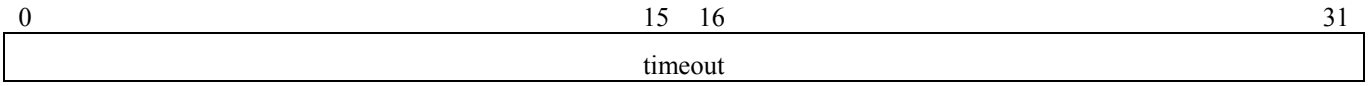
This register returns an error to the application if it is not supported and the application tries to access it, according to [O27-7cd].

[CR27-32cd] This register indicates the amount of time the device MUST wait for acknowledge after a message is sent on the message channel before timeout when an acknowledge is requested when a message channel is supported.

[CR27-33cd] When a message channel is supported, if the timeout expires, then the device MUST try to resend the same message. The number of retries is indicated by MCRC register. When MCTT is 0, then no acknowledge is requested (ACKNOWLEDGE bit of the command header is cleared).

[CR27-34cd] When a message channel is supported and when MCTT is different from 0, then the ACKNOWLEDGE bit of the command header MUST be set.

Address	0x0B14 *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	Device-specific



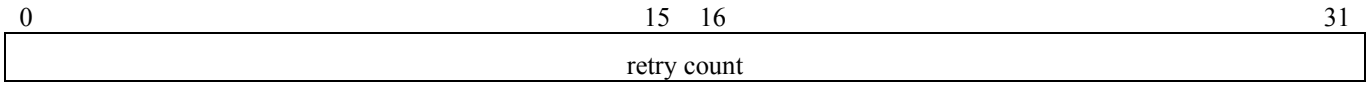
Bits	Name	Description
0 – 31	Timeout	Transmission timeout value in ms.

27.43 Message Channel Retry Count Register (MCRC)

This optional register indicates the number of retransmissions allowed when a message channel message times out.

[CR27-35cd] When a message channel is supported and when MCRC is set to 0, then the device MUST not retransmit a message.

Address	0x0B18 *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	Device-specific



Bits	Name	Description
0 – 31	Retry count	Number of retransmissions allowed on the message channel.

27.44 Message Channel Source Port Register (MCSP)

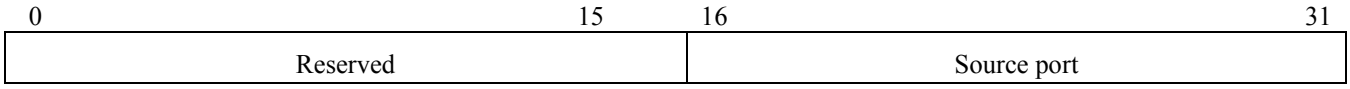
This optional register indicates the source UDP port for the message channel. The purpose of this information is to allow the host application to take measures to ensure asynchronous event traffic from the device is allowed back to the host. The expected usage of this is to send dummy UDP packets from the application's message channel reception port to that device's source port, thus making the message channel traffic appear to be requested by the application instead of unsolicited.

[CR27-43cd] If supported by the device, MCSP MUST return a non-zero value corresponding to the source UDP port of the message channel when MCPS is non-zero. It is valid for it to be non-zero at any point in time prior as well as long as it then remains constant for the duration of the control session.

[CR27-44cd] If supported by the device, when MCSP is non-zero the device MUST silently ignore any UDP traffic coming from the address and port combination listed in MCDA and MCP (if message channel has been opened) targeted at the device MCSP port if they don't match an EVENT_ACK or EVENTDATA_ACK.

The MCSPx register returns an error to the application if the message channel is not supported and the application tries to access it, according to [O27-7cd].

Address	0x0B1C *[optional]*
Length	4 bytes
Access type	Read-Only
Factory Default	0 if unavailable



Bits	Name	Description
0 - 15	Reserved	Must be 0
16 - 31	Source Port	Indicates the UDP source port message channel traffic will be generated from while the message channel is open.

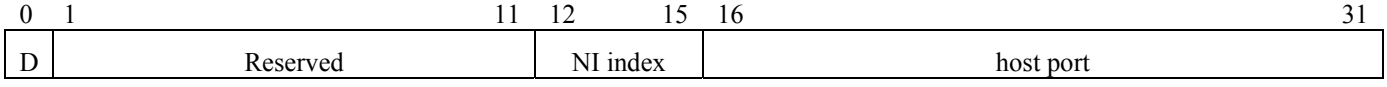
27.45 Stream Channel Port Registers (SCPx)

These registers provide port information for the given stream channel.

[R27-36ca] The stream channel MUST be activated by the application by writing the host_port field with a value different from 0. Otherwise, the channel is closed.

The SCPx register returns an error to the application if the corresponding stream channel is not supported and the application tries to access it, according to [O27-7cd].

Address	0x0D00 + 0x40 * x with 0 ≤ x < 512. *[optional for 0 < x < 512 for GVSP transmitters and receivers]* *[Not applicable for peripherals]*
Length	4 bytes
Access type	Read/Write
Factory Default	Device-specific



Bits	Name	Description
0	Direction	This bit is read-only and reports the direction of the stream channel. It must take one of the following values 0: Transmitter 1: Receiver
1 – 11	Reserved	Must be 0
12 – 15	network interface index	Index of network interface to use (from 0 to 3). Specific streams might be hard-coded to a specific network interfaces. Therefore this field might not be programmable on certain devices. It is read-only for this case.
16 - 31	host_port	The port to which a GVSP transmitter must send data stream. The port from which a GVSP receiver may receive data stream. Setting this value to 0 closes the stream channel.

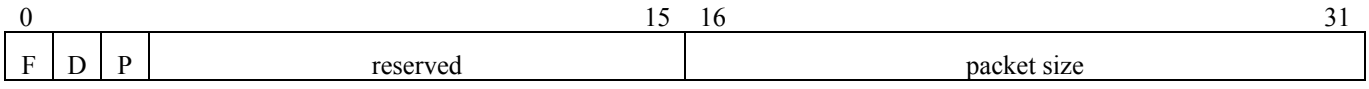
27.46 Stream Channel Packet Size Registers (SCPSx)

These registers indicate the packet size in bytes for the given stream channel. This is the total packet size, including all headers (IP, UDP and GVSP). For GVSP transmitters, they also provide a way to set the IP header "do not fragment" bit and to send stream test packet to the GVSP receiver.

The SCPSx register returns an error to the application if the corresponding stream channel is not supported and the application tries to access it, according to [O27-7cd].

Write access to this register is not possible while streaming (transmission) is active on this channel.

Address	0x0D04 + 0x40 * x with $0 \leq x < 512$. *[optional for 0 < x < 512 for GVSP transmitters and receivers]* *[Not applicable for peripherals]*
Length	4 bytes
Access type	Read/Write
Factory Default	0x40000240 if transmitter (SCPx Direction field is set to 0) Device-specific if receiver (SCPx Direction field is set to 1)



Bits	Name	Description
0	Fire test packet	When this bit is set and the stream channel is a transmitter, the GVSP transmitter will fire one test packet of size specified by bit 16-31. The "don't fragment" bit of IP header must be set for this test packet. If the GVSP transmitter supports the LFSR generator, then the test packet payload will be filled with data according to this polynomial. Otherwise, the payload data is "don't care". When the stream channel is a receiver, this bit is doesn't have any effect and always reads as zero.
1	Do Not Fragment	For GVSP transmitters, this bit is copied into the "do not fragment" bit of IP header of each stream packet. It can be used by the application to prevent IP fragmentation of packets on the stream channel. When the stream channel is a receiver, this bit is doesn't have any effect and always reads as zero.
2	Pixel Endianess	Endianess of multi-byte pixel data for this stream. 0: little endian 1: big endian This is an optional feature. A GVSP transmitter or receiver that does not support this feature must support little-endian and always leave that bit clear.
3 – 15	Reserved	Must be set to 0.
16 – 31	packet_size	For GVSP transmitters, this field represents the stream packet size to send on this channel, except for data leader and data trailer; and the last data packet which might be of smaller size (since packet size is not necessarily a multiple of block size for stream channel). The value is in bytes. If a GVSP transmitter cannot support the requested packet_size, then it must not fire a test packet when requested to do so. For GVSP receivers, this field represents the maximum GVSP packet size supported by the receiver. It is read-only.

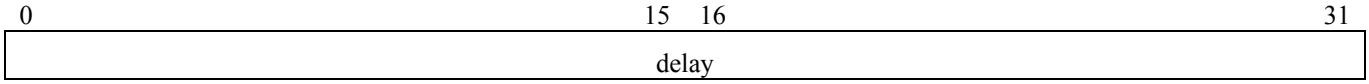
27.47 Stream Channel Packet Delay Registers (SCPDx)

These registers indicate the delay (in timestamp counter unit) to insert between each packet for the given stream channel. This can be used as a crude flow-control mechanism if the GVSP receiver cannot keep up with the packets coming from the device.

