GigE Vision

Camera Interface Standard for Machine Vision

version 1.0

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

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

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

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 a GigE Vision device to communicate with an application. 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 device using a crossover cable; a more complex one is a GigE camera on an IP network using routers. It is crucial that GigE devices 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 Giga-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/). This committee was co-chaired by Toshi Hori (JAI PULNiX) and George Chamberlain (Pleora Technologies) up to December 2005. In January 2006, Eric Carey (DALSA Coreco) was appointed as the chairmain with François Gobeil (Pleora Technologies) serving as vice-chairman. The following companies have participated to the GigE Vision Technical Committee.

- Basler AG
- Coreco
- DALSA
- GigaLinx
- JAI A/S and JAI PULNiX
- Matrox Electronic Systems
- National Instruments
- Pleora Technologies
- STEMMER IMAGING
- Toshiba TELI

1.3 Definitions and Acronyms

1.3.1 Definitions

Application	GigE Vision application software running on a host, typically a PC.
Control protocol	GigE Vision control protocol (GVCP) defining commands supported by GigE Vision compliant devices.
Device	GigE Vision compliant device, typically a camera.
Gratuitous ARP	An ARP request sent by the device to check if another device is using the same IP address.
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.
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 device or the application software. It is mandatory to support the feature to ensure interoperability.	[Rx-y<ad>]
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<ad>]
Absolute Objective	Feature that SHOULD be supported by the device or the application software. It is recommended, but not essential.	[Ox-y<ad>]
Conditional Objective	Feature the SHOULD be supported IF another feature is present.	[COx-y<ad>]

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. Product type: Application software (receiver of streams), Device (transmitter of streams) or both. ‘a’ represents a requirement exclusive to application software, ‘d’ represents a requirement exclusive to devices. When neither ‘a’ nor ‘d’ is specified, the requirement applies to both the device and the application software.

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

A requirement might apply to the device, the application software or both. A compliancy matrix is provided at the end of this document to summarize which requirements are applicable to device and to application software.

1.3.3 Acronyms

AOI	Area of Interest
BOOTP	Bootstrap Protocol
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
GigE	Giga-Ethernet
GVCP	GigE Vision Control Protocol
GVSP	GigE Vision Streaming Protocol
IANA	Internet Assigned Numbers Authority
IP	Internet Protocol
IPv4	Internet Protocol version 4
LLA	Link-Local Address
LSB	Least Significant Byte
MAC	Media Access Control
MSB	Most Significant Byte
MTU	Maximum Transmission Unit
OUI	Organizationally Unique Identifier
PC	Personal Computer
RFC	Request For Comments
ROI	Region of Interest
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

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

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 GenAPI	Generic Interface for Camera device, EMVA, version 1.0, July 2006
GenICam/GigE Vision Standard Features Naming Convention	Standard Features Naming Convention, draft 1.00.02, June 2006

NOTE – References listed above were current at the time of publication of this standard. All standards are subject to revision, and anyone using this standard is encouraged to investigate the possibility of applying the most recent editions of the standards referenced.

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

PART 1 – Device Discovery

2 Device Discovery Summary

2.1 Overview

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

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.
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 camera 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 section 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.

3 IP Configuration

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

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

[R3-2d] 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 cameras are power-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.

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

[CR3-4d] 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.

[R3-5d] 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

[R3-6d] 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.

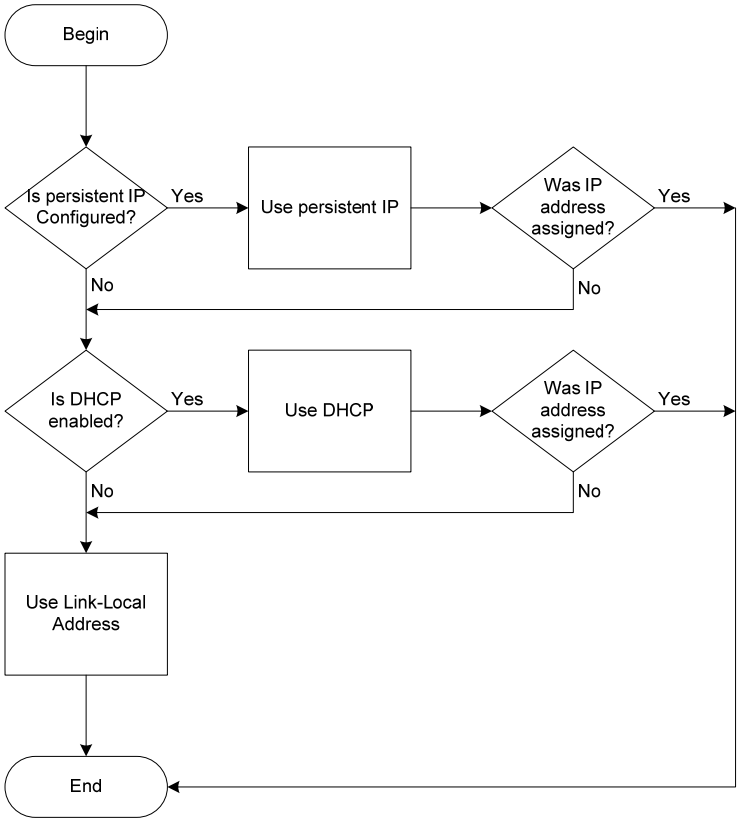


Figure 3-1: Protocol Selection Flowchart

[R3-7d] 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.

[CR3-8d] 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.

[CR3-9d] 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.

[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.

This validation should be performed when the device retrieves the Persistent IP address and subnet mask during IP configuration sequence.

[CO3-11d] 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.

[R3-12d] A device MUST support DHCP.

Refer to RFC2131 (Dynamic Host Configuration Protocol) and RFC2132 (DHCP Options and BOOTP Vendor Extensions) for more information.

[O3-13d] 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).

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

[CR3-15d] 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.

[CR3-16d] 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.

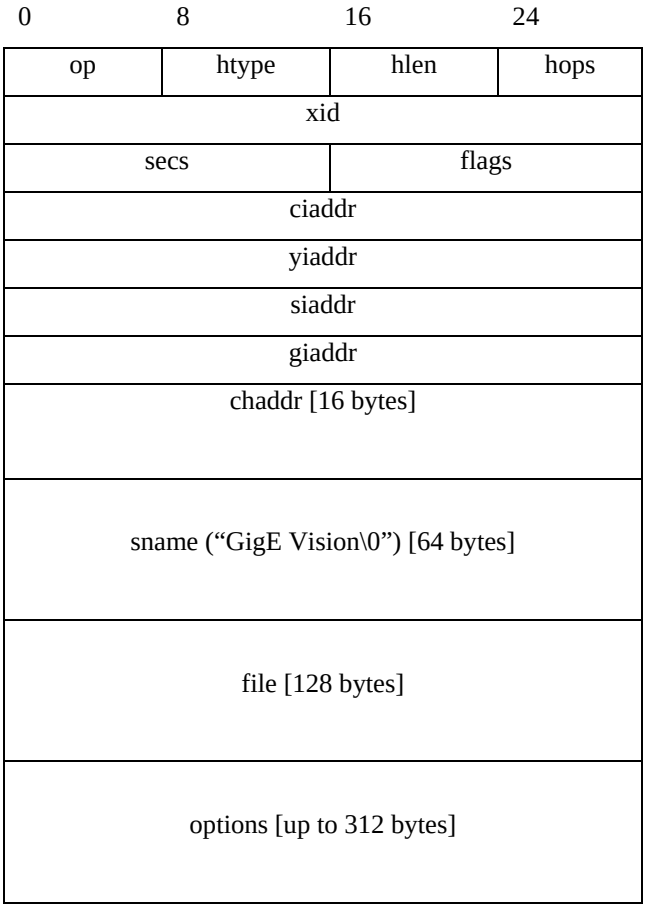


Figure 3-2: DHCP message

[O3-17d] The ‘sname’ field SHOULD be the ASCII null terminated string “GigE Vision”.

The DHCP server could use the ‘sname’ field to list GigE Vision devices. The way this information could be passed from the DHCP server to the application is left as a quality of implementation since it requires a strong interaction between the DHCP server and the application.

[O3-18d] 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).

[R3-19d] 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).

3.4 Link-Local Address

[R3-20d] The device MUST support Link-Local Address (LLA).

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

LLA is sometime called “Auto IP”.

Link-local address scheme is supported by Microsoft Windows and Apple MacOS when no DHCP server is present on the subnet. In this case, the device automatically selects an IP address in the 169.254.x.x class B range and then verifies if this IP address is indeed available (using ARP). This scheme is only suitable for

subnet consisting of a single segment where no routing device is available. It is plug'n'play from the device's point of view since it can autonomously select an IP address without having the user configure the segment (no need for an IP address server). The device will check that no conflict exists on the segment.

[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.

In this case, the device is only allowed to broadcast on the network to answer device discovery requests until its IP address is different from 0.0.0.0.

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).

Refer to RFC3927 (Dynamic Configuration of IPv4 Link-Local Addresses) for a complete description of Link Local Address.

4 Device Enumeration

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

[R4-2d] 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.

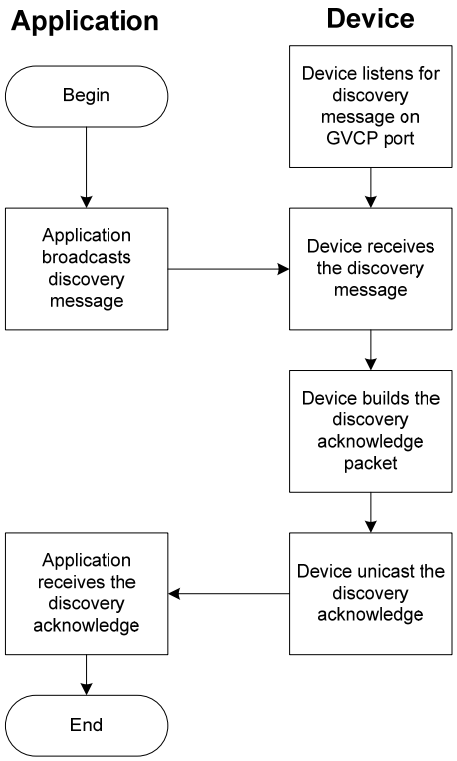


Figure 4-1: Device Discovery Flowchart

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.

[R4-3d] 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.

[O4-4d] If a device does not have a valid IP address after IP configuration sequence (such as 0.0.0.0 or a Persistent IP address on an incorrect 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.

[R4-5d] 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.

[R4-6d] 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.

[O4-7d] 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.

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.

[O5-1a] 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, it is going to timeout on the video stream not coming anymore from the device.

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

PART 2 – GVCP

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 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. 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.
- Define a mechanism for a GigE Vision device to send asynchronous event messages to a GigE Vision application.
- 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 section provides:

- Specification of the control protocol.
- Description of GVCP headers.
- List of all command and acknowledge messages.
- List of the different status codes representing error on the device.
- List of the different asynchronous event codes.

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

This control protocol does not cover GigE Vision data streaming protocol (GVSP). This is discussed in a different section of the standard.

7 GVCP Transport Protocol Considerations

For this version of the GigE Vision specification,

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

[R7-2] 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 engine.

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.

[R7-3] 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.

7.1.1 Fragmentation

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

[R7-4] 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.

[O7-5] 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).

[R7-6] Application and device 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.

[R7-7] 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.

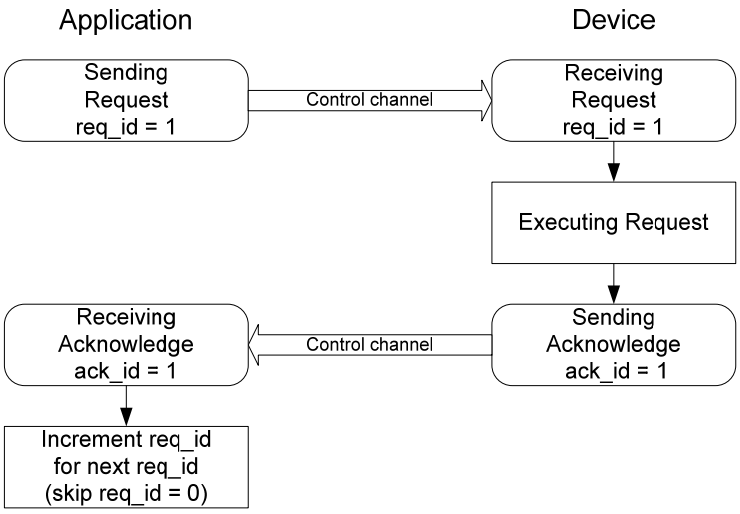


Figure 7-1: Use of Acknowledge

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.

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

[R7-9a] 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.

[O7-10a] 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.

[O7-11a] 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).

[R7-12a] 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.

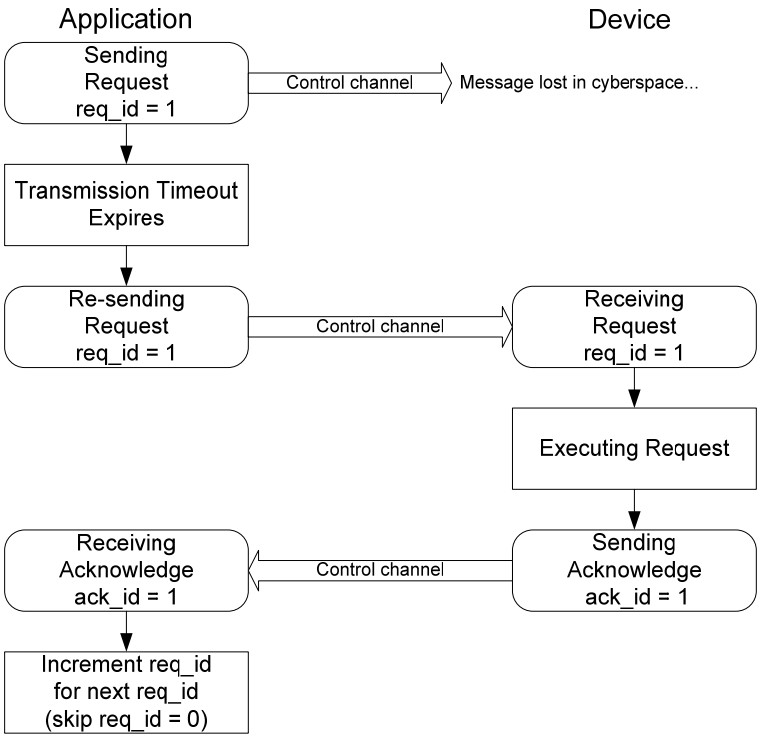


Figure 7-2: Timeout on Request

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.

[R7-13d] 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.

[R7-14d] 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).