The SCPDx register returns an error to the application if the corresponding stream channel is a receiver or is not supported and the application tries to access it, according to [O27-7cd].

This delay normally uses the same granularity as the timestamp counter to ensure a very high precision in the packet delay. This counter is typically of very high frequency. Inter-packet delay is typically very small. If the timestamp is not supported, then this register has no effect.

Address	0x0D08 + 0x40 * x with 0 ≤ x < 512. *[optional for 0 < x < 512 for GVSP transmitters]* *[Not applicable for GVSP receivers and peripherals]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



Bits	Name	Description
0 – 31	Delay	Inter-packet delay in timestamp tick.

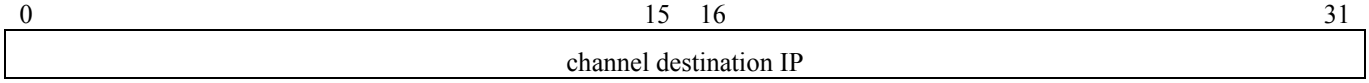
27.48 Stream Channel Destination Address Registers (SCDAx)

For GVSP transmitters, these registers indicate the destination IP address for the given stream channel. For GVSP receivers, these registers indicate the destination IP address from which the given receiver may receive data stream from.

The SCDAx register returns an error to the application if the corresponding stream channel is not supported and the application tries to access it, according to [O27-7cd].

For GVSP transmitters, write access to this register is not possible while streaming is active on this channel.

Address	0x0D18 + 0x40 * x with 0 ≤ x < 512. *[optional for 0 < x < 512 for GVSP transmitters and receivers]* *[Not applicable for peripherals]*
Length	4 bytes
Access type	Read/Write
Factory Default	0 if transmitter (SCPx Direction field is set to 0) Device-specific if receiver (SCPx Direction field is set to 1)



Bits	Name	Description
0 – 31	Channel destination IP	Stream channel destination IPv4 address. The destination address can be a multicast or a unicast address.

27.49 Stream Channel Source Port Registers (SCSPx)

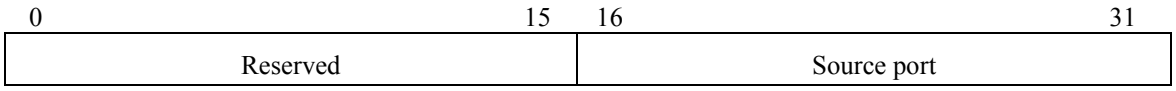
These optional registers indicate the source UDP port for the given GVSP transmitter stream channel. The purpose of this information is to allow the host application to take measures to ensure streaming traffic from the device is allowed back to the GVSP receiver. The expected usage of this is to send dummy UDP packets from the GVSP receiver stream reception port to that GVSP transmitter source port, thus making the streaming traffic appear to be requested by the GVSP receiver instead of unsolicited.

[CR27-45cd] If supported by the device, SCSPx MUST return a non-zero value corresponding to the source UDP port of the GVSP transmitter stream channel when SCPx is non-zero. It is valid for it to be non-zero at any point in time prior as well as long as it then remains constant for the duration of the control session.

[CR27-46cd] If supported by the device, when SCPx is non-zero the device MUST silently ignore any UDP traffic coming from the address and port combination listed in SCDAx and SCPx (if stream channel has been opened) targeted at the device SCSPx port.

The SCSPx register returns an error to the application if the corresponding stream channel is a receiver or is not supported and the application tries to access it, according to [O27-7cd].

Address	0x0D1C + 0x40 * x with $0 \leq x < 512$. *[optional for $0 \leq x < 512$ for GVSP transmitters]* *[Not applicable for GVSP receivers and peripherals]*
Length	4 bytes
Access type	Read-Only
Factory Default	0 if unavailable



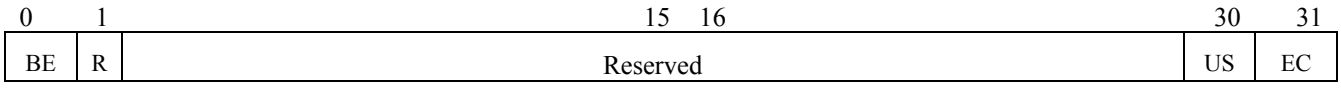
Bits	Name	Description
0 - 15	Reserved	Must be 0
16 - 31	Source Port	Indicates the UDP source port GVSP traffic will be generated from while the streaming channel is open.

27.50 Stream Channel Capability Registers (SCCx)

These optional registers provide a list of capabilities specific to the given stream channel.

These registers return an error to the application if they are not supported and the application tries to access them, according to [O27-7cd].

Address	0x0D20 + 0x40 * x with 0 ≤ x < 512. *[optional for 0 ≤ x < 512 for GVSP transmitters and receivers]* *[Not applicable for peripherals]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



Bits	Name	Description
0	Big and Little Endian supported (BE)	Indicates this stream channels supports both big and little endian. Note that all stream channel must support little endian.
1	IP reassembly supported (R)	For GVSP receivers, indicates this stream channel supports the reassembly of fragmented IP packets. Otherwise, it reads as zero.
2-29	Reserved	Must be 0
30	Unconditional streaming supported (US)	For GVSP transmitters, indicates that the stream channel supports unconditional streaming capabilities. Otherwise, it reads as zero.
31	Extended Chunk Data supported (EC)	Indicates that the stream channel supports the extended chunk data payload type.

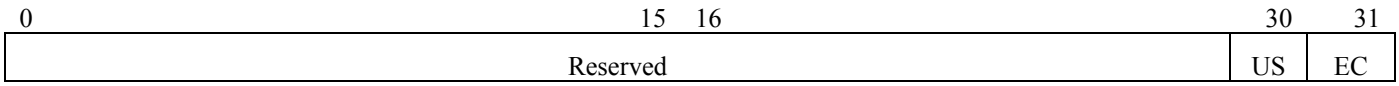
27.51 Stream Channel Configuration Registers (SCCFGx)

These optional registers provide additional control over optional features specific to the given stream channel. These optional features must be indicated by the Stream Channel Capability register of the given stream channel.

For instance, they can be used to enable GVSP transmitters to use the extended chunk data payload type.

These registers return an error to the application if they are not supported and the application tries to access them, according to [O27-7cd].

Address	0x0D24 + 0x40 * x with 0 ≤ x < 512. *[optional for 0 ≤ x < 512 for GVSP transmitters and receivers]* *[Not applicable for peripherals]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



Bits	Name	Description
0 – 29	Reserved	Always 0
30	Unconditional streaming enabled (US)	Enable GVSP transmitters (SCPx Direction field set to 0) to continue to stream if the control channel of its associated device is closed or regardless of any ICMP messages, such as the destination unreachable messages, received by the device associated to them. This bit has no effect for GVSP receivers (SCPx Direction field is set to 1). In the latter case, it always read back as zero.
31	Extended Chunk Data enable (EC)	Enable GVSP transmitters (SCPx Direction field set to 0) to use the extended chunk data payload type. This bit has no effect for GVSP receivers (SCPx Direction field is set to 1). In the latter case, it always read back as zero.

27.52 Manifest Table

The optional Manifest Table starts with a header indicating the number of entries in the table followed by a list of entries, each describing one document. This table is only present if the Manifest Table capability bit is set in the GVCP Capability register (at address 0x0934).

For 8-byte entries in the Manifest Table, the application needs to access the MSB (bit 0 to 31, lowest address) before accessing the LSB (bit 32 to 63, highest address).

Address	0x9000 *[optional]*
Length	512 bytes
Access type	Read
Factory Default	Device-specific

Address	Name	Support	Type	Length (bytes)	Description
0x9000	ManifestHeader	M*	R	8	Header of the manifest table
0x9000+8	ManifestEntry_1	M*	R	8	First entry in the manifest table
0x9000+16	ManifestEntry_2	O	R	8	Second entry in the manifest table
…	…	…	…	…	…
0x9000+8*N	ManifestEntry_N	O	R	8	Entry N in the manifest table
…	…	…	…	…	
0x9000+8*63	ManifestEntry_63	O	R	8	Last entry in the manifest table

*: mandatory only if Manifest Table is supported.

27.52.1 ManifestHeader

The ManifestHeader contains the number of manifest entries.

The number of entries can be 0, in which situation the First URL register (at address 0x0200) and Second URL register (at address 0x0400) shall be used to retrieve the XML device description file.

Bits	Name	Description
0 – 5	NumEntries	Number (N) of entries in the Manifest Table where 0≤N≤63. When N equals 0, then the First URL register and Second URL register must be used.
6 – 63	Reserved	Must be set to 0.

27.52.2 ManifestEntry

Each ManifestEntry describes the type and version of one XML document and refers to a pair of URL registers indicating the location of the XML document. This URL pair must provide addresses outside of the bootstrap register section. This means the URL pair must be stored in the manufacturer-specific register space (which starts at address 0xA000).

Bits	Name	Description
0 – 5	XMLMajorVersion	The major version number of the referenced GenApi XML file
6 – 11	XMLMinorVersion	The minor version number of the referenced GenApi XML file
12 – 17	XMLSubMinorVersion	The subminor version number of the referenced GenApi XML file
18 – 23	SchemaMajorVersion	The major version number of the schema used by the XML file.
24 – 29	SchemaMinorVersion	The minor version number of the schema used by the XML file
30 – 31	Reserved	Must be 0
32 – 63	URLRegisterAddress	This field contains the address to the URLPair Register used to name and locate the document. This address must be located in the manufacturer-specific register space except if this register points to the First URL register at address 0x0200.

The ManifestRegister is defined in a way that the First URL Register and the Second URL Register defined since version 1.0 of this specification can be reused.

A minimum Manifest Table for a camera supporting an XML file with revision 3.1.0 which is compliant to the GenApi schema version 1.0 would for example look like this:

```
Manifest.ManifestHeader.NumEntries = 1
Manifest.ManifestEntry_1.XMLMajorVersion = 3
Manifest.ManifestEntry_1.XMLMinorVersion = 1
Manifest.ManifestEntry_1.XMLSubMinorVersion = 0
Manifest.ManifestEntry_1.SchemaMajorVersion = 1
Manifest.ManifestEntry_1.SchemaMinorVersion = 0
Manifest.ManifestEntry_1.URLRegisterAddress = 0x0200
```

27.52.3 URL Pair

For each document, a primary and a secondary URL is given describing how to retrieve the document. An application should first try the primary URL and only if that fails fall back to the secondary URL. This is the same scheme used for the First URL bootstrap register (at address 0x0200) and Second URL bootstrap register (at address 0x0400).

[CR27-47cd] If a URL Pair is supported, then the URL strings referenced by URL Pair MUST use the character set indicated in the “Device Mode” register.

Address	Device-specific
Length	1024 bytes
Access type	Read
Factory Default	Device-specific

Bytes	Name	Description
0 – 511	Primary URL	NULL-terminated string providing the primary URL to the document.
512 – 1023	Secondary URL	NULL-terminated string providing the secondary URL to the document.

[CR27-48cd] If a URL Pair is supported, then in case a file is zipped, the referenced document MUST be the first one to be found in the ZIP file. Any number of other files may follow but their content is not defined by this specification.

[CR27-49cd] If a URL Pair is supported, then the Ntuple {XMLMajorVersion, XMLMinorVersion, XMLSubMinorVersion, SchemaMajorVersion, SchemaMinorVersion} in the manifest entry MUST change if the document's content changes. Thus this Ntuple can be used as key for caching the document

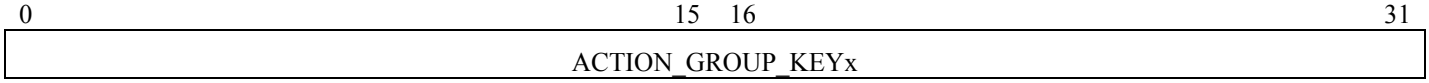
27.53 Action Group Key Registers (ACTION_GROUP_KEYx)

These optional registers provide the group key associated with the given device signal (ACTION_x). Up to 128 actions can be supported by the device (ACTION_0 up to ACTION_127).

These registers return an error to the application if they are not supported and the application tries to access them, according to [O27-7cd].

When action commands are supported, these registers provide the 32-bit group key the device uses to match against the group_key field of incoming ACTION_CMD packets for the given device signal.

Address	0x9800 + 0x10 * x with 0 ≤ x < 128. *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Bits	Name	Description
0 – 31	ACTION_GROUP_KEYx	Action Group Key entry.

27.54 Action Group Mask Registers (ACTION_GROUP_MASKx)

These optional registers provide the group mask associated with the given device signal (ACTION_x). Up to 128 actions can be supported by the device.

These registers return an error to the application if they are not supported and the application tries to access them, according to [O27-7cd].

When action commands are supported, these registers provide the 32-bit group mask the device uses to "bitwise AND" against the group_mask field of incoming ACTION_CMD packets for the given device signal.

Address	0x9804 + 0x10 * x with $0 \leq x < 128$. *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0 (disabled)

0	15	16	31
ACTION_GROUP_MASKx			

Bits	Name	Description
0 – 31	ACTION_GROUP_MASKx	Action Group Mask entry.

28 Standard Features List for Cameras

28.1 Introduction

The GigE Vision specification relies on GenICam™ (www.genicam.org) to describe the features supported by the camera. This description takes the form of an XML device description file respecting the syntax defined by the GenApi module of the GenICam™ specification.

[R28-1cd] To allow interoperability between GigE Vision cameras and GigE Vision application software, any GigE Vision device MUST provide an XML device description file compliant to the syntax of the GenApi module of GenICam™.

This XML file is retrieved and interpreted by the application software to enumerate the features supported by the device. The XML device description file provides the mapping between a device feature and the device register supporting it.

Since a large portion of GigE Vision devices are cameras, this section provides a list of mandatory features that must be present in the GigE Vision camera XML description file. Note that non-camera devices are not covered by this section. Any camera providing an XML device description file compliant to the syntax of the GenApi module of GenICam™ and including the mandatory features presented in this section is considered suitable for GigE Vision.

Note the requirements in this section only apply to camera devices. Therefore, all requirements and objectives below are conditional as they only apply to this type of GigE Vision devices.

28.2 GenICam™

GenICam™ is standard of the European Machine Vision Association (EMVA, www.emva.org).

GenICam™ provides a high level of dynamism since the feature mapping can be tailored to a specific camera. This is very different from the GigE Vision bootstrap registers which forces a unique mapping for all cameras. This dynamism offers the advantage that features of the camera can be determined and described by the camera manufacturer. The naming of these features can thus follow the manufacturer naming convention.

The drawback of this flexibility is that application software cannot recognize the meaning of a particular feature name. A way around this limitation is to provide a set of standard feature names to be used across various camera models. This way, application software is aware of the meaning associated to a given feature name. The extent to which these standard feature names are defined may limit the freedom of camera manufacturer to implement a given feature. Therefore care should be taken not to over-specify all features.

28.3 Level of Interoperability

An important consideration is the level of interoperability achieved between GigE Vision cameras and application software.

The simplest level of interoperability is realized when a graphical user interface (GUI) simply displays the list of features. This is often realized by a camera configuration program. In this case, it is the user who

looks and interprets the meaning of each feature. On-line help (such as tooltips) can be used to explain the meaning of the feature.

A more complex level of interoperability is achieved when the software is able to correctly interpret the functionality associated to a particular feature name. In this case, the software can take appropriate decision without the need for the user to get involved.

One issue with the level of interoperability is the number of characteristics associated to a feature:

1. Name
2. Representation (integer, float, boolean, enumeration, …)
3. Unit of measurement
4. Behavior

For instance, the feature "Gain" can be expressed as a float using the decibels unit where the value represents the level of amplification to apply to the analog video signal coming out of the sensor. But one could also define the Gain as an enumeration that can take a pre-determined set of value, such as {GAIN_x1, GAIN_x2, GAIN_x4}. Thus only standardizing the feature name is insufficient to guaranty interoperability.

Considering the large number of camera manufacturers and image processing software provider, it is difficult to reach a consensus on feature name, representation, unit and behavior. In many situations, the representation of a feature is directly linked to the camera architecture.

28.4 Use Cases

One of the goals of GigE Vision is to allow a user to buy a GigE Vision camera and easily interface it to any GigE Vision compliant application software. In order to guaranty a minimal level of interoperability, the GigE Vision specification addresses two use cases:

1. Continuous acquisition and display of images
2. Simplest GigE Vision camera

The features required to support these two use cases are defined mandatory by this specifications. These mandatory features are described in this section.

Some other features are listed in a separate document: "GenICam Standard Features Naming Convention". The GigE Vision specification recommends to camera manufacturers they use the name provided in that document whenever possible to facilitate interoperability with third-party software.