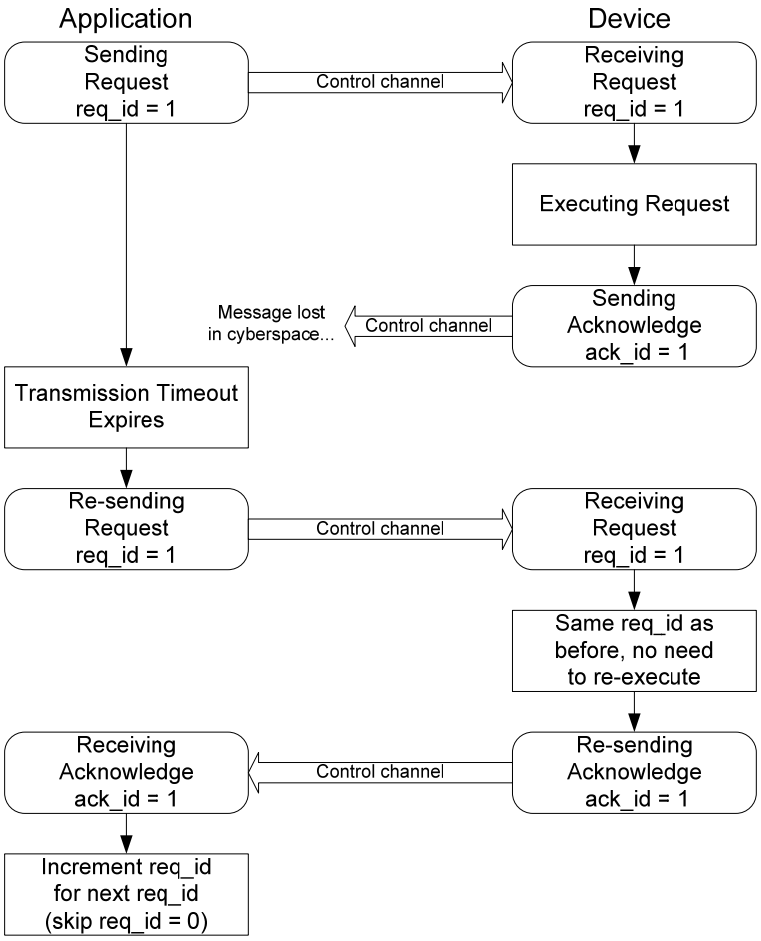


Figure 7-3: Timeout on Acknowledge

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.

[O7-15d] 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.

[R7-16] If a device or an application receives an erroneous 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.

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:

[R7-17a] 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.

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

[R7-19a] 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, GVCP introduces the concept of “channels”.

A channel is a virtual link used to transfer information between a device and an application. GVCP supports three types of channels:

1. ***Control channel*** (there is always 1 control channel for the primary application)
2. ***Stream channel*** (from 1 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.

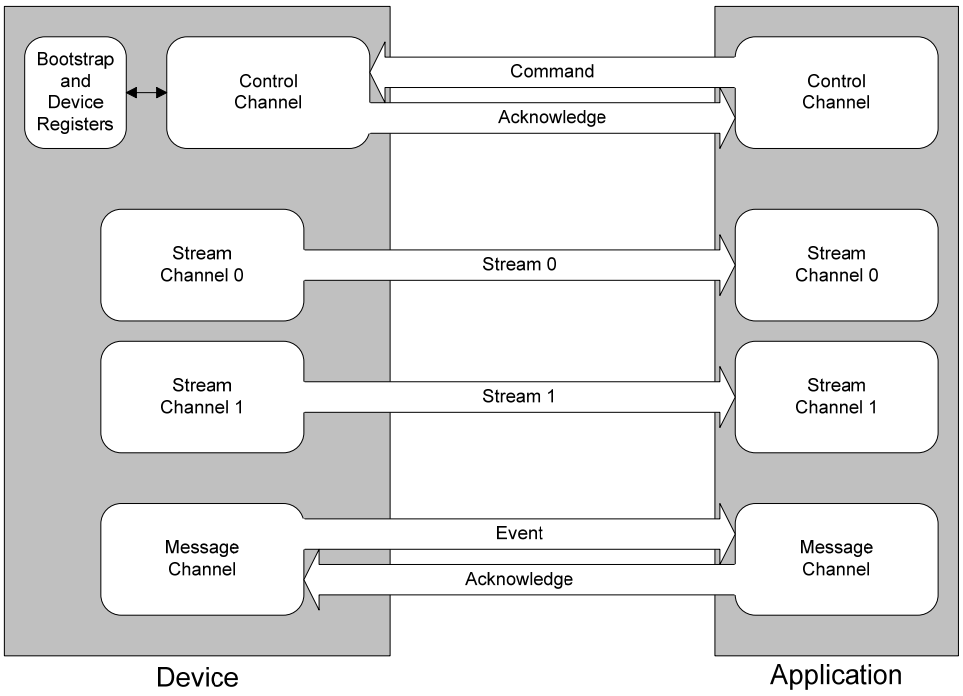


Figure 8-1: Channels Example

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.

[R8-1d] A device MUST offer at least one and at most 512 stream channels.

[R8-2d] A device MUST offer either one or no message channel.

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

The application can open or close these channels as needed.

[R8-3a] 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 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 image/data streams to 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).

[R9-1d] 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)

[O9-2d] 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).

[R9-3a] An application MUST NOT send a second command before it has received the first acknowledge (the only exceptions being 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 three 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.

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

[R9-5d] 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 only if there is no application registered with exclusive access or with control access.

[R9-6d] 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 DISCOVERY_CMD and PACKETRESEND_CMD.

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.

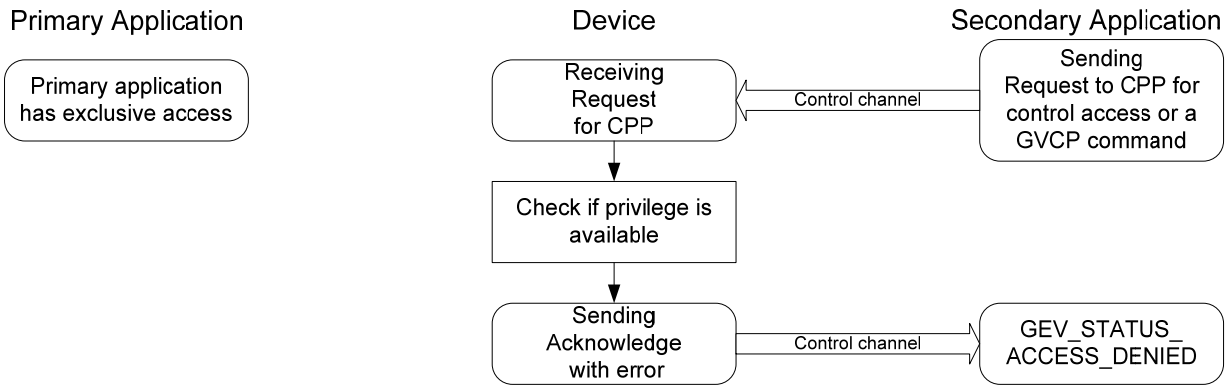


Figure 9-1: Exclusive Access

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.

[CO9-7d] 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.

[CR9-8d] When a device supports control access, it can grant control access only if there is no application registered with exclusive access or with control access. Otherwise, it MUST return an error (GEV_STATUS_ACCESS_DENIED).

Control access is granted by writing into the CCP register.

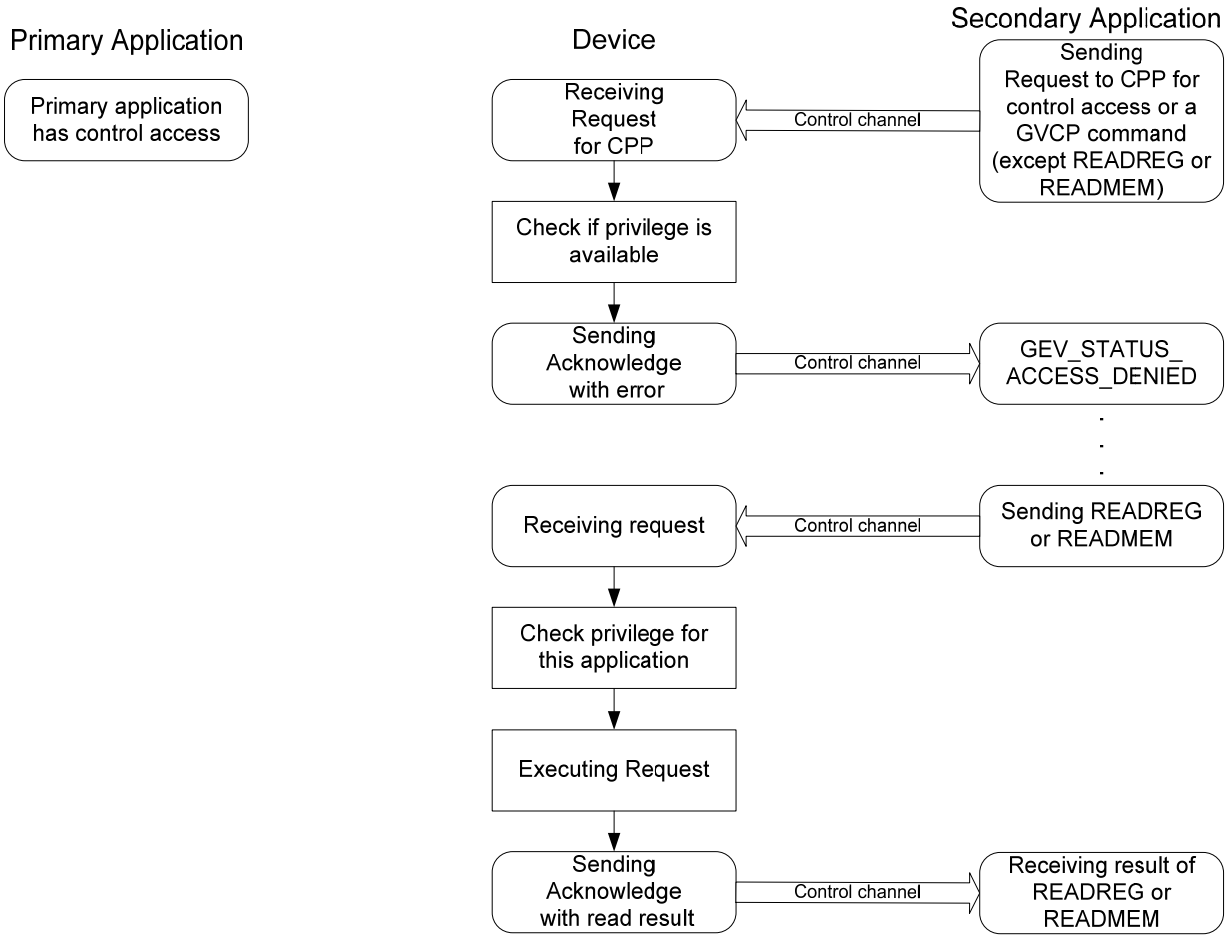


Figure 9-2: Control Access

3. ***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.

[R9-9d] 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 or control access)

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.

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

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. Heartbeat timeout

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

9.3 Opening a Control Channel

An application opens a control channel to take exclusive or control access. 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. 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.

[R9-10d] 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.

[R9-11d] A device MUST allow an application to directly switch between Exclusive and Control privilege without having to first disconnect the control channel.

9.4 Closing a Control Channel

A primary application closes its opened control channel by writing 0 to the CCP register. A secondary application cannot write into the CCP register. It gets a GEV_STATUS_ACCESS_DENIED from the device.

[R9-12d] When the primary control channel is closed by the application, the device MUST stop streaming, reset all connection states (including message and stream channels) 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. 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.

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.

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

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

[R9-15d] Any valid GVCP command from the primary application MUST reset the heartbeat counter on the corresponding device, with the exception of the PACKETRESEND_CMD and the DISCOVERY_CMD.

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.

[R9-16d] 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), 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 only send READREG and READMEM commands to read device state. All other commands return a GEV_STATUS_ACCESS_DENIED error code.

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.

PACKETRESEND command can also be sent at any time by any application and be correctly answered by the device.

If the device does not have a valid IP address at the end of the configuration cycle (such as IP 0.0.0.0), then it can broadcast the DISCOVERY_ACK message. Otherwise, it can use unicast transfer to the source of the discovery request.

10 Stream Channel

A stream channel enables data transfer from the device to the application. This transfer uses the GigE Vision Streaming Protocol (GVSP).

[R10-1d] A device MUST support at least one stream channel.

[R10-2d] A device 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 device may support up to 512 stream channels. The stream can transport any data type, such as images, histogram, image statistics, etc.

10.1 Stream Channel Registers

Each stream channel provides the following registers:

1. Stream Channel Port register (SCP)
2. Stream Channel Packet Size register (SCPS)
3. Stream Channel Packet Delay register (SCPD)
4. Stream Channel Destination Address register (SCDA)

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.

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

10.3 Closing a Stream Channel

[R10-4a] 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 device should stop streaming data as soon as possible after the stream channel is closed. When closing a stream channel, it is possible the last block of data is not totally sent to the application. But any packet has to be sent in its entirety. It is up to the application to correctly handle this situation.

[R10-5d] A device 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 device 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 application and device 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: 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 device 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 application 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 application, then the application can reduce the packet size. If the test packet reaches the application, then the application can increase the packet size. This way, the application can converge to the optimal packet size. In most scenarios, the larger the packet size, the lower the overhead of packet transmission.

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

The application will need to find a packet size supported by the device and by all network elements between the device and the host where the application resides.

Note: This is not a bullet-proof method of finding the maximum packet size when multiple routes are available from the device to the application. 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 device 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.

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

10.6 Impact of Multiple Network Interfaces

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

[CR10-7] 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.

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)

[CR11-1d] 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)

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.

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

11.3 Closing the Message Channel

[CR11-3a] 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.

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

[CR11-5a] When a message channel is supported, if an acknowledge message is received when the 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.

[CR11-6d] 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.

12 Device with Multiple Network Interfaces

A device may have multiple network interfaces.

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

[R12-2d] 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 GVSP packets (from device to application); they never receive GVCP packets (from application to device).

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

[O12-4d] 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).

[R12-5d] 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 device 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

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

[R12-7d] 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

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

[R12-9a] 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

[CR12-10] 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.

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

[O13-2] A recipient of a command message SHOULD perform minimal validation on the command header.

This minimally includes the validation of the key value against 0x42 (hexadecimal).

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

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

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

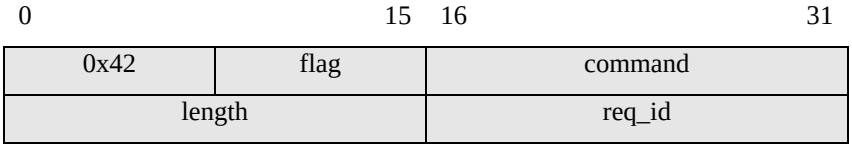


Figure 13-1: Command Message Header

0x42	8 bits	Hard-coded key code value used for message validation.
flag	8 bits	Upper 4 bits (bit 0 to bit 3) are common to all GVCP commands. Lower 4 bits (bit 4 to bit 7) are specific to each 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 3 to 0 – 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.