The mandatory feature list only provides for a single stream channel. If multiple stream channels are available, follow the recommendation provided in "GenICam Standard Features Naming Convention". This involves a separate feature called the "Selector" used to indicate the index of the stream channel to configure.

28.4.1 Use Case #1: Continuous Acquisition and Display

For this test case, the application must be able to initialize the camera to start a free-running acquisition and to display the acquired images to the screen. The camera's factory default settings must ensure the camera shows a suitable live image when acquisition control is turned on without any further configuration.

To achieve this, the following actions are required:

1. Control the camera using GVCP
2. Create a stream channel using GVSP registers
3. Retrieve image characteristics through the XML camera description file
4. Allocate image buffers on the PC
5. Start the continuous acquisition through the stream channel

Step 1 and 2 require the use of the GigE Vision bootstrap registers.

Step 3 and 5 require the use of the standard features provided in the XML description file of the camera.

Step 4 does not require any interaction with the camera.

28.4.2 Use Case #2: Simplest GigE Vision Camera

For this test case, we consider the simplest camera possible. This is basically the equivalent of an RS-170 analog camera. This type of camera does not offer any of the following: trigger control, exposure control, analog gain control, etc. It is only a basic camera that acquire continuously at its nominal frame rate.

The idea is that mandatory features must be present on all cameras. Features not required on all cameras cannot be mandatory.

28.5 XML Description File Mandatory Features

[CR28-2st] On top of the bootstrap registers that are required to control the camera and instantiate stream channels, all GigE Vision cameras MUST support the feature provided in Table 28-1 in their XML description files.

Table 28-1: GigE Vision Mandatory Feature for XML description file

Name	Interface	Access	Units	Description
"Width"	IInteger	R/(W)	pixels	Width of the image
"Height"	IInteger	R/(W)	pixels	Height of the image. For linescan, this represents the height of the frame created by combining consecutive lines together.
"PixelFormat"	IEnumeration	R/(W)	N/A	Pixel Format specified in GigE Vision specification. Refer to the Pixel format section (p. 159)
"PayloadSize"	IInteger	R	Bytes	Maximum number of bytes transferred for each image on the stream channel, including any end-of-line, end-of-frame statistics or other stamp data. This is the maximum total size of data payload for a block. UDP and GVSP headers are not considered. Data leader and data trailer are not included. This is mainly used by the application software to determine size of image buffers to allocate (largest buffer possible for current mode of operation). For example, an image with no statistics or stamp data as PayloadSize equals to (width x height x pixel size) in bytes. It is strongly recommended to retrieve PayloadSize from the camera instead of relying on the above formula.

"AcquisitionMode"	IEnumeration	R/(W)	N/A	Used by application software to set the mode of acquisition. This feature indicates how image are sequenced out of the camera (continuously, single shot, multi-shot, …) Only a single value is mandatory for GigE Vision. {"Continuous"} *Continuous*: Configures the camera to stream an infinite sequence of images that must be stopped explicitly by the application. Note that the AcquistionMode can have more than this single entry. However, only this one is mandatory. The "AcquisitionStart" and "AcquisitionStop" features are used to control the acquisition state.
"AcquisitionStart"	ICommand	(R)/W	N/A	Start image acquisition using the specified acquisition mode.
"AcquisitionStop"	ICommand	(R)/W	N/A	Stop image acquisition using the specified acquisition mode.

Note: Access modes given between parentheses are optional.

This list is only intended for camera devices. Non-camera devices do not have to support those features.

It is not required that all GigE Vision bootstrap registers are exposed via the XML description file although it is recommended as per [O27-37cd]. The XML description file is intended to expose features which are relevant to the user.

28.6 Width and Height Features

Width and Height represent the dimensions of the image coming out of the camera. This is basically the dimensions of the area of interest (AOI) that gets extracted from the sensor. Figure 28-1 shows the various features used to describe the AOI. Other features (MaxHeight, MaxWidth, Offset Y, Offset X) are recommended names to use, as defined in "GenICam Standard Features Naming Convention". But only Width and Height are mandatory.

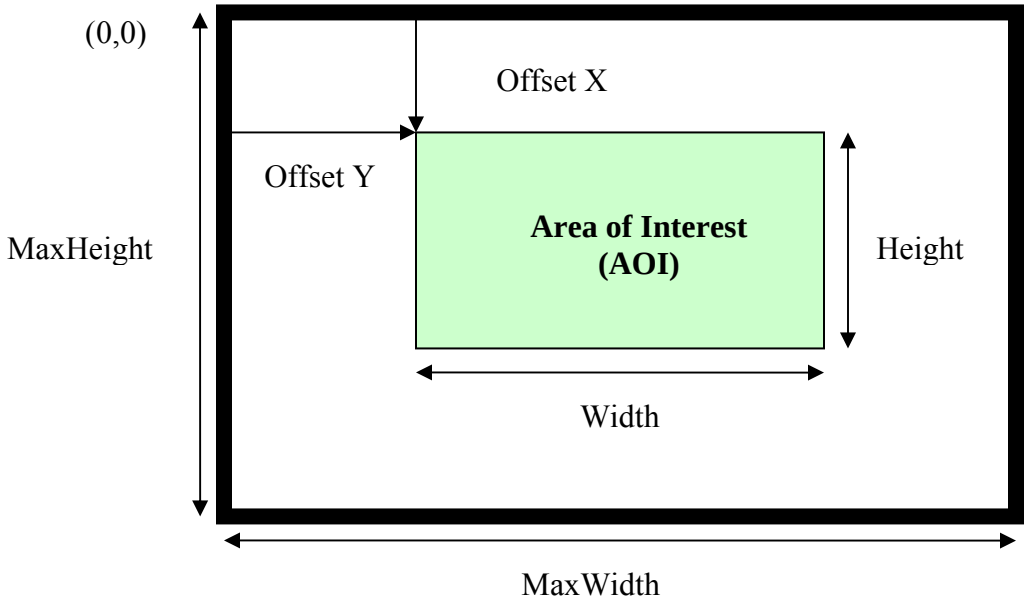


Figure 28-1: Width and Height Features

[CR28-3st] Width and Height features MUST be represented using the IInteger interface of GenICam (64-bit integer).

[CR28-4st] Width and Height features MUST be expressed in pixels.

Width can take any value ranging from 1 up to MaxWidth.

[CR28-5st] Width feature MUST take only positive values.

[CR28-6st] Width feature MUST be readable.

Width might be writable if it is possible to configure the size of the AOI. Default value for Width is typically the sensor width.

Height can take any value ranging from 1 up to the MaxHeight.

[CR28-7st] Height feature MUST take only positive values.

[CR28-8st] Height feature MUST be readable

Height might be writable if it is possible to configure the size of the AOI. Default value for Height is typically the sensor height.

If no other features than Width and Height are provided to describe the AOI, then the image starts at position (0, 0). That is equivalent to

- Offset Y = 0
- Offset X = 0
- MaxWidth = Width
- MaxHeight = Height

When Width and Height are writable, they can be used to crop the image coming out of the sensor.

For linescan cameras, Height represents the number of lines combined together to form a frame. Each of these frames is encapsulated between a data leader and a data trailer packet, as defined by GVSP. Note it is perfectly legal to have frames of only one line (this is the traditional definition of linescan).

Examples

1) For a 640x480 8-bit monochrome areascan camera:
   - Width = 640
   - Height = 480

2) For a 640x480 RGB 8-bit packed monochrome areascan camera:
   - Width = 640
   - Height = 480

3) For a 4Kpixels linescan camera using a frame size of 64 lines:
   - Width = 4096
   - Height = 64

4) For a 4Kpixels linescan camera with frame of one line:
   - Width = 4096
   - Height = 1

28.7 PixelFormat Feature

PixelFormat provides the type of pixel as defined in GVSP (see p. 159).

[CR28-9st] PixelFormat feature MUST be represented using the IEnumeration interface of GenICam.

~~[CR28-10d] If only one pixel format is supported by the camera, then PixelFormat feature MUST be read-only. Otherwise, it MUST be readable and writable.~~

[CR28-11st] A camera MUST use the strings (and corresponding values) provided in Table 28-2 for PixelFormat enumeration.

Table 28-2: PixelFormat Strings

Pixel Format	Name	Value (hexadecimal)
GVSP_PIX_MONO8	“Mono8”	0x01080001

Pixel Format	Name	Value (hexadecimal)
GVSP_PIX_MONO8_SIGNED	"Mono8Signed"	0x01080002
GVSP_PIX_MONO10	"Mono10"	0x01100003
GVSP_PIX_MONO10_PACKED	"Mono10Packed"	0x010C0004
GVSP_PIX_MONO12	"Mono12"	0x01100005
GVSP_PIX_MONO12_PACKED	"Mono12Packed"	0x010C0006
GVSP_PIX_MONO14	"Mono14"	0x01100025
GVSP_PIX_MONO16	"Mono16"	0x01100007
GVSP_PIX_BAYGR8	"BayerGR8"	0x01080008
GVSP_PIX_BAYRG8	"BayerRG8"	0x01080009
GVSP_PIX_BAYGB8	"BayerGB8"	0x0108000A
GVSP_PIX_BAYBG8	"BayerBG8"	0x0108000B
GVSP_PIX_BAYGR10	"BayerGR10"	0x0110000C
GVSP_PIX_BAYRG10	"BayerRG10"	0x0110000D
GVSP_PIX_BAYGB10	"BayerGB10"	0x0110000E
GVSP_PIX_BAYBG10	"BayerBG10"	0x0110000F
GVSP_PIX_BAYGR12	"BayerGR12"	0x01100010
GVSP_PIX_BAYRG12	"BayerRG12"	0x01100011
GVSP_PIX_BAYGB12	"BayerGB12"	0x01100012
GVSP_PIX_BAYBG12	"BayerBG12"	0x01100013
GVSP_PIX_BAYGR10_PACKED	"BayerGR10Packed"	0x010C0026
GVSP_PIX_BAYRG10_PACKED	"BayerRG10Packed"	0x010C0027
GVSP_PIX_BAYGB10_PACKED	"BayerGB10Packed"	0x010C0028
GVSP_PIX_BAYBG10_PACKED	"BayerBG10Packed"	0x010C0029
GVSP_PIX_BAYGR12_PACKED	"BayerGR12Packed"	0x010C002A
GVSP_PIX_BAYRG12_PACKED	"BayerRG12Packed"	0x010C002B
GVSP_PIX_BAYGB12_PACKED	"BayerGB12Packed"	0x010C002C
GVSP_PIX_BAYBG12_PACKED	"BayerBG12Packed"	0x010C002D
GVSP_PIX_BAYGR16_PACKED	*"BayerGR16Packed" (deprecated, use BayerGR16)*	0x0110002E
GVSP_PIX_BAYRG16_PACKED	*"BayerRG16Packed" (deprecated, use BayerRG16)*	0x0110002F
GVSP_PIX_BAYGB16_PACKED	*"BayerGB16Packed" (deprecated, use BayerGB16)*	0x01100030

Pixel Format	Name	Value (hexadecimal)
GVSP_PIX_BAYBG16_PACKED	*"BayerBG16Packed" (deprecated, use BayerBG16)*	0x01100031
GVSP_PIX_BAYGR16	"BayerGR16"	0x0110002E
GVSP_PIX_BAYRG16	"BayerRG16"	0x0110002F
GVSP_PIX_BAYGB16	"BayerGB16"	0x01100030
GVSP_PIX_BAYBG16	"BayerBG16"	0x01100031
GVSP_PIX_RGB8_PACKED	"RGB8Packed"	0x02180014
GVSP_PIX_BGR8_PACKED	"BGR8Packed"	0x02180015
GVSP_PIX_RGBA8_PACKED	"RGBA8Packed"	0x02200016
GVSP_PIX_BGRA8_PACKED	"BGRA8Packed"	0x02200017
GVSP_PIX_RGB10_PACKED	"RGB10Packed"	0x02300018
GVSP_PIX_BGR10_PACKED	"BGR10Packed"	0x02300019
GVSP_PIX_RGB12_PACKED	"RGB12Packed"	0x0230001A
GVSP_PIX_BGR12_PACKED	"BGR12Packed"	0x0230001B
GVSP_PIX_RGB16_PACKED	"RGB16Packed"	0x02300033
GVSP_PIX_RGB10V1_PACKED	"RGB10V1Packed"	0x0220001C
GVSP_PIX_RGB10V2_PACKED	"RGB10V2Packed"	0x0220001D
GVSP_PIX_RGB12V1_PACKED	"RGB12V1Packed"	0x02240034
GVSP_PIX_RGB565_PACKED	"RGB565Packed"	0x02100035
GVSP_PIX_BGR565_PACKED	"BGR565Packed"	0x02100036
GVSP_PIX_YUV411_PACKED	"YUV411Packed"	0x020C001E
GVSP_PIX_YUV422_PACKED	"YUV422Packed"	0x0210001F
GVSP_PIX_YUV422_YUYV_PACKED	"YUYVPacked"	0x02100032
GVSP_PIX_YUV444_PACKED	"YUV444Packed"	0x02180020
GVSP_PIX_RGB8_PLANAR	"RGB8Planar"	0x02180021
GVSP_PIX_RGB10_PLANAR	"RGB10Planar"	0x02300022
GVSP_PIX_RGB12_PLANAR	"RGB12Planar"	0x02300023
GVSP_PIX_RGB16_PLANAR	"RGB16Planar"	0x02300024

Bayer pixel format names represent the first 2 pixels of the Bayer mosaic of the sensor.

Examples

1) For a 640x480 8-bit monochrome areascan camera:

- PixelFormat = “Mono8”

2) For a 4Kpixels RGB 8-bit packed into 32-bit monochrome linescan camera:

- PixelFormat = “RGBA8Packed”

28.8 PayloadSize Feature

PayloadSize is a read-only feature representing the maximum number of data bytes sent for one block_id in the data payload packets on the stream channel. This does not include IP, UDP, GVSP headers or the data leader and data trailer packets. But it does include any data that can be appended to the image itself. The primary usage of PayloadSize is to provide an easy to retrieve buffer size to allocate for data transfer on the image stream channel.

In the case where data is of variable size, then the maximum possible value must be provided by PayloadSize.

[CR28-12st] PayloadSize feature MUST be represented using the IInteger interface of GenICam (64-bit integer).

[CR28-13st] PayloadSize feature MUST be expressed in bytes (8-bit value).

[CR28-14st] PayloadSize feature MUST only take positive values.

The typical value for PayloadSize is (Width x Height x Pixel Size) when no extra information is appended to the image. If the stream channel concatenates other information to the image (such as end-of-frame statistics or padding), then the PC buffer required to store the data might by larger than the size of the image.

Examples

1) For a 640x480 8-bit monochrome areascan camera:

- PayloadSize = 640 x 480 = 307200

2) For a 4Kpixels RGB 8-bit packed into 32-bit monochrome linescan camera with frames of 64 lines:

- PayloadSize = 4K pixels x 4 bytes/pixels x 64 lines = 1048576

28.9 AcquisitionMode Feature

The AcquistionMode feature is used to determine the sequencing of images. This typically refers to the number of images captured after the acquisition has been started. It could represent uninterrupted acquisition or the acquisition of a pre-determined number of frames. Note that this is totally independent from the stream channel being opened or not.

[CR28-15st] AcquisitionMode feature MUST be readable.

[CR28-16st] When AcquisitionMode provides multiple values in its enumeration, then it has to be writable.

[CR28-17st] AcquisitionMode feature MUST be represented by the IEnumeration interface of GenICam. It is mandatory to list at least the following mode of acquisition:

1. "Continuous"

By default, Acquisition mode MUST take the "Continuous" state.

Note that other acquisition modes are recommended by the "GenICam Standards Feature Naming Convention". The string used to represent a given mode of acquisition is provided in Table 28-1. The value associated to each string is specific to the camera.

Table 28-3: AcquisitionMode Strings

Mode	String
Uninterrupted acquisition	"Continuous"

AcquisitionMode should only be changed when image acquisition is stopped. Result of changing AcquisitionMode after issuing an AcquisitionStart command (but before an AcquisitionStop command) is not defined by this specification. This feature should only be changed after an AcquisitionStop command has been fired.

[CR28-18st] When AcquisitionMode is set to "Continuous", image acquisition is initiated by the "AcquisitionStart" command and will execute until the "AcquisitionStop" command is issued by the application. It is allowed to have other commands (such as an optional AcquisitionAbort command) end the acquisition.

28.10 AcquisitionStart Feature

The AcquisitionStart command begins image acquisition using the mode specified by AcquisitionMode. It is expected that image acquisition starts as soon as possible after the AcquisitionStart command has been issued.

[CR28-19st] AcquisitionStart feature MUST be realized using the ICommand feature of GenICam.

~~[CR28-20st] Image acquisition MUST start as soon as possible after issuing the AcquisitionStart command.~~

Note the stream channel must be setup through the bootstrap registers and the other camera features must be initialized prior to start an acquisition.

[CR28-21st] Re-issuing an AcquisitionStart command after it has already been initiated, but before an AcquisitionStop command, MUST not affect the image acquisition status. That is, camera must remain with acquisition active.