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

[O13-7] 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.

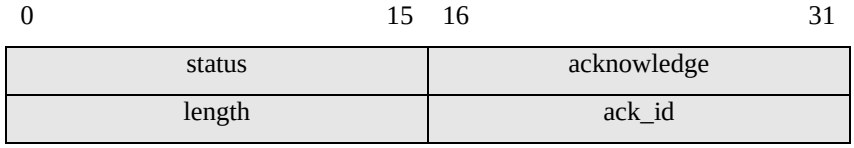


Figure 13-2: Acknowledge Message Header

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

[R13-9] 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.

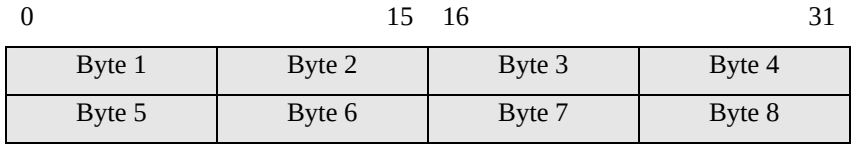


Figure 13-3: Byte Sequencing

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.

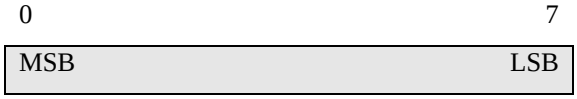


Figure 13-4: Byte sequencing for an 8-bit word

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:

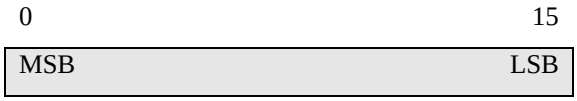


Figure 13-5: Byte sequencing for a 16-bit word

For a 32-bit word:

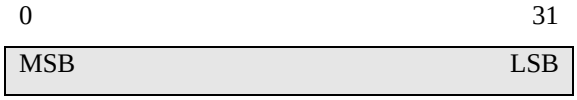


Figure 13-6: Byte sequencing for a 32-bit word

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

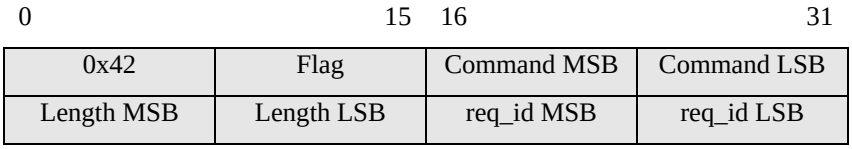


Figure 13-7: GVCP Header Showed in Byte Quantities

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.

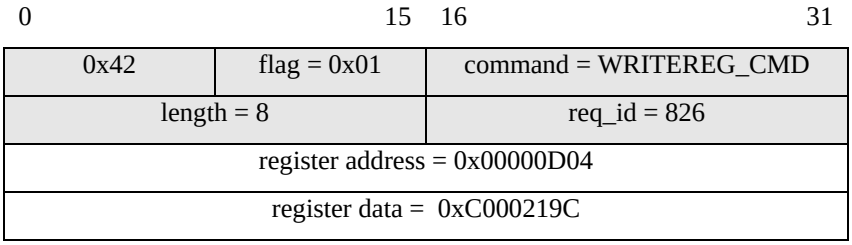


Figure 13-8: Example Command for Byte Ordering

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.

[R14-1d] 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

[R14-2d] 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.

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

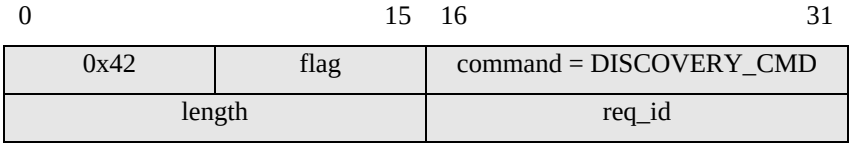


Figure 14-1: DISCOVERY_CMD message

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 in the following situation: the subnet of the DISCOVERY_CMD does not match the subnet of the device. When this bit is cleared, the device cannot broadcast its acknowledge message.

14.1.2 DISCOVERY_ACK

[R14-4d] 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.

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

A device may optionally broadcast its DISCOVERY_ACK when the following 3 conditions are met:

1. the DISCOVERY_CMD was not received on the device subnet
2. the DISCOVERY_CMD was broadcasted by the application
3. 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 3 conditions are met. This is left as a quality of implementation.

[O14-6d] 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.

[R14-7] 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).

[R14-8d] 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, “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.

[R14-9d] 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

[R14-10a] The FORCEIP_CMD MUST be broadcasted by the application on the GVCP port. It must contain the MAC address of the device to access.

[R14-11d] 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.

[R14-12d] A device MUST process FORCEIP_CMD only when there is no application with exclusive or control privilege. 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:

[R14-13d] 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.

[R14-14d] 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.

[R14-15] 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	
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

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.

[R14-16] 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.

14.3 READREG

[R14-17d] 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”.

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

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

[CR14-20d] 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 registers 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.

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

[R14-22d] 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

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

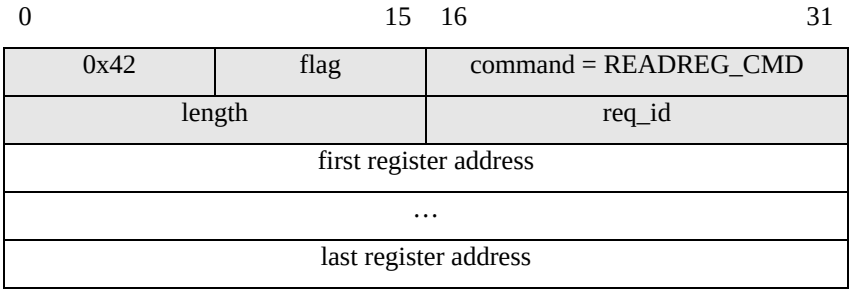


Figure 14-5: READREG_CMD message

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:

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

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

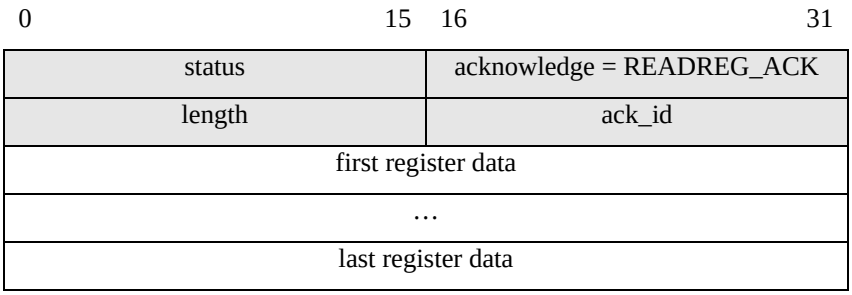


Figure 14-6: READREG_ACK message

READREG_ACK	
register data	Value of the data being requested

[R14-25d] 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

[R14-26d] 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”.

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

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

[CR14-29d] 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.

Only the primary application is allowed to send a WRITEREG message.

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

[R14-31d] A device MUST return a GEV_STATUS_ACCESS_DENIED status code to any secondary application trying to execute this command.

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. But once a device is granted exclusive or control access, then no other application can write to CCP.

[R14-32d] 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

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

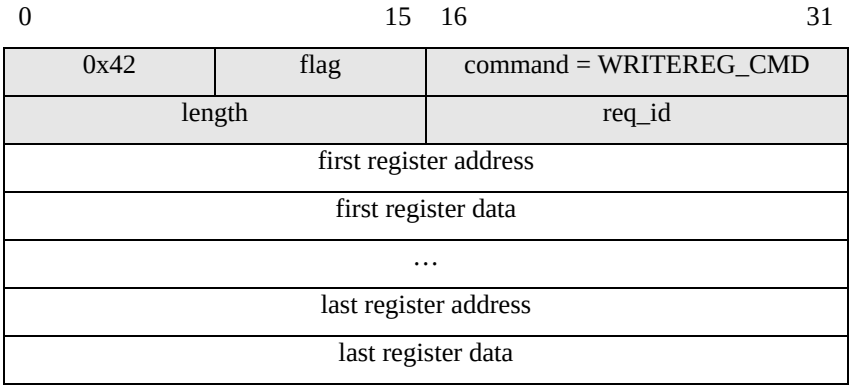


Figure 14-7: WRITEREG_CMD message

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

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

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

0 – 15	16 – 31
status	acknowledge = WRITEREG_ACK
length	ack_id
reserved	index

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.

[R14-35d] 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

[R14-36d] 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.

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

[R14-38] 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).

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

[R14-40d] 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

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

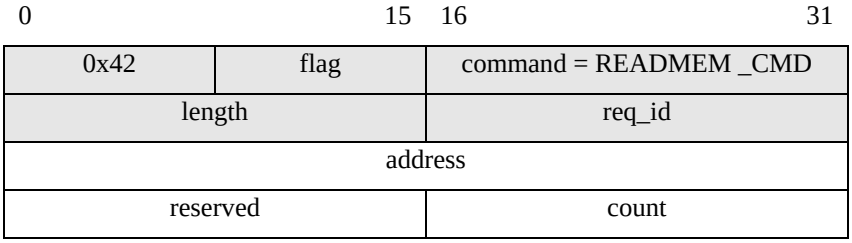


Figure 14-9: READMEM_CMD message

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

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

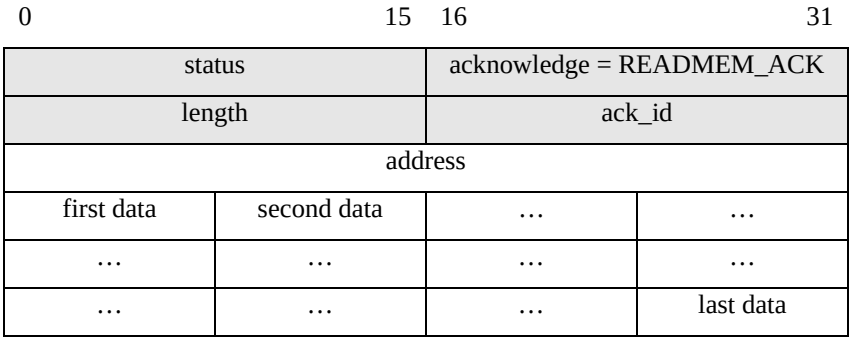


Figure 14-10: READMEM_ACK

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.

[R14-43d] 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.

[O14-44a] 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.

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

[CR14-46] 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.

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

[CR14-48d] If WRITEMEM is supported, a device MUST return a GEV_STATUS_ACCESS_DENIED status code to any secondary application trying to execute this command. In this case, the length field is set to 0 (no data payload).

14.6.1 WRITEMEM_CMD

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

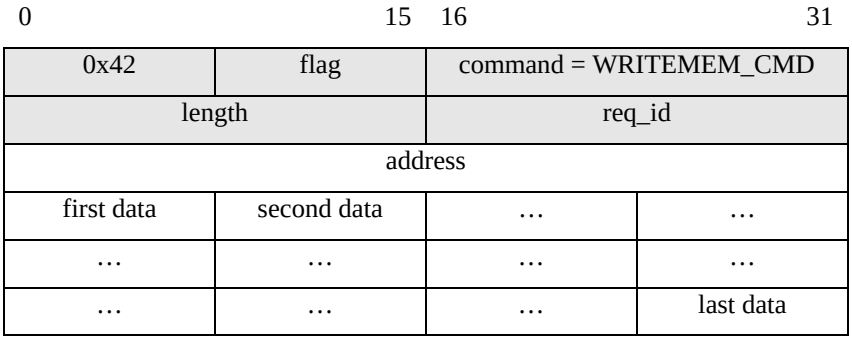


Figure 14-11: WRITEMEM_CMD message

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

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

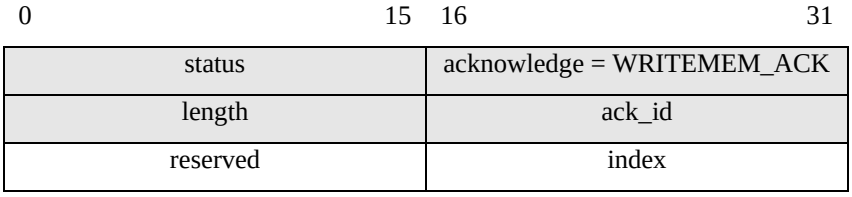


Figure 14-12: WRITEMEM_ACK message

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 519) where the error occurred.

[CR14-51d] 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 539); 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.

[CR14-52d] 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.

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

[O14-54a] 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) receiving stream packets can issue a PACKETRESEND message at any time to tell the device to resend a packet of the data stream, typically when this packet was not received by the application. 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.

[R14-55a] 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.

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

It is up to secondary applications to decide if they want to issue PACKETRESEND or not. This is left as a quality of implementation. A typical scenario is when the stream channel uses multicast. The main danger is that many applications might request the same packet to be resent, possibly overloading the device. 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.

[CO14-57d] A device SHOULD return GEV_STATUS_NOT_IMPLEMENTED in the stream packet to the application 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.

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

14.7.1 PACKETRESEND_CMD

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

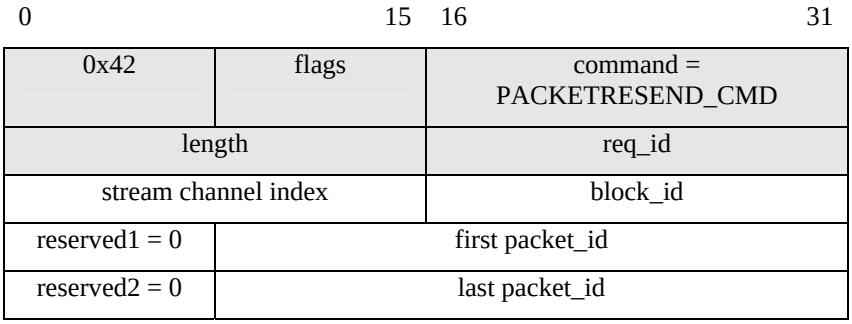


Figure 14-13: PACKETRESEND_CMD message

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 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.

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

[CR14-61d] When PACKETRESEND is supported, if the device 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 length field set to 0 and status code set to GEV_STATUS_PACKET_UNAVAILABLE for each applicable packet_id.

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 error code must be returned for any packet in that range that cannot be resent.

14.7.3 Packet Resend handling on the application-side

Since the amount of memory on the device is limited, it is important for the application to send any PACKETRESEND command as fast as possible to ensure the missing data is still available from the device memory. Otherwise, the device will return GEV_STATUS_PACKET_UNAVAILABLE because the data has been lost (possibly overwritten by newer data).

Handling of PACKETRESEND message by the 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 device is directly connected to the PC hosting the application 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, the application can use the packet_id to track down the sequence of packets. If a packet_id is skipped, then the application can 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, the 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 the application to offer a mechanism that suits the uncertainty introduced by packet re-ordering.

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.

[CR15-1d] 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.

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

15.1 EVENT

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

[O15-4a] 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”.

[CR15-5d] 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.

[CO15-6d] 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

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

0	15	16	31
0x42	flag	command = EVENT_CMD	
length		req_id	
reserved		first event identifier	
stream channel index for first event		block_id for first event	
timestamp for first event (most significant bits)			
timestamp for first event (least significant bits)			
…			
reserved		last event identifier	
stream channel index for last event		block_id for last event	
timestamp for last event (high part)			
timestamp for last event (low part)			

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.

[CR15-8] 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.

[O15-9a] 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.

[CR15-10d] 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.

[CR15-11d] 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.

[CO15-12d] 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

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

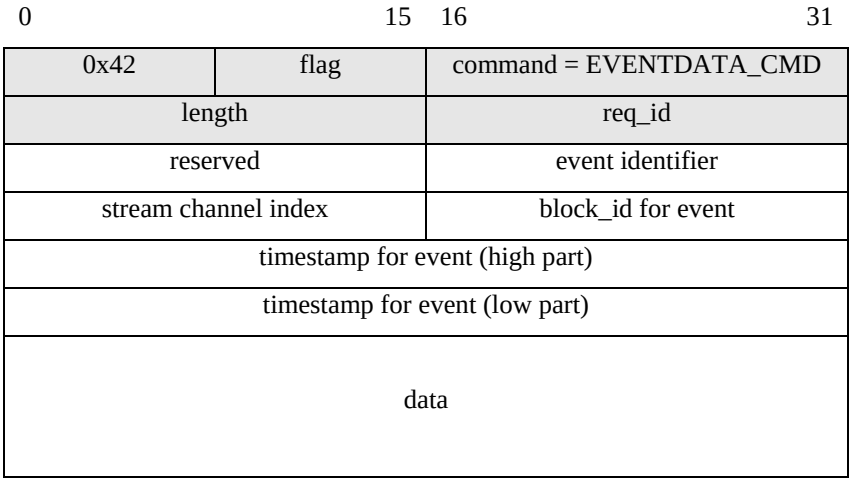


Figure 15-3: EVENTDATA_CMD message

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.

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

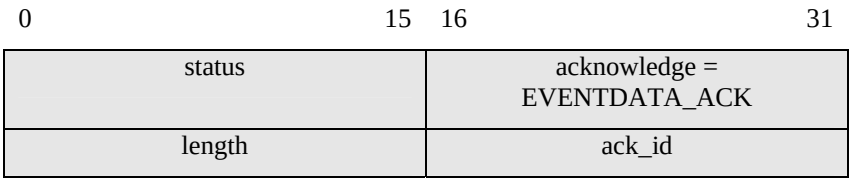


Figure 15-4: EVENTDATA_ACK message

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

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. These device-specific message should be described by the XML device description file.

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.

[R16-1] Device and application MUST use the following value to represent the various messages:

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
PACKETRESEND_ACK	R	Control	0x0041
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
Asynchronous Events			
EVENT_CMD	R	Message	0x00C0
EVENT_ACK	R	Message	0x00C1
EVENTDATA_CMD	O	Message	0x00C2
EVENTDATA_ACK	O	Message	0x00C3

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.

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

[CR17-2d] 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).

[R17-3a] 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 algorithm. The file extension MUST be .ZIP. This specification uses the DEFLATE algorithm defined by RFC1951. 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, and was later specified in RFC1951.

Note: Source code for a C language implementation of a "deflate" compliant compressor and decompressor is available within the zlib package at ftp://ftp.uu.net/pub/archiving/zip/zlib, as indicated by RFC1951l

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.

[R17-4a] 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.

[R17-5d] 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.

[O17-6a] 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.

18 Status Code

This section lists the various status codes that can be returned in an acknowledge message. 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.

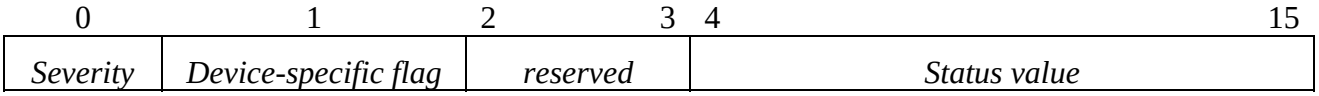
Device-specific status codes are defined in the XML description file allowing tailoring of the status field of the acknowledge packet to a particular device with a customized description of the error.

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

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

A device 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_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 request packet is not available anymore.	0x800C
GEV_STATUS_DATA_OVERRUN	Internal memory of device 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
GEV_STATUS_ERROR	Generic error. Try to avoid and use a more descriptive status code from list above.	0x8FFF

19 Events

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

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

Table 19-1: List of Events

Event Identifier	Description	Data	Value (hexadecimal)
GEV_EVENT_TRIGGER	The device has been triggered.	None	0x0002
GEV_EVENT_START_OF_EXPOSURE	Start of sensor exposure	None	0x0003
GEV_EVENT_END_OF_EXPOSURE	End of sensor exposure	None	0x0004
GEV_EVENT_START_OF_TRANSFER	Start of transfer on the stream channels	None	0x0005
GEV_EVENT_END_OF_TRANSFER	End of transfer on the stream channels	None	0x0006
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

[R20-1d] 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.

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

21 GVSP Summary

21.1 Overview

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

GVSP packets always travel from the device to the application.

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 Stream Protocol (GVSP) are:

- Define a mechanism for a GigE Vision device to send image data, image status and other data to the application.
- Support self-describing data.
- Minimize IP stack complexity in the GigE Vision device.
- 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 types.

22 GVSP Transport Protocol Considerations

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

[R22-2] Device and application 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 engine.

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.

[R22-3a] The application 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 the device to the application, it is up to the application to decide if it can support IP fragmentation. The SCPSx provides a “do not fragment” bit that is copied into the IP header of each packet of the corresponding stream channel.

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 can request them using the PACKETRESEND command (when this command is supported by the device). 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 application side to react to a missing packet is application dependent. Remember that UDP does not guaranty packets are delivered in the same order they were transmitted by the device.

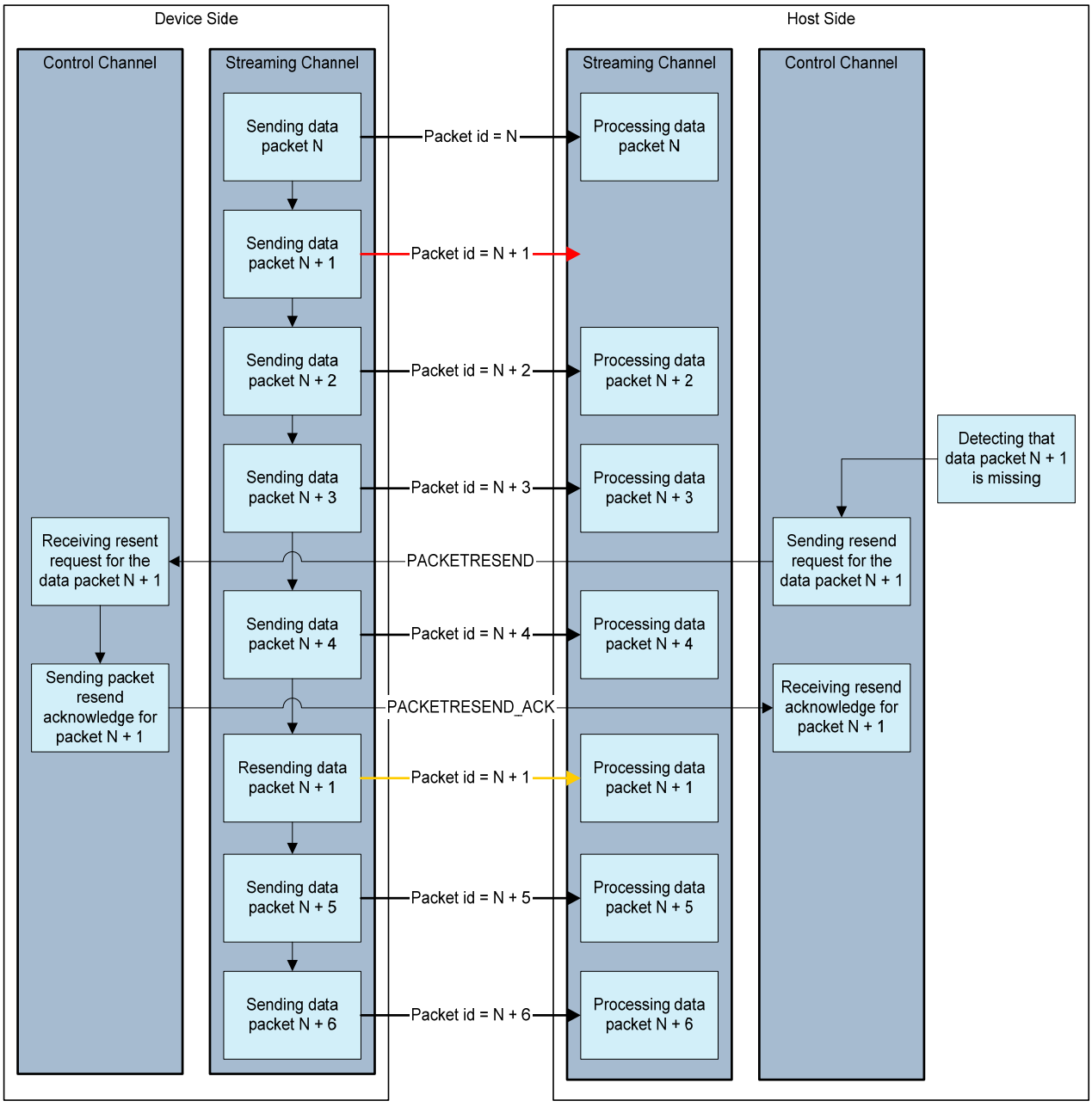


Figure 22-1: Data Resend Flowchart

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.

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. If the device is disconnected, the application will stop to receive streaming packets.

22.1.6 Device error handling during acquisition

[R22-4d] If an acquisition error occurs in the device, the device 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 device must send one data trailer for each discarded block of data, even if a single packet is lost in the acquisition section. This ensures the application will not ask for packet resend pointlessly.

For example a memory or data overrun error in the device should lead to a data trailer packet sent with a status field set to GEV_STATUS_DATA_OVERRUN. This is to ensure that the application is 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. The application would thus be able to track images by looking at the block_id associated with each data block.

[R23-1] 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.

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

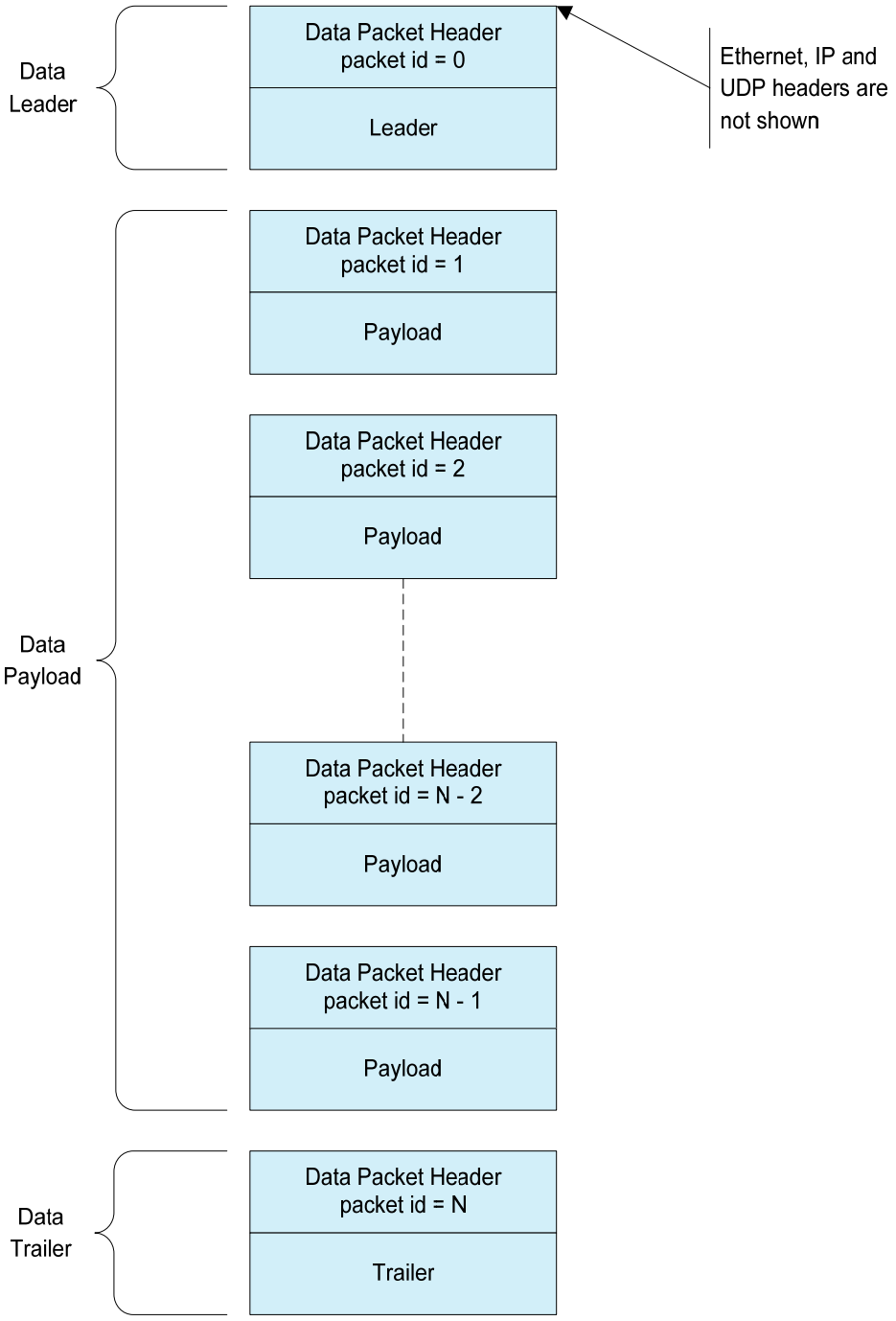


Figure 23-1: Data Block

Data blocks are sent sequentially on a given stream channel.

[R23-2d] A device 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 on in a previous data block.

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

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

23.1 Image Payload Type

[CR23-3d] 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 device 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.