28.11 AcquisitionStop Feature

The AcquisitionStop command terminates image acquisition after the current frame is completed.

[CR28-22st] AcquisitionStop feature MUST be realized using the ICommand feature of GenICam.

[CR28-23st] Image acquisition MUST stop as soon as possible on a frame boundary (after the data trailer) when issuing the AcquisitionStop command.

[CR28-24st] Re-issuing an AcquisitionStop command after it has already been initiated, but before an AcquisitionStart command, MUST not affect the image acquisition status. That is, camera must remain with acquisition inactive.

28.12 Link to Naming Convention

This section only described the mandatory features provided in the XML camera description file. Other recommended features are covered in "GenICam Standard Feature Naming Convention" (available from http://www.genicam.org). This naming convention should be respected when implementing features not described in this section.

Proposal for new recommended feature names shall be addressed to the GenICam committee.

29 Appendix 1 – Compliancy Matrix

The Compliancy Matrix must be filled by manufacturer for each product in order to obtain certification.

The Product Identification table of the compliancy matrix includes a section to define what group of requirements and objectives are applicable to a given product based upon its nature (application, device, GVSP transmitter and GVSP receiver). For instance, requirements using the 'c', 'cd', 's' and 'st' suffixes apply to a standard camera since it includes a device and one or more GVSP transmitters.

For each requirement and objective, please indicate the level of compliancy from the following list:

Acronym	Name	Description
FC	Fully Compliant	The requirement or objective is supported and fully complies with the text of this specification.
NC	Not Compliant	The requirement or objective is implemented, but does not comply with the text of this specification.
NS	Not Supported	The objective is not supported. This is not a valid entry for a requirement.
NA	Not Applicable	The requirement or objective is not applicable to the registered product. For instance, requirements applicable only to applications do not apply to typical GigE Vision cameras and are marked as NA.

For each conditional requirement or conditional objective, please indicate the level of compliancy from the following list:

Acronym	Name	Description
FC	Fully Compliant	1) The condition is not supported (hence conditional requirement does not apply). "NS" can also be used in this situation. or 2) The condition is met, and the conditional requirement or conditional objective is supported and fully complies with the text of this specification.
NC	Not Compliant	The condition is met, and the conditional requirement or conditional objective is implemented, but does not comply with the text of this specification.
NS	Not Supported	The condition is not met. The conditional requirement or conditional objective is not supported. "FC" can also be used in this situation.
NA	Not Applicable	Same as in the previous table.

29.1 Matrix for GigE Vision Product

The following tables describe the compliancy matrix being used to register products compliant with this version of the specification.

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

Product Identification		
Manufacturer:	Product/Family Name:	☐ Individual ☐ Family
Supported Entities: ☐ Application ('c' and 'ca' requirements and objectives) ☐ Device ('c' and 'cd' requirements and objectives) ☐ GVSP Transmitter ('s' and 'st' requirements and objectives) ☐ GVSP Receiver ('s' and 'sr' requirements and objectives)		
Version/Release:		
Comments/Notes:		

Table 29-1: Product Compliancy Matrix

Req.	Compliant	Comments
[R7-1c]		
[R7-2c]		
[R7-3c]		
[R7-4c]		
[O7-5c]		
[R7-6c]		
[R7-7c]		
~~[R7-16c]~~	N/A	Removed
[R12-6c]		
[CR12-10c]		
[R13-1c]		
[O13-2c]		
[R13-3c]		
[O13-4c]		
[O13-5c]		
[R13-6c]		
[O13-7c]		
[R13-9c]		
[R14-3c]		
[R14-7c]		
[R14-23c]		
[R14-24c]		
[R14-33c]		

Req.	Compliant	Comments
[R14-34c]		
[R14-37c]		
[R14-38c]		
[R14-41c]		
[R14-42c]		
[CR14-45c]		
[CR14-46c]		
[CR14-49c]		
[CR14-50c]		
[CR14-59c]		
[CR14-68c]		
[CR14-73c]		
[CR14-75c]		
[CR15-7c]		
[CR15-8c]		
[CR15-13c]		
[CR15-14c]		
[R16-1c]		
[O5-1ca]		
[O7-8ca]		
[R7-9ca]		
[O7-10ca]		
[O7-11ca]		
[R7-12ca]		
~~[R7-17ca]~~	N/A	Removed
[R7-18ca]		
[R7-19ca]		
[R8-3ca]		
[R9-3ca]		
[R9-13ca]		
[R10-4ca]		
[CR10-9ca]		
[CR11-3ca]		
[CR11-5ca]		
[CR11-7ca]		
[CR11-8ca]		

Req.	Compliant	Comments
[R12-9ca]		
[CR14-10ca]		
[O14-44ca]		
[O14-54ca]		
[R14-55ca]		
[R14-62ca]		
[O14-71ca]		
[CR14-76ca]		
[O15-15ca]		
~~[O15-4ca]~~	N/A	Removed
~~[O15-9ca]~~	N/A	Removed
[R17-3ca]		
[R17-4ca]		
[O17-6ca]		
[O27-3ca]		
[R27-4ca]		
[O27-9ca]		
[CR27-26ca]		
[CR27-28ca]		
[R27-30ca]		
[CR27-31ca]		
[R27-36ca]		
[R3-1cd]		
[R3-2cd]		
[CR3-3cd]		
[CR3-4cd]		
[R3-5cd]		
[R3-6cd]		
[R3-7cd]		
[CR3-8cd]		
[CR3-9cd]		
[CO3-11cd]		
[R3-12cd]		
[O3-13cd]		
[CR3-14cd]		
[CR3-15cd]		

Req.	Compliant	Comments
[CR3-16cd]		
[O3-18cd]		
[R3-19cd]		
[R3-20cd]		
[R3-21cd]		
[R4-1cd]		
[R4-2cd]		
[R4-3cd]		
[O4-4cd]		
[R4-5cd]		
[R4-6cd]		
[O4-7cd]		
[R7-13cd]		
[R7-14cd]		
~~[O7-15cd]~~	N/A	Removed
[R8-2cd]		
[R9-1cd]		
[O9-2cd]		
[R9-4cd]		
[R9-5cd]		
[R9-6cd]		
[CO9-7cd]		
[CR9-8cd]		
[R9-9cd]		
[R9-10cd]		
[R9-11cd]		
[R9-12cd]		
[R9-14cd]		
[R9-15cd]		
[R9-16cd]		
[CO9-17cd]		
[CR9-18cd]		
[CR11-1cd]		
[CO11-2cd]		
[CR11-4cd]		
[CR11-6cd]		

Req.	Compliant	Comments
[R12-1cd]		
[R12-2cd]		
[CR12-3cd]		
[O12-4cd]		
[R12-5cd]		
[R12-7cd]		
~~[O13-4cd]~~	N/A	Removed
~~[O13-5cd]~~	N/A	Removed
[R14-1cd]		
[R14-2cd]		
[R14-4cd]		
[R14-5cd]		
[O14-6cd]		
[R14-8cd]		
[R14-9cd]		
[R14-11cd]		
[R14-12cd]		
[R14-13cd]		
[R14-14cd]		
[R14-15cd]		
[R14-16cd]		
[R14-17cd]		
[R14-18cd]		
[CR14-19cd]		
[CR14-20cd]		
[R14-21cd]		
[R14-22cd]		
[R14-25cd]		
[R14-26cd]		
[R14-27cd]		
[CR14-28cd]		
[CR14-29cd]		
[R14-30cd]		
[R14-31cd]		
[R14-32cd]		
[R14-35cd]		

Req.	Compliant	Comments
[R14-36cd]		
[R14-39cd]		
[R14-40cd]		
[R14-43cd]		
[CR14-47cd]		
[CR14-48cd]		
[CR14-51cd]		
[CR14-52cd]		
[O14-53cd]		
[CR14-56cd]		
[CO14-57cd]		
[R14-58cd]		
[CR14-60cd]		
[CR14-61cd]		
[CO14-63cd]		
[CR14-64cd]		
[CR14-65cd]		
[CR14-66cd]		
[CR14-67cd]		
~~[CR14-68cd]~~	N/A	Removed
[CR14-69cd]		
[CR14-70cd]		
[CR14-72cd]		
[CR14-74cd]		
[CR15-1cd]		
[CO15-2cd]		
[O15-3cd]		
[CR15-5cd]		
[CO15-6cd]		
[CR15-10cd]		
[CR15-11cd]		
[CO15-12cd]		
[R17-1cd]		
[CR17-2cd]		
[R17-5cd]		
[O18-1cd]		