[CR23-4d] If Image payload is supported, the device MUST send data packet sequentially (from 1 to N-1).

If packets are lost, an application can use the PACKETRESEND command (when supported by the device) to ask for a group of consecutive packets.

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

[CO23-6d] If Image payload is supported, when a multi-tap device 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 application to reconstruct the image (at the expense of increased CPU usage). 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 device documentation.

This specification recommends that a device 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 the device to the application. For instance, this can be used to send acquisition statistics.

[CR23-7d] 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 the application hard disk. For instance, a device could use JPEG compression to deliver compressed files to the application.

[CR23-8d] 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 in bytes

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. The number of bytes MUST be multiple of 4.
K	CID [4 Bytes]	The chunk identifier
K+4	Length [4 Bytes]	The length of the data

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

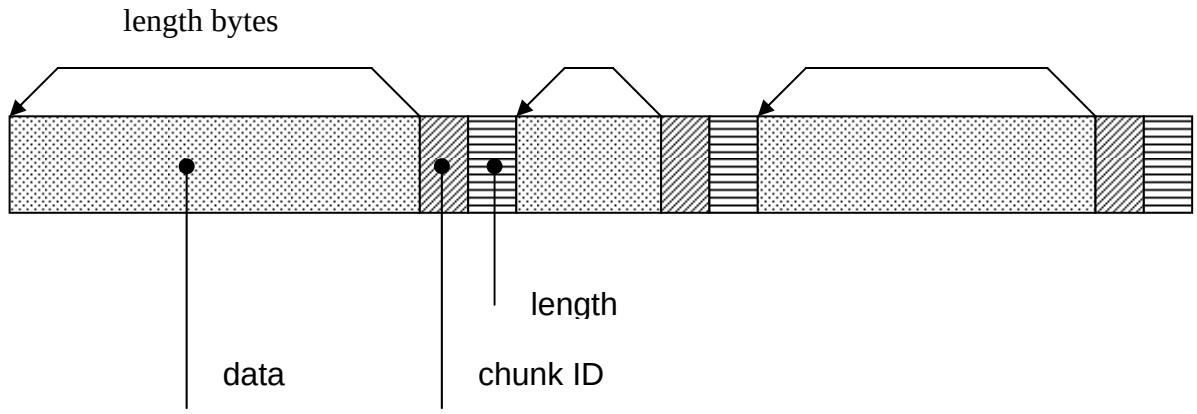


Figure 23-2: Chunk Data Diagram

[CR23-9d] 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.

The chunk parser decodes a block in chunk format beginning with the last received data walking to the beginning of the block.

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

23.5 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.

24 Data Packet Headers

[R24-1] 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: the application uses 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. The application is responsible to optimize packet size. This could be achieved using the test packet for instance. This facilitates the positioning of information by the application in the buffer if a packet is missing.

[R24-2] All GVSP headers MUST use network byte order (Big-endian).

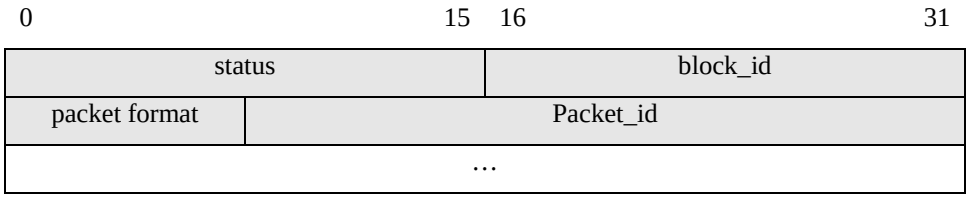


Figure 24-1: Data Packet Header

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. The data payload contains raw image data. • 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

[R24-3d] The data leader MUST be the first packet of the block.

[R24-4d] 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.

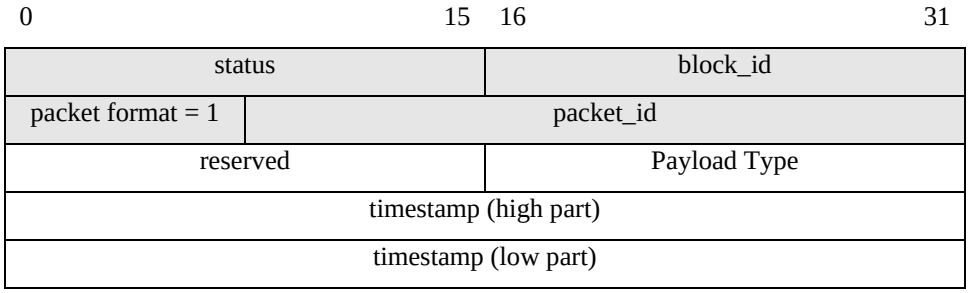


Figure 24-2: Generic Data Leader

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 • 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

[CR24-5] 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

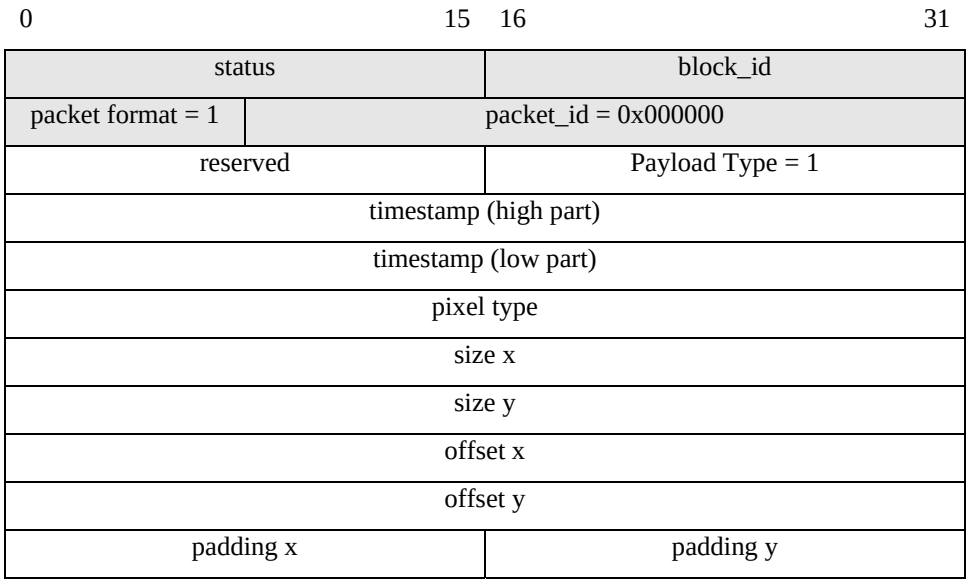


Figure 24-3: Image Data Leader

IMAGES DATA LEADER	
pixel type	This field gives the pixel format of payload data. Refer to pixel type format.
size x	Width in pixels. The actual width received can be lower than the maximum supported by the device when a ROI is set. When no ROI is defined, this field must be set to the maximum number of pixels supported by the device.
size y	Height in lines. The actual number of lines received can be lower than the maximum supported by the device when a ROI is set. When no ROI is defined, this field must be set to the maximum number of lines supported by the device.
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

[CR24-6] If Raw Data payload is supported, its Data Leader MUST follow the layout of Figure 24-4.

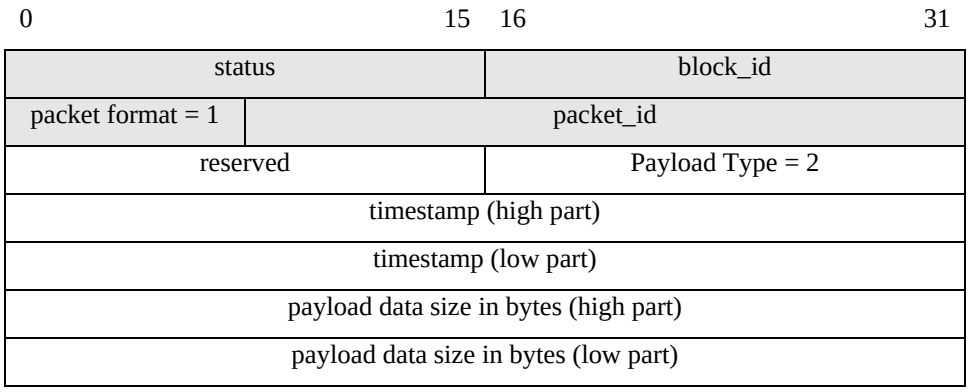


Figure 24-4: Raw Data Leader

RAW DATA LEADER	
Payload data size	Total size of payload data in bytes

24.1.3 Payload Type = File

[CR24-7] If File payload is supported, its Data Leader MUST follow the layout of Figure 24-5.

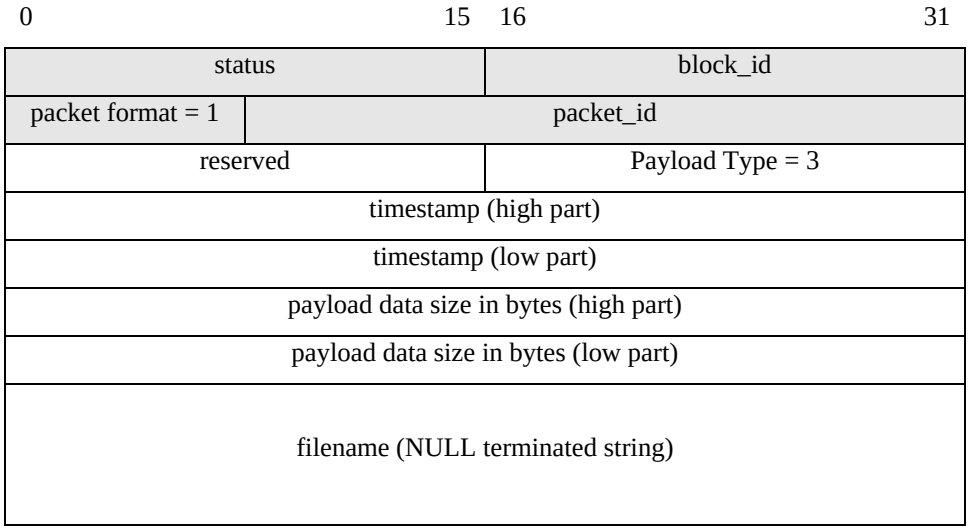


Figure 24-5: File Data Leader

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

[CR24-8] If Chunk Data payload is supported, its Data Leader MUST follow the layout of Figure 24-6

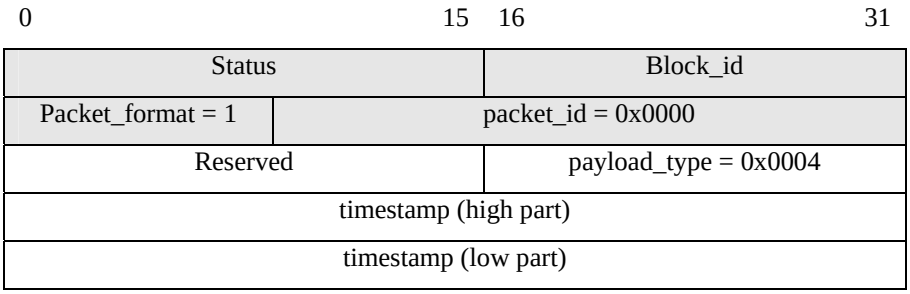


Figure 24-6: Chunk Data Leader

24.1.5 Payload Type = Device-specific

[CR24-9] If Device-specific payload is supported, its Data Leader MUST follow the layout of Figure 24-7.

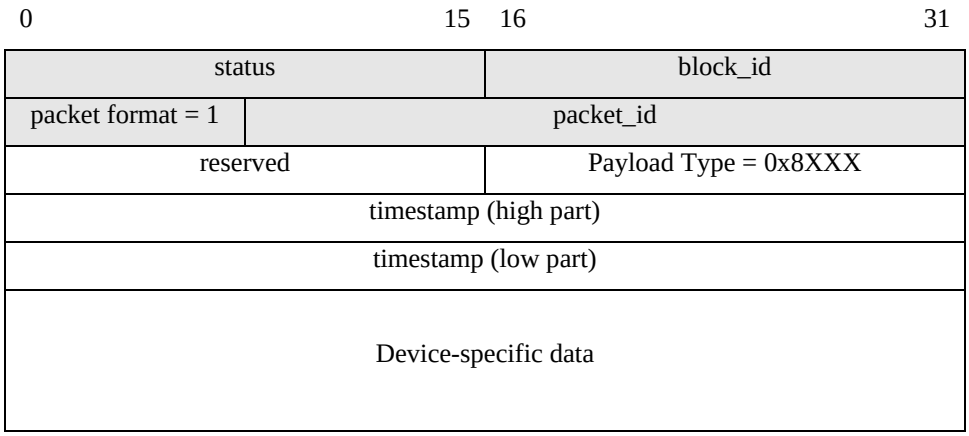


Figure 24-7: Device-specific Data Leader

24.2 Data Payload

Data payload packets transport the actual information to the application. 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.

[R24-10d] Overlapping data from multiple ROIs MUST be retransmitted with each ROI so that each image is provided completely in its block.

[CR24-11d] For camera device, 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.

[R24-12d] All data payload packets except the last one MUST use the requested packet_size.

This way, the application 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

[CR24-13] If Image payload is supported, its Data Payload MUST follow layout of Figure 24-8

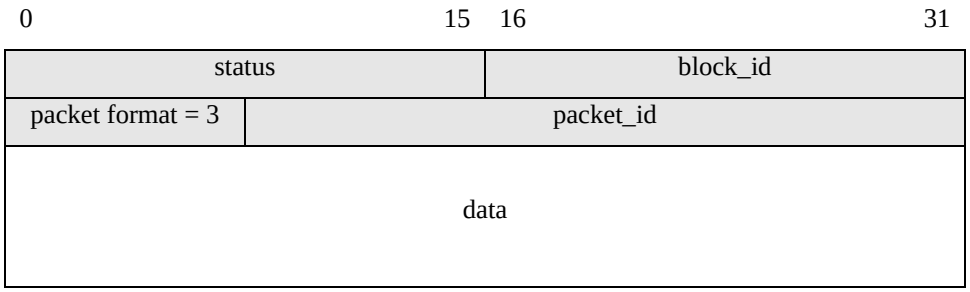


Figure 24-8: Image Data Payload

DATA PAYLOAD	
data	Image data formatted as specified in the Data Leader pixel type 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 (See GVCP). The only exception is the last Data Payload packet which may be smaller.

24.2.2 Payload Type = Raw data

[CR24-14] If Raw payload is supported, its Data Payload MUST follow layout of Figure 24-9.

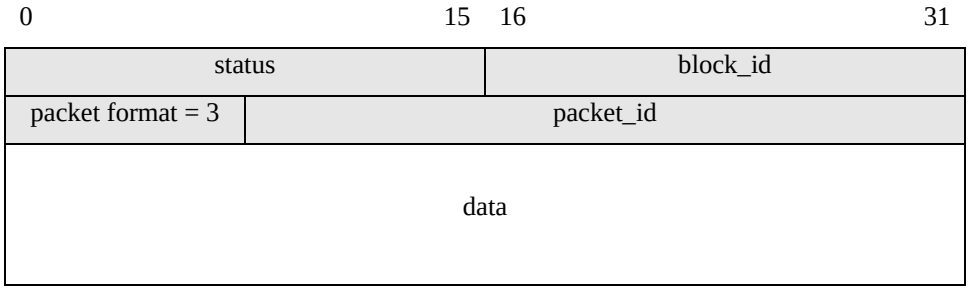


Figure 24-9: Raw Data Payload

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 (See GVCP). The only exception is the last Data Payload packet which may be smaller.

24.2.3 Payload Type = File

[CR24-15] If File payload is supported, its Data Payload MUST follow layout of Figure 24-10.

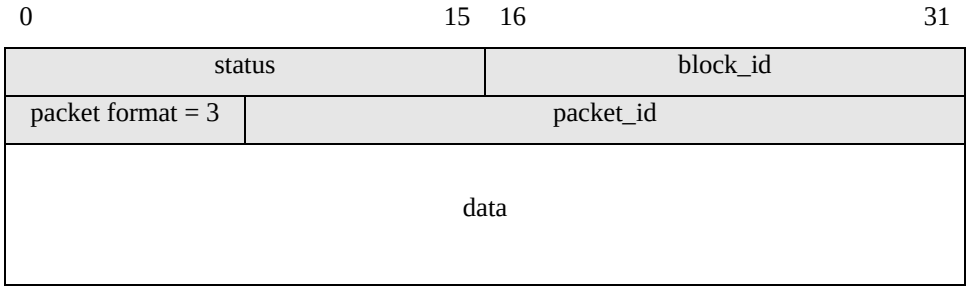


Figure 24-10: File Data Payload

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 (See GVCP). The only exception is the last Data Payload packet which may be smaller.

24.2.4 Payload Type = Chunk Data

[CR24-16] If Chunk Data payload is supported, its Data Payload MUST follow the layout of Figure 24-10

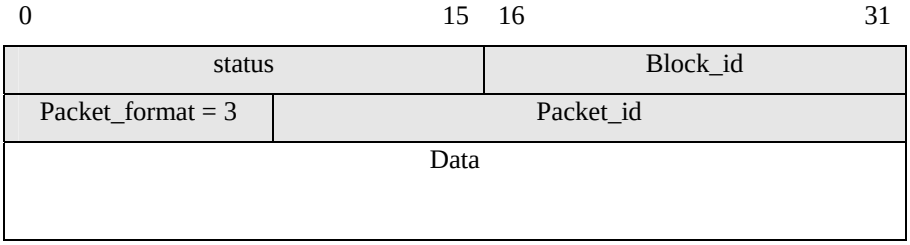


Figure 24-11: Chunk Data Payload

DATA PAYLOAD	
data	Chunk data. For 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 (See GVCP). The only exception is the last Data Payload packet which may be smaller.

24.2.5 Payload Type = Device-specific

[CR24-17] If Device-specific payload is supported, its Data Payload MUST follow the layout of Figure 24-12.

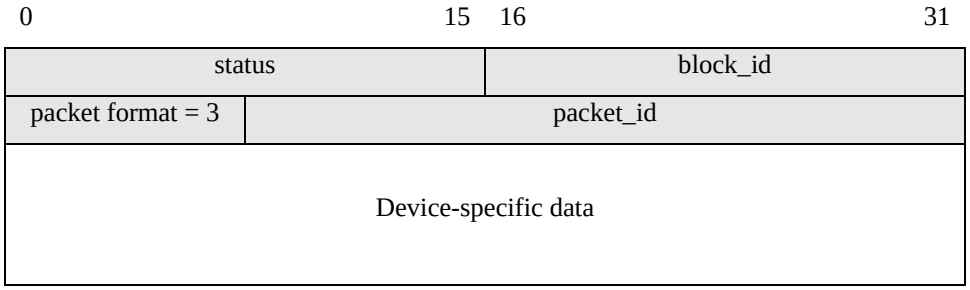


Figure 24-12: Device-specific Data Payload

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 (See GVCP). The only exception is the last Data Payload packet which may be smaller.

24.3 Data Trailer

[R24-18d] The data trailer MUST be the last packet of the block.

[R24-19d] 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).

[R24-20d] 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.

[R24-21] The Data Trailer MUST follow the layout of Figure 24-13.

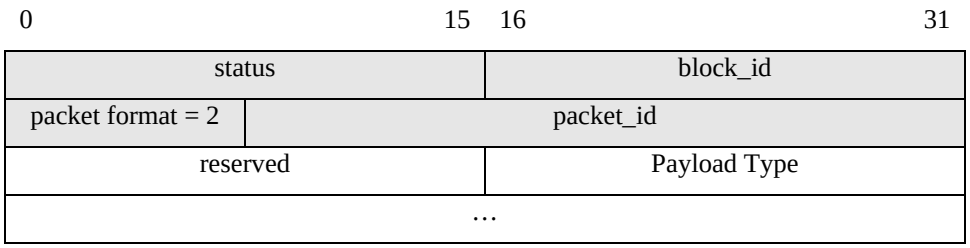


Figure 24-13: Generic Data Trailer

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 • 0x8000 and above are device-specific

24.3.1 Payload Type = Image

[CR24-22] If Image payload is supported, its Data Trailer MUST follow the layout of Figure 24-14.

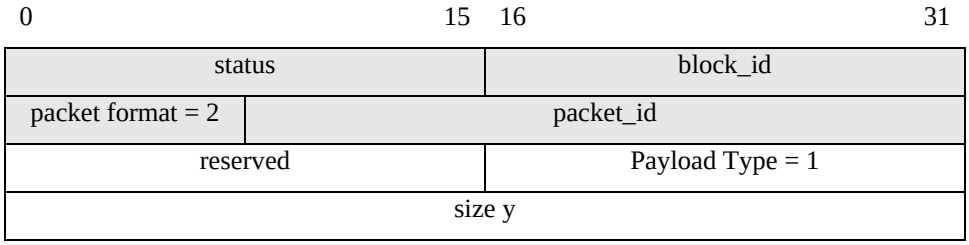


Figure 24-14: Image Data Trailer

IMAGES DATA TRAILER	
size y	The size y is put again in the data trailer. This is done to support variable frame size devices. Note that the size y specified in the data leader is the maximum supported by the device (when no ROI is defined). This field is the actual size for this particular block.

24.3.2 Payload Type = Raw data

[CR24-23] If Raw payload is supported, its Data Trailer MUST follow the layout of Figure 24-15.

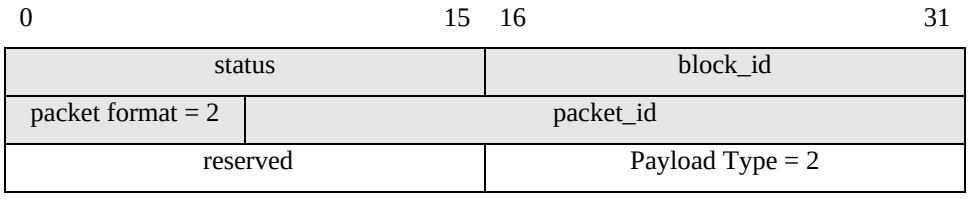


Figure 24-15: Raw Data Trailer

24.3.3 Payload Type = File

[CR24-24] If File payload is supported, its Data Trailer MUST follow the layout of Figure 24-16.

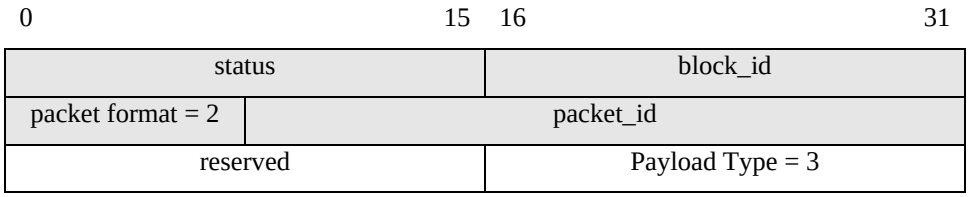


Figure 24-16: File Data Trailer

24.3.4 Payload Type = Chunk Data

[CR24-25] If Chunk Data Payload is supported, its Data Trailer MUST follow the layout of Figure 24-15

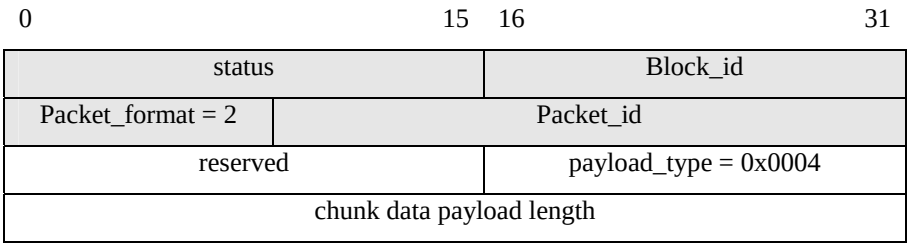


Figure 24-17: Chunk Data Trailer

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 applications finding the end of the chunk data payload

24.3.5 Payload Type = Device-specific

[CR24-26] If Device-specific payload is supported, its Data Trailer MUST follow the layout of Figure 24-18.

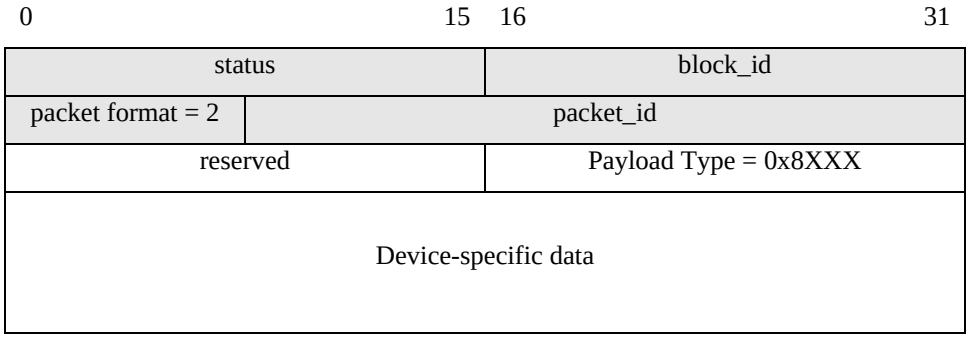


Figure 24-18: Device-specific Data Trailer

24.4 Test packet

An application can use test packets to discover the MTU of the connection. This is done on each network interfaces of the device using the stream channel registers.

[R24-27] 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) or payload is “don’t care”.

A device 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).

[R24-28] The Test Packet MUST follow the layout of Figure 24-19.

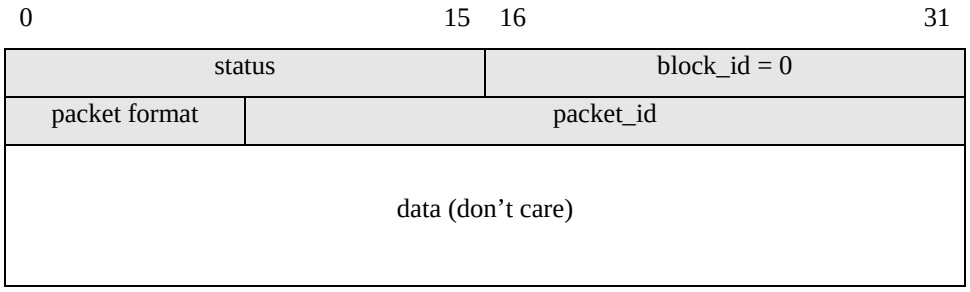


Figure 24-19: Test Packet

Test packet	
data	Filled with don’t care data. 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 (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).

25 Pixel Format

[O25-1d] Devices providing pixel depths greater than 8 bits SHOULD also provide an 8 bit transfer mode to facilitate display of images by the application. 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 Pixel Alignment

[R25-2] In order to minimize pixel processing time on the application side, pixel data using multiple-byte in the payload data packets MUST be little-endian by default.

A device may implement a multiplexer to convert pixels from little-endian to big-endian. This multiplexer can be activated using the SCPSx bootstrap register when 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.

25.1.1 Align_Mono8

Description : 8-bit monochrome unsigned
Pixel Size : 1 byte
Value range : 0 to 255

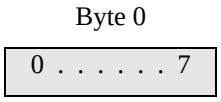


Figure 25-1: Align_Mono8

25.1.2 Align_Mono8Signed

Description : 8-bit monochrome signed
Pixel Size : 1 byte
Value Range : -128 to 127

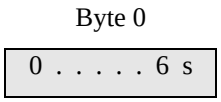


Figure 25-2: Align_Mono8Signed

25.1.3 Align_Mono10

Description : 10-bit monochrome
Pixel Size : 2 bytes

Value range : 0 to 1023

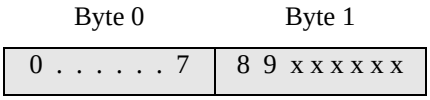


Figure 25-3: Align_Mono10

25.1.4 Align_Mono10Packed

Description : 10-bit packed monochrome unsigned
Pixel Size : 3 bytes for two pixels
Value range : 0 to 1023

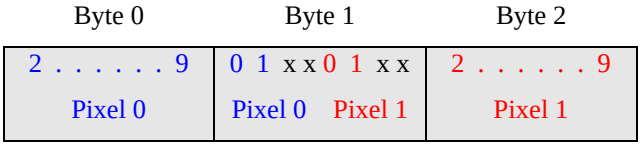


Figure 25-4: Align_Mono10Packed

25.1.5 Align_Mono12

Description : 12-bit monochrome unsigned
Pixel Size : 2 bytes
Value range : 0 to 4095

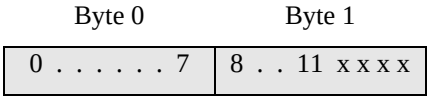


Figure 25-5: Align_Mono12

25.1.6 Align_Mono12Packed

Description : 12-bit packed monochrome unsigned
Pixel Size : 3 bytes for two pixels
Value range : 0 to 1023

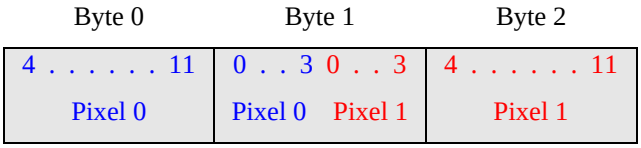


Figure 25-6: Align_Mono12Packed

25.1.7 Align_Mono16

Description : 16-bit monochrome unsigned
Pixel Size : 2 bytes
Value range : 0 to 65535

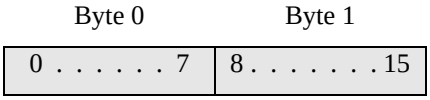


Figure 25-7: Align_Mono16

25.1.8 Align_RGB10Packed_V1

Description : 10-bit RGB packed unsigned version 1
Pixel Size : 4 bytes
Value range : 0 to 1023

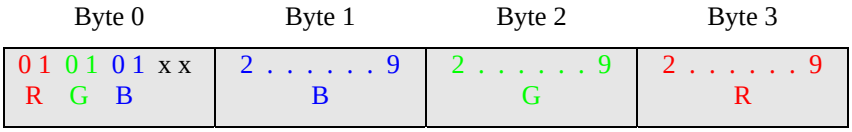


Figure 25-8: Align_RGB10Packed_V1

25.1.9 Align_RGB10Packed_V2

Description : 10-bit RGB packed unsigned version 2
Pixel Size : 4 bytes
Value range : 0 to 1023

Figure 25-9: Align_RGB10Packed_V2

25.2 Pixel Types

This section illustrates the various pixel types natively supported by GVSP.