Req.	Compliant	Comments
[R18-2cd]		
[CR19-1cd]		
[R20-1cd]		
[R27-1cd]		
[R27-2cd]		
[R27-5cd]		
[R27-6cd]		
[O27-7cd]		
[O27-8cd]		
[R27-10cd]		
[R27-11cd]		
[R27-12cd]		
[R27-13cd]		
[R27-14cd]		
[R27-15cd]		
[CO27-16cd]		
[CO27-17cd]		
[CR27-18cd]		
[R27-19cd]		
[R27-20cd]		
[CR27-21cd]		
[CR27-22cd]		
[CR27-23cd]		
[O27-24cd]		
[O27-25cd]		
[CR27-27cd]		
[CR27-29cd]		
[CR27-32cd]		
[CR27-33cd]		
[CR27-34cd]		
[CR27-35cd]		
[O27-37cd]		
[CR27-38cd]		
[CR27-39cd]		
[CR27-40cd]		
[CR27-41cd]		

Req.	Compliant	Comments
[CR27-42cd]		
[CR27-43cd]		
[CR27-44cd]		
[CR27-45cd]		
[CR27-46cd]		
[CR27-47cd]		
[CR27-48cd]		
[CR27-49cd]		
[R28-1cd]		
[R8-1s]		
[R10-1s]		
[R10-2s]		
[CR10-7s]		
[R22-1s]		
[R22-2s]		
[R23-1s]		
[CR23-12s]		
[CR23-13s]		
[CR23-14s]		
[R24-1s]		
[R24-2s]		
[CR24-5s]		
[CR24-6s]		
[CR24-7s]		
[CR24-8s]		
[CR24-9s]		
[CR24-13s]		
[CR24-14s]		
[CR24-15s]		
[CR24-16s]		
[CR24-17s]		
[R24-21s]		
[CR24-22s]		
[CR24-23s]		
[CR24-24s]		
[CR24-25s]		

Req.	Compliant	Comments
[CR24-26s]		
[CR24-31s]		
[CR24-32s]		
[O25-3s]		
[CR25-4s]		
[CR25-5s]		
[CR25-6s]		
[CR25-7s]		
[CR25-8s]		
[CR25-9s]		
[CR25-10s]		
[CR25-12s]		
[CR25-13s]		
[CR25-14s]		
[CR25-15s]		
[CR25-16s]		
[CR25-17s]		
[CR25-18s]		
[CR25-19s]		
[CR25-20s]		
[CR25-21s]		
[CR25-22s]		
[CR25-23s]		
[CR25-24s]		
[CR25-25s]		
[CR25-26s]		
[CR25-27s]		
[CR25-28s]		
[CR25-29s]		
[CR25-30s]		
[CR25-31s]		
[CR25-32s]		
[CR25-33s]		
[CR25-34s]		
[CR25-35s]		
[CR25-36s]		

Req.	Compliant	Comments
[CR25-37s]		
[CR25-38s]		
[CR25-39s]		
[CR25-40s]		
[CR25-41s]		
[CR25-42s]		
[CR25-43s]		
[CR25-44s]		
[CR25-45s]		
[CR25-46s]		
[CR25-47s]		
[CR25-48s]		
[CR25-49s]		
[CR25-50s]		
[CR25-51s]		
[CR25-52s]		
[CR25-53s]		
[CR25-54s]		
[CR25-55s]		
[CR25-56s]		
[CR25-57s]		
[CR25-58s]		
[R12-11sr]		
[R22-3sr]		
[CO24-30sr]		
[O10-3st]		
[R10-5st]		
[R10-6st]		
[CR10-8st]		
[R12-8st]		
[O18-3st]		
[R18-4st]		
[R22-4st]		
[R23-2st]		
[CR23-3st]		
[CR23-4st]		

Req.	Compliant	Comments
[CR23-5st]		
[CO23-6st]		
[CR23-7st]		
[CR23-8st]		
[CR23-9st]		
[CO23-10st]		
~~[CR23-12st]~~	N/A	Removed
~~[CR23-13st]~~	N/A	Removed
[R24-3st]		
[R24-4st]		
[CR24-10st]		
[CR24-11st]		
[R24-12st]		
[R24-18st]		
[R24-19st]		
[R24-20st]		
[R24-27st]		
[R24-28st]		
[CR24-29st]		
[O25-1st]		
[R25-2st]		
[CR25-11st]		
[CR28-2st]		
[CR28-3st]		
[CR28-4st]		
[CR28-5st]		
[CR28-6st]		
[CR28-7st]		
[CR28-8st]		
[CR28-9st]		
[CR28-11st]		
[CR28-12st]		
[CR28-13st]		
[CR28-14st]		
[CR28-15st]		
[CR28-16st]		

Req.	Compliant	Comments
[CR28-17st]		
[CR28-18st]		
[CR28-19st]		
~~[CR28-20st]~~	N/A	Removed
[CR28-21st]		
[CR28-22st]		
[CR28-23st]		
[CR28-24st]		

Automated Imaging Association
P.O. Box 3724
Ann Arbor, MI 48106-3724

30 Document History

Version	Date	Description of Changes
1.0	2006-05-03	-Initial release
draft 1.1a	2007-03-21	-Added CCP to list of acronym. -Added reference to DHCP RFC in [R3-12d]. -Added reference to LLA RFC in [R3-20d] -Clarified to use application dynamic port as destination for GVCP ACK message. -Changed CPP to CCP in figure 9-1 and figure 9-2. -[R9-12d] now clearly indicate MCP and SCPx must be cleared on Control Channel Connection closure. -[O13-2] now indicates the device should validate the 0x42 key. -Clarification to [O13-4d] to indicate the acknowledge message value must be the unsupported command message value + 1. -New requirement [R14-62a] to force ACKNOWLEDGE bit to be set in flag field of a DISCOVERY_CMD. -Added the GEV_STATUS_PACKET_RESEND status code. Note this is an informational status code. -New conditional objective [CO14-63d] to recommend usage of GEV_STATUS_PACKET_RESEND status code on a packet resend. -Removed various references to feature not supported in XML (Device-specific message, Device-specific status code, Device-specific pixel format). -Clarified inter-packet delay is from end to start of packets. -Added support for GVSP_PIX_MONO14. -Corrected a typo in figure for GVSP_PIX_YUV422_PACKED. -Clarified that FC can be use in compliancy matrix for a conditional requirement/objective if the condition is false. -Removed [O3-22d] since LLA cannot fail (it retries infinitely). -Added a note to [R3-5d] to indicate 2 acceptable interpretations possible for DHCP client when moving to LLA. -Add note above [CO3-10d] to indicate Persistent IP 0.0.0.0 and 255.255.255.255 are not valid when interface is configured. -Clarify behavior when DHCP lease expire. Added a new section for this. -Remove [O3-17d]. -New conditional requirement [R11-7a] to use the UDP source port from the message channel as the UDP destination port in the acknowledge message. -Note highlighted the spec. does not define an acknowledge timeout. -Change wording in [R9-6d] and [R9-15d] to reference FORCEIP_CMD. -Change to [CR14-48d] to remove length = 0.

		-Adjusted the definition of .ZIP file format in section 17. -Add a note about ICMP Destination Unreachable in section 20. -Removed [CO23-11d]. -Change to [O27-8d] to differentiate error returned on a read vs write access.
draft 1.1b	2007-05-19	-Clarify usage of CPP register when both exclusive_access and control_access are set. -Change [R24-10d] into a conditional requirement CR24-10d. -Possibility to broadcast FORCEIP_ACK. -Rewording of [CR14-61d] -Change GVSP_PIX_RGB (now deprecated) to GVSP_PIX_COLOR. -Remove [CR28-10d] -AcquisitionStart and AcquisitionStop features are optionally readable. -New status code: GEV_STATUS_WRONG_CONFIG. -Add a new bootstrap register at 0x0950 to represent the max. randomized delay for DISCOVERY_ACK -Change Version bootstrap register to refer to version 1.1 of this specification. -Clarification to [R4-3d] to support any type of broadcast request. -Remove [CO3-10d]. -Factory default of MCRC and MCTT are now device-specific. -New bootstrap register at 0x0954 for Heartbeat Control. New capability bit to indicate if Heartbeat can be disabled. -New optional bootstrap register to report current Link Speed of each supported network interface. A capability bit indicates if these registers are supported. -Deprecate events related to image acquisition. These should now be specified by GenICam Standard Naming Convention for GigE Vision. -Update reference to GenICam and Standard Feature Naming Convention to reflect most recent release. -New objective indicating bootstrap registers should be part of the XML device configuration file and respect the rules of GenICam Standard Feature Naming Convention for GigE Vision. -Further clarifications about .ZIP usage: only DEFLATE and STORE compression algorithms are permitted. -Added clarification to Chunk Data Payload format and an example. -New bootstrap registers to retrieve Primary Application Port and IP address. -Support for PENDING_ACK message and Pending Timeout bootstrap register. -Introduction of the Manifest Table. -Added support for the following pixel types: GVSP_PIX_BAYBG10_PACKED, GVSP_PIX_BAYGB10_PACKED, GVSP_PIX_BAYGR10_PACKED, GVSP_PIX_BAYRG10_PACKED, GVSP_PIX_BAYBG12_PACKED, GVSP_PIX_BAYGB12_PACKED, GVSP_PIX_BAYGR12_PACKED, GVSP_PIX_BAYRG12_PACKED, GVSP_PIX_BAYGR16_PACKED, GVSP_PIX_BAYRG16_PACKED, GVSP_PIX_BAYGB16_PACKED,

		GVSP_PIX_BAYBG16_PACKED, GVSP_PIX_YUV422_YUYV_PACKED, GVSP_PIX_RGB12V1_PACKED and GVPS_PIX_RGB16_PACKED. -Capability to support LFSR-generated data to fill the payload data of the test packet. -Option to support SCSPx and MCSP to help firewall traversal. Addition of stream channel and message channel capability registers. -Add SCCx bootstrap registers for per stream channel capabilities. Introduction of big endian support flag. -Add GEV_STATUS_PACKET_NOT_YET_AVAILABLE status code that can be used by linescan cameras when the line trigger rate is slower than the application timeout for packet retransmission. -Device Mode register factory default is now device-specific. -Clarified definitions of FC, NC and NS for conditional requirement and conditional objectives. Put them in tables with clear definition. -Remove [O13-8a]. -[R14-10a] is now a conditional requirement since application don't have requirement to support FORCEIP_CMD. [R14-15] and [R14-16] now only apply to devices. -[R24-27] and [R24-28] now only apply to devices. -[R25-2] now only applies to devices. -Reword [CR27-28a] to clarify the condition since an application does not have to use this feature.
draft 1.1c	2007-06-10	-Add support for ACTION_CMD and ACTION_ACK. Up to 128 actions can be defined. -Add new contributors to technical committee (members attending the technical meetings). -Fix pixel order for GVSP_PIX_BAYBG8 and GVSP_PIX_BAYBG10 and GVSP_PIX_BAYBG12. -Change example in [O4-4d] to remove an incorrect subnet for a PersistentIP address since we do not validate classes anymore. -Add indication for optional registers. -Put unique numbers to new requirements. -Add new requirements to the compliancy matrices. Indicates requirements no longer applicable.
Rev. 1.1RC1	2008-10-14	-Add note under [R14-5d] to highlight fact that device might not be able to adequately verify if it reside on same subnet as the application. -Insert clarification for READREG and WRITEREG when accessing string register. Little-endian device will convert data to big-endian as required by [R3-19]. -Highly recommend supporting WRITEMEM for device used through GenICam interface. -Bit 0 of Device Mode register indicates the endianess of all device registers. -Remove reference to chair and vice-chair of GigE Vision committee. This should be part of other AIA documentation. -Clarify device should move to next IP configuration scheme if PersistenIP is 0.0.0.0. -Remove use case from [O4-4d]. -Revisit the graphics for the various Pixel Formats. -Change to Manifest Table to support only XML documents. Other documents will be supported

		by entries within the XML file. - Indicate that Primary Application Source port and IP address must be reset when control channel is closed in [R9-12d]. -Add enable bit to GVCP Configuration register to enable extended status codes introduced in version 1.1. -Modify CR27-46d to only discard packets targeted at the device SCSPx port and MCSP port.
Rev. 1.1RC2	2008-10-29	-Correction to figure representing YUV411. -Set reference to GenICam version 1.1.1. -Add note to indicate why ACTION_CMD does not reset the heartbeat. -Discovery ACK Delay register is optional. Change [R27-42d] to [CR27-42d]. Add a capability bit for availability. Add capability bit when it is writable. -GVCP Configuration register default is 0 to allow backward compatibility to version 1.0 Application software. -Add 2 status codes: GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY and GEV_STATUS_PACKET_REMOVED_FROM_MEMORY. -Clarify length of LFSR algorithm. -Link speed registers expressed in Mbps in a single 32-bit register. -Add explanation for multi-word to access the lowest address first. -For compliancy matrix, allow FC to be used for unsupported conditional requirements. -Remove one of the conditions allowing a device to broadcast a DISCOVERY_ACK. -Remove action_id field in ACTION_CMD.
Rev 1.1RC3	2008-12-10	-Added capability bit in GVCP capability register to indicate if extended status codes are supported. -Add note below [CR14-61d] to indicate a device is to use GEV_STATUS_PACKET_UNAVAILABLE if it does not support extended packet resend status codes. -Change [O14-6d] to reflect the case when the Discovery ACK delay register is not supported. -Switch order of the conditions in Table 9-1.
Rev 1.1	2009-02-04	Official release of GigE Vision 1.1 specification
draft 1.2a	2009-03-29	Updated the first 2 pages of each section as a proof of concept for the new product classification initiative.
draft 1.2b	2009-05-07	Updated full bootstrap registers proposal for new product classification initiative.
draft 1.2c	2009-07-03	First draft of the new product classification proposal made available for public review. It also includes a number of the maintenance work identified for the release of version 1.2 of this specification. This includes: - Clarified requirements [O4-4cd], [R9-3ca], [R9-6cd], [R9-16cd], [CR14-61cd], [O15-4ca] and [O15-9ca]. - Clarified section describing device control (Section 9.6). - Updated the description of group_key and group_mask in Section 9.8 in order to be consistent with other sections of the standard text.

		- Described GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY and GEV_STATUS_PACKET_REMOVED_FROM_MEMORY in the core of the packet resend text (Section 14.7.2). - Removed PACKETRESEND_ACK from Section 16. - Clarified GEV_STATUS_WRONG_CONFIG status code in Section 18. - Clarified recommendation on how to handle destination unreachable messages in Section 20. - Removed PACKETRESEND_ACK from Data Resend Flowchart. - Clarified that the last data packet of a block can be padded so that all data packet have the same size. - Added the "Image Format" field in the table below [CR25-42]. - Fixed typos in requirements [CR27-45cd] and [CR27-46cd]. SCPx is now referenced as opposed to SCPSx and SCSPx. - Deleted minimal GenICam XML file. - Fixed link to GenICam Standard Features Naming Convention to point to GenICam web site. - Fixed some typos.
Rev 1.2RC1	2009-08-30	- Defined management entity. - Added primary application switchover concept. - Better documented suffix terminology for requirements and objectives. - Removed [O7-15cd], [R7-16c] since considered as redundant. - Augmented registers lists in Section 9.2, 10.1 and 11.1. - Added unconditional streaming concept. - Reworked Section 9.6 in order to improve its technical accuracy. - Added a note to clarify PENDING_ACK behavior with respect to secondary applications. - Updated Figure 9-8. - Moved note indented to promote transceivers compliance with GigE Vision 1.0 and 1.1 applications to Section 10. - Added a note with respect to the usage of GEV_STATUS_PACKET_UNAVAILABLE, GEV_STATUS_PACKET_AND_PREV_REMOVED_FROM_MEMORY and GEV_STATUS_PACKET_REMOVED_FROM_MEMORY. - Renamed [CR14-68cd] to [CR14-68c]. - Replaced [O15-4ca] and [O15-9ca] by [O15-15ca]. - Added extended chunk data payload type. - Replaced [CR23-12st] and [CR23-13st] by [CR23-12s] and [CR23-13s]. - Added RGB565 and BGR565 pixel formats. - Deprecated BayerGR16Packed, BayerRG16Packed, BayerGB16Packed and BayerBG16Packed pixel format names and replaced them by BayerGR16, BayerRG16, BayerGB16 and BayerBG16. - Removed [CR28-20st] since not testable. - Clarified some ambiguous text and fixed some typos.

		- Improved layout (deleted some carriage returns, etc.).
Rev 1.2RC2	2009-10-09	- Defined primary application switchover request concept. - Updated LSB and MSB definitions. - Updated GenICam version. - Removed [R7-17ca]. - Replaced [O13-4cd] and [O13-5cd] by [O13-4c] and [O13-5c]. - Clarified some ambiguous text and fixed some typos.
Rev 1.2	2009-12-23	Official release of GigE Vision 1.2 specification. - Removed RC2 tag. - Removed line numbers. - Added copyrights page. - Inserted page breaks in order to make the document more printer friendly. - Updated licensing statement in Section 1.1. - Updated list of contributors in Section 1.2.