[O25-3d] A device SHOULD support a subset of the pixel types listed in this section.

Note: Device-specific pixel types are supported by setting bit 31 of the pixel type value. They should be described by the XML device description file.

25.2.1 GVSP_PIX_MONO8

[CR25-4] If supported, GVSP_PIX_MONO8 MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono8	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x00000001
Pixel Type:	**GVSP_PIX_MONO8**	0x01080001

Y0	Y1	Y2	Y3	Y4
Y5	Y6	Y7	Y8	Y9

25.2.2 GVSP_PIX_MONO8_SIGNED

[CR25-5] If supported, GVSP_PIX_MONO8_SIGNED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono8Signed	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x00000002
Pixel Type:	**GVSP_PIX_MONO8 SIGNED**	0x01080002

Y0	Y1	Y2	Y3	Y4
Y5	Y6	Y7	Y8	Y9

25.2.3 GVSP_PIX_MONO10

[CR25-6] If supported, GVSP_PIX_MONO10 MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono10	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000003
Pixel Type:	**GVSP_PIX_MONO10**	0x01100003

Y0	Y1	Y2	Y3	Y4
Y5	Y6	Y7	Y8	Y9

25.2.4 GVSP_PIX_MONO10_PACKED

[CR25-7] If supported, GVSP_PIX_MONO10_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono10Packed	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x00000004
Pixel Type:	**GVSP_PIX_MONO10_PACKED**	0x010C0004

Byte 0	Byte 1	Byte 2	Byte 3
Y0/Y1			Y2/Y3
Y2/Y3	Y4/Y5		Y6/Y7
Y6/Y7		Y8/Y9	

25.2.5 GVSP_PIX_MONO12

[CR25-8] If supported, GVSP_PIX_MONO12 MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono12	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000005
Pixel Type:	**GVSP_PIX_MONO12**	0x01100005

Y0	Y1	Y2	Y3	Y4
Y5	Y6	Y7	Y8	Y9

25.2.6 GVSP_PIX_MONO12_PACKED

[CR25-9] If supported, GVSP_PIX_MONO12_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono12Packed	
Color/Mono	MONO	0x01000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x00000006
Pixel Type:	**GVSP_PIX_MONO12_PACKED**	0x010C0006

Byte 0	Byte 1	Byte 2	Byte 3
Y0/Y1			Y2/Y3
Y2/Y3	Y4/Y5		Y6/Y7
Y6/Y7		Y8/Y9	

25.2.7 GVSP_PIX_MONO16

[CR25-10] If supported, GVSP_PIX_MONO16 MUST follow the layout below.

	Description	Value
Pixel alignment	Y = Align_Mono16	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000007
Pixel Type:	**GVSP_PIX_MONO16**	0x01100007

Y0	Y1	Y2	Y3	Y4
Y5	Y6	Y7	Y8	Y9

25.2.8 GVSP_PIX_BAYGR8

[CR25-11d] All Bayer pixel formats MUST represent the Bayer pattern of the sensor.

[CR25-12] If supported, GVSP_PIX_BAYGR8 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x00000008
Pixel Type:	**GVSP_PIX_BAYGR8**	0x01080008

Odd Lines:

G0	R1	G2	R3	G4
G10	R11	G12	R13	G14

Even Lines:

B5	G6	B7	G8	B9
B15	G16	B17	G18	B19

25.2.9 GVSP_PIX_BAYRG8

[CR25-13] If supported, GVSP_PIX_BAYRG8 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x00000009
Pixel Type:	**GVSP_PIX_BAYRG8**	0x01080009

Odd Lines:

R0	G1	R2	G3	R4
R10	G11	R12	G13	R14

Even Lines:

G5	B6	G7	B8	G9
G15	B16	G17	B18	G19

25.2.10 GVSP_PIX_BAYGB8

[CR25-14] If supported, GVSP_PIX_BAYGB8 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x0000000A
Pixel Type:	**GVSP_PIX_BAYGB8**	0x0108000A

Odd Lines:

G0	B1	G2	B3	G4
G10	B11	G12	B13	G14

Even Lines:

R5	G6	R7	G8	R9
R15	G16	R17	G18	R19

25.2.11 GVSP_PIX_BAYBG8

[CR25-15] If supported, GVSP_PIX_BAYBG8 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Color/Mono	MONO	0x01000000
Bits per pixel	8 Bit	0x00080000
Pixel ID	Id of pixel	0x0000000B
Pixel Type:	**GVSP_PIX_BAYBG8**	0x0108000B

Odd Lines:

B0	G1	B2	G3	B4
B10	G11	B12	G13	B14

Even Lines:

R5	G6	R7	G8	R9
R15	G16	R17	G18	R19

25.2.12 GVSP_PIX_BAYGR10

[CR25-16] If supported, GVSP_PIX_BAYGR10 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000000C
Pixel Type:	**GVSP_PIX_BAYGR10**	0x0110000C

Odd Lines:

G0	R1	G2	R3	G4
G10	R11	G12	R13	G14

Even Lines:

B5	G6	B7	G8	B9
B15	G16	B17	G18	B19

25.2.13 GVSP_PIX_BAYRG10

[CR25-17] If supported, GVSP_PIX_BAYRG10 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000000D
Pixel Type:	**GVSP_PIX_BAYRG10**	0x0110000D

Odd Lines:

R0	G1	R2	G3	R4
R10	G11	R12	G13	R14

Even Lines:

G5	B6	G7	B8	G9
G15	B16	G17	B18	G19

25.2.14 GVSP_PIX_BAYGB10

[CR25-18] If supported, GVSP_PIX_BAYGB10 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000000E
Pixel Type:	**GVSP_PIX_BAYGB10**	0x0110000E

Odd Lines:

G0	B1	G2	B3	G4
G10	B11	G12	B13	G14

Even Lines:

R5	G6	R7	G8	R9
R15	G16	R17	G18	R19

25.2.15 GVSP_PIX_BAYBG10

[CR25-19] If supported, GVSP_PIX_BAYBG10 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000000F
Pixel Type:	**GVSP_PIX_BAYBG10**	0x0110000F

Odd Lines:

B0	G1	B2	G3	B4
B10	G11	B12	G13	B14

Even Lines:

R5	G6	R7	G8	R9
R15	G16	R17	G18	R19

25.2.16 GVSP_PIX_BAYGR12

[CR25-20] If supported, GVSP_PIX_BAYGR12 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000010
Pixel Type:	**GVSP_PIX_BAYGR12**	0x01100010

Odd Lines:

G0	R1	G2	R3	G4
G10	R11	G12	R13	G14

Even Lines:

B5	G6	B7	G8	B9
B15	G16	B17	G18	B19

25.2.17 GVSP_PIX_BAYRG12

[CR25-21] If supported, GVSP_PIX_BAYRG12 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000011
Pixel Type:	**GVSP_PIX_BAYRG12**	0x01100011

Odd Lines:

R0	G1	R2	G3	R4
R10	G11	R12	G13	R14

Even Lines:

G5	B6	G7	B8	G9
G15	B16	G17	B18	G19

25.2.18 GVSP_PIX_BAYGB12

[CR25-22] If supported, GVSP_PIX_BAYGB12 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000012
Pixel Type:	**GVSP_PIX_BAYGB12**	0x01100012

Odd Lines:

G0	B1	G2	B3	G4
G10	B11	G12	B13	G14

Even Lines:

R5	G6	R7	G8	R9
R15	G16	R17	G18	R19

25.2.19 GVSP_PIX_BAYBG12

[CR25-23] If supported, GVSP_PIX_BAYBG12 MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Color/Mono	MONO	0x01000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x00000013
Pixel Type:	**GVSP_PIX_BAYBG12**	0x01100013

Odd Lines:

B0	G1	B2	G3	B4
B10	G11	B12	G13	B14

Even Lines:

R5	G6	R7	G8	R9
R15	G16	R17	G18	R19

25.2.20 GVSP_PIX_RGB8_PACKED

[CR25-24] If supported, GVSP_PIX_RGB8_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Color/Mono	COLOR	0x02000000
Bits per pixel	24 Bit	0x00180000
Pixel ID	Id of pixel	0x00000014
Pixel Type:	**GVSP_PIX_RGB8_PACKED**	0x02180014

R0	G0	B0	R1	G1
B1	R2	G2	B2	R3

25.2.21 GVSP_PIX_BGR8_PACKED

[CR25-25] If supported, GVSP_PIX_BGR8_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	B = G = R = Align_Mono8	
Color/Mono	COLOR	0x02000000
Bits per pixel	24 Bit	0x00180000
Pixel ID	Id of pixel	0x00000015
Pixel Type:	**GVSP_PIX_BGR8_PACKED**	0x02180015

B0	G0	R0	B1	G1
R1	B2	G2	R2	B3

25.2.22 GVSP_PIX_RGBA8_PACKED

[CR25-26] If supported, GVSP_PIX_RGBA8_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = a = Align_Mono8	
Color/Mono	COLOR	0x02000000
Bits per pixel	32 Bit	0x00200000
Pixel ID	Id of pixel	0x00000016
Pixel Type:	**GVSP_PIX_RGBA8_PACKED**	0x02200016

R0	G0	B0	a0	R1
G1	B1	a1	R2	G2

25.2.23 GVSP_PIX_BGRA8_PACKED

[CR25-27] If supported, GVSP_PIX_BGRA8_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	B = G = R = a = Align_Mono8	
Color/Mono	COLOR	0x02000000
Bits per pixel	32 Bit	0x00200000
Pixel ID	Id of pixel	0x00000017
Pixel Type:	**GVSP_PIX_BGRA8_PACKED**	0x02200017

B0	G0	R0	a0	B1
G1	R1	a1	B2	G2

25.2.24 GVSP_PIX_RGB10_PACKED

[CR25-28] If supported, GVSP_PIX_RGB10_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000018
Pixel Type:	**GVSP_PIX_RGB10_PACKED**	0x02300018

R0	G0	B0	R1	G1
B1	R2	G2	B2	R3

25.2.25 GVSP_PIX_BGR10_PACKED

[CR25-29] If supported, GVSP_PIX_BGR10_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	B = G = R = Align_Mono10	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000019
Pixel Type:	**GVSP_PIX_BGR10_PACKED**	0x02300019

B0	G0	R0	B1	G1
R1	B2	G2	R2	B3

25.2.26 GVSP_PIX_RGB12_PACKED

[CR25-30] If supported, GVSP_PIX_RGB12_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x0000001A
Pixel Type:	**GVSP_PIX_RGB12_PACKED**	0x0230001A

R0	G0	B0	R1	G1
B1	R2	G2	B2	R3

25.2.27 GVSP_PIX_BGR12_PACKED

[CR25-31] If supported, GVSP_PIX_BGR12_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	B = G = R = Align_Mono12	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x0000001B
Pixel Type:	**GVSP_PIX_BGR12_PACKED**	0x0230001B

B0	G0	R0	B1	G1
R1	B2	G2	R2	B3

25.2.28 GVSP_PIX_BGR10V1_PACKED

[CR25-32] If supported, GVSP_PIX_BGR10V1_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	RGB = Align_RGB10Packed_V1	
Color/Mono	COLOR	0x02000000
Bits per pixel	32 Bit	0x00200000
Pixel ID	Id of pixel	0x0000001C
Pixel Type:	**GVSP_PIX_BGR10V1_PACKED**	0x0220001C

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RGB0				RGB1			
RGB2				RGB3			

25.2.29 GVSP_PIX_BGR10V2_PACKED

[CR25-33] If supported, GVSP_PIX_BGR10V2_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	RGB = Align_RGB10Packed_V2	
Color/Mono	COLOR	0x02000000
Bits per pixel	32 Bit	0x00200000
Pixel ID	Id of pixel	0x0000001D
Pixel Type:	**GVSP_PIX_BGR10V2_PACKED**	0x0220001D

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RGB0				RGB1			
RGB2				RGB3			

25.2.30 GVSP_PIX_YUV411_PACKED

[CR25-34] If supported, GVSP_PIX_YUV411_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = U = V = Align_Mono8	
Color/Mono	COLOR	0x02000000
Bits per pixel	12 Bit	0x000C0000
Pixel ID	Id of pixel	0x0000001E
Pixel Type:	**GVSP_PIX_YUV411_PACKED**	0x020C001E

U0	Y0	Y1	V0	Y2
Y3	U4	Y4	Y5	V4

25.2.31 GVSP_PIX_YUV422_PACKED

[CR25-35] If supported, GVSP_PIX_YUV422_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = U = V = Align_Mono8	
Color/Mono	COLOR	0x02000000
Bits per pixel	16 Bit	0x00100000
Pixel ID	Id of pixel	0x0000001F
Pixel Type:	**GVSP_PIX_YUV422_PACKED**	0x0210001F

U0	Y0	V0	V1	U2
Y2	V2	Y3	U4	Y4

25.2.32 GVSP_PIX_YUV444_PACKED

[CR25-36] If supported, GVSP_PIX_YUV444_PACKED MUST follow the layout below.

	Description	Value
Pixel alignment	Y = U = V = Align_Mono8	
Color/Mono	COLOR	0x02000000
Bits per pixel	24 Bit	0x00180000
Pixel ID	Id of pixel	0x00000020
Pixel Type:	**GVSP_PIX_YUV444_PACKED**	0x02180020

U0	Y0	V0	U1	Y1
V1	U2	Y2	V2	U3

25.2.33 GVSP_PIX_RGB8_PLANAR

[CR25-37] If supported, GVSP_PIX_RGB8_PLANAR MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono8	
Color/Mono	COLOR	0x02000000
Bits per pixel	24 Bit	0x00180000
Pixel ID	Id of pixel	0x00000021
Pixel Type:	**GVSP_PIX_RGB8_PLANAR**	0x02180021

Red plane (stream channel 0):

R0	R1	R2	R3	R4

Green plane (stream channel 1):

G0	G1	G2	G3	G4

Blue plane (stream channel 2):

B0	B1	B2	B3	B4

25.2.34 GVSP_PIX_RGB10_PLANAR

[CR25-38] If supported, GVSP_PIX_RGB10_PLANAR MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono10	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000022
Pixel Type:	**GVSP_PIX_RGB10_PLANAR**	0x02300022

Red plane (stream channel 0):

R0	R1	R2	R3	R4

Green plane (stream channel 1):

G0	G1	G2	G3	G4

Blue plane (stream channel 2):

B0	B1	B2	B3	B4

25.2.35 GVSP_PIX_RGB12_PLANAR

[CR25-39] If supported, GVSP_PIX_RGB12_PLANAR MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono12	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000023
Pixel Type:	**GVSP_PIX_RGB12_PLANAR**	0x02300023

Red plane (stream channel 0):

R0	R1	R2	R3	R4

Green plane (stream channel 1):

G0	G1	G2	G3	G4

Blue plane (stream channel 2):

B0	B1	B2	B3	B4

25.2.36 GVSP_PIX_RGB16_PLANAR

[CR25-40] If supported, GVSP_PIX_RGB16_PLANAR MUST follow the layout below.

	Description	Value
Pixel alignment	R = G = B = Align_Mono16	
Color/Mono	COLOR	0x02000000
Bits per pixel	48 Bit	0x00300000
Pixel ID	Id of pixel	0x00000024
Pixel Type:	**GVSP_PIX_RGB16_PLANAR**	0x02300024

Red plane (stream channel 0):

R0	R1	R2	R3	R4

Green plane (stream channel 1):

G0	G1	G2	G3	G4

Blue plane (stream channel 2):

B0	B1	B2	B3	B4

26 Pixel Type Defines

The following provides #define for the various pixel types supported by GVSP. Each pixel type 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 type.

```
//===============================================
// PIXEL TYPES
//===============================================
// Indicate if pixel is monochrome or RGB
#define GVSP_PIX_MONO                           0x01000000
#define GVSP_PIX_RGB                            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 type
#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_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)
#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)
```

26.3 RGB Packed buffer format defines

```
#define GVSP_PIX_RGB8_PACKED      (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY24BIT  | 0x0014)
#define GVSP_PIX_BGR8_PACKED      (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY24BIT  | 0x0015)
#define GVSP_PIX_RGBA8_PACKED     (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY32BIT  | 0x0016)
#define GVSP_PIX_BGRA8_PACKED     (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY32BIT  | 0x0017)
#define GVSP_PIX_RGB10_PACKED     (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY48BIT  | 0x0018)
#define GVSP_PIX_BGR10_PACKED     (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY48BIT  | 0x0019)
#define GVSP_PIX_RGB12_PACKED     (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY48BIT  | 0x001A)
#define GVSP_PIX_BGR12_PACKED     (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY48BIT  | 0x001B)
#define GVSP_PIX_RGB10V1_PACKED   (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY32BIT  | 0x001C)
#define GVSP_PIX_RGB10V2_PACKED   (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY32BIT  | 0x001D)
```

26.4 YUV Packed buffer format defines

```
#define GVSP_PIX_YUV411_PACKED    (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY12BIT  | 0x001E)
#define GVSP_PIX_YUV422_PACKED    (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY16BIT  | 0x001F)
#define GVSP_PIX_YUV444_PACKED    (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY24BIT  | 0x0020)
```

26.5 RGB Planar buffer format defines

```
#define GVSP_PIX_RGB8_PLANAR      (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY24BIT  | 0x0021)
#define GVSP_PIX_RGB10_PLANAR     (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY48BIT  | 0x0022)
#define GVSP_PIX_RGB12_PLANAR     (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY48BIT  | 0x0023)
#define GVSP_PIX_RGB16_PLANAR     (GVSP_PIX_RGB  | GVSP_PIX_OCCUPY48BIT  | 0x0024)
```

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

PART 4 – Bootstrap Registers

27 Bootstrap Registers

[R27-1d] 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.

Device registers are accessed using GVCP. Device-specific registers can start after bootstrap registers memory-space.

[R27-2d] All device registers MUST be 32-bit aligned.

[O27-3a] 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.

[R27-4a] 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.

[R27-5d] 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.

[R27-6d] All unused bits of bootstrap registers MUST be set to 0.

[O27-7d] Trying to access an unsupported register SHOULD return GEV_STATUS_INVALID_ADDRESS.

[O27-8d] Trying to read from a write-only register or to write to a read-only register SHOULD return GEV_STATUS_ACCESS_DENIED.

[O27-9a] 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.
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.

Address	Name	Support	Type	Length (bytes)	Description
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.
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	-	-	-	

Address	Name	Support	Type	Length (bytes)	Description
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	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.
0x06E0	Reserved	-	-	-	
0x06EC	Persistent default gateway (Network interface #1)	O	R/W	4	Only available if Persistent IP is supported by the device.
0x06F0	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

Address	Name	Support	Type	Length (bytes)	Description
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	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.
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	-	-	-	

Address	Name	Support	Type	Length (bytes)	Description
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	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 1 to 512.
0x0908	Reserved	-	-	-	
0x0934	GVCP Capability	M	R	4	This is a capability register indicating which one of the non-mandatory GVCP commands 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	Reserved		-		
0x0A00	CCP	M	R/W	4	Control Channel Privilege register.
0x0A04	Reserved				
0x0B00	MCP	O	R/W	4	Message Channel Port register.

Address	Name	Support	Type	Length (bytes)	Description
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
0x0B1C	Reserved		-	-	
0x0D00	SCP0	M	R/W	4	First Stream Channel Port register.
0x0D04	SCPS0	M	R/W	4	First Stream Channel Packet Size register.
0x0D08	SCPD0	M	R/W	4	First Stream Channel Packet Delay register.
0x0D0C	Reserved	-	-	-	
0x0D18	SCDA0	M	R/W	4	First Stream Channel Destination Address register.
0x0D1C	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	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	Reserved		-	-	
0xA000	Start of manufacturer-specific register space	-	-	-	Start of device-specific registers. These are not covered by the specification.

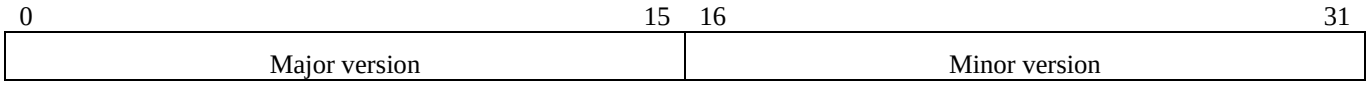
v1.0 195569

27.1 Version Register

This register indicates the version of the GigE Vision specification implemented by this device. Version 1.0 of this specification shall return 0x00010000.

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	0x00010000

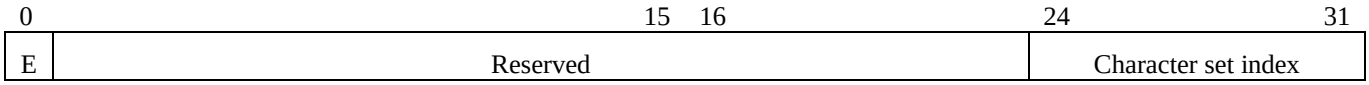


Bits	Name	Description
0 – 15	Major version	This field represents the major version of the specification. For instance, GigE Vision version 1.0 would have the major version set to 1.
16 – 31	Minor version	This field represents the minor version of the specification. For instance, GigE Vision version 1.0 would have the minor version set to 0.

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.

Address	0x0004
Length	4 bytes
Access type	Read
Factory Default	0x00000001



Bits	Name	Description
0	Endianess	Endianess might be used to interpret multi-byte data for READMEM and WRITEMEM commands. This represents the endianess of bootstrap 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 – 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

This register stores the MAC address of the given network interface.

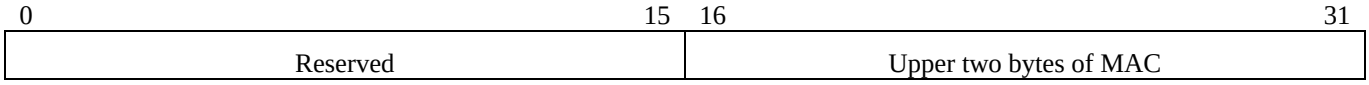
This register is mandatory on all supported interfaces.

[R27-10d] The device MUST support Interface #0 for the Device MAC register. Device must return an error for unsupported interfaces.

An application need to access the high part before accessing the low part.

27.3.1 High Part

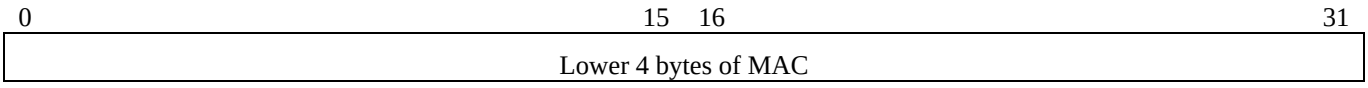
Address	0x0008 (interface #0) 0x0680 (interface #1) *[optional]* 0x0700 (interface #2) *[optional]* 0x0780 (interface #3) *[optional]*
Length	4 bytes
Access type	Read
Factory Default	Device-specific



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

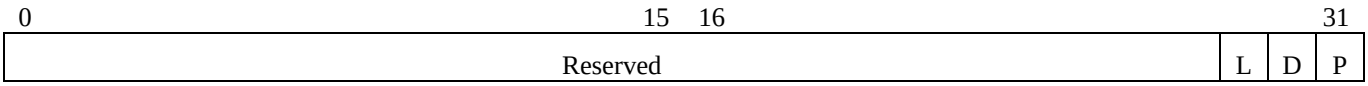
27.4 Supported IP Configuration Registers

This register indicates the IP configuration scheme supported on the given network interface. Multiple schemes can be supported simultaneously.

This register is mandatory on all supported interfaces.

[R27-11d] 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.

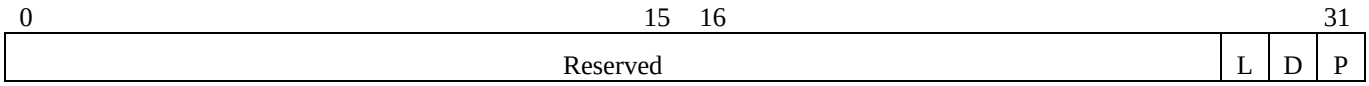
27.5 Current IP Configuration Registers

This register indicates which IP configurations schemes are currently activated on the given network interface.

This register is mandatory on all supported interfaces.

[R27-12d] 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.

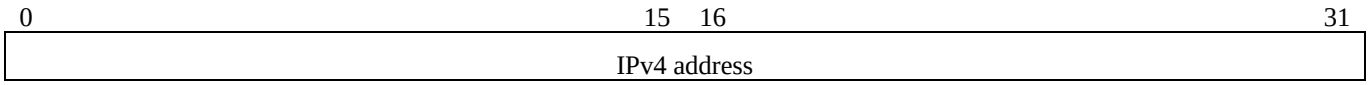
27.6 Current IP Address Registers

This register reports the IP address for the given network interface once it has been configured.

This register is mandatory on all supported interfaces.

[R27-13d] 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

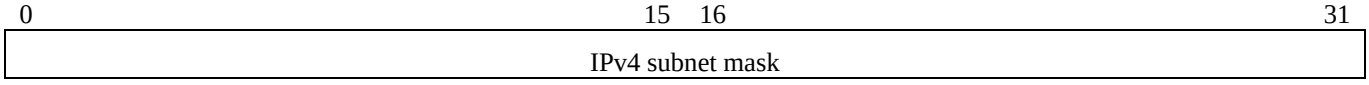
27.7 Current Subnet Mask Registers

This register provides the subnet mask of the given interface.

This register is mandatory on all supported interfaces.

[R27-14d] 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

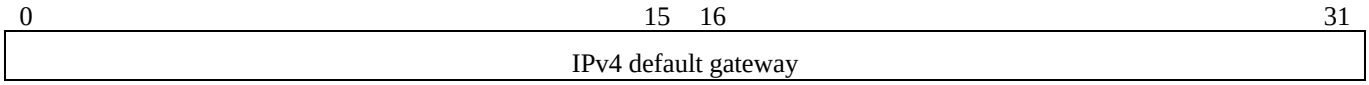
27.8 Current Default Gateway Registers

This register indicates the default gateway IP address to be used on the given network interface.

This register is mandatory on all supported interfaces.

[R27-15d] 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 registers 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 registers 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.

[CO27-16d] 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
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.

[CO27-17d] 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.

[CR27-18d] 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
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.

[R27-19d] 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.

[R27-20d] 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.

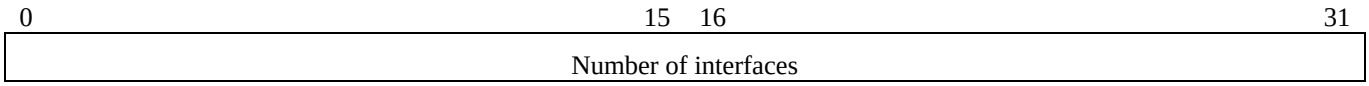
27.17 Number of Network Interfaces Register

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



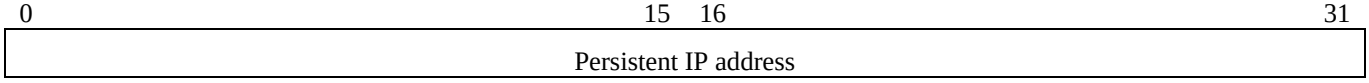
Bits	Name	Description
0 – 31	Number of interfaces	A value from 1 to 4. All other values are invalid.

27.18 Persistent IP Address Registers

This register indicates the Persistent IP address for this network interface. It is only used when the device boots with the Persistent IP configuration scheme.

[CR27-21d] 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



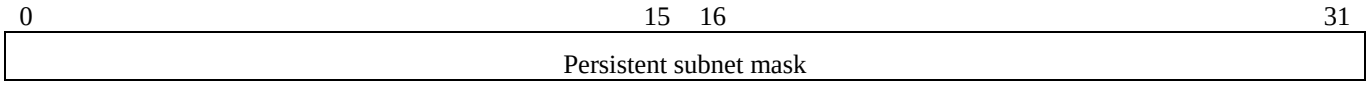
Bits	Name	Description
0 – 31	Persistent IP address	IPv4 persistent IP address for this network interface

27.19 Persistent Subnet Mask Registers

This register indicates the Persistent subnet mask associated with the Persistent IP address on this network interface. It is only used when the device boots with the Persistent IP configuration scheme.

[CR27-22d] 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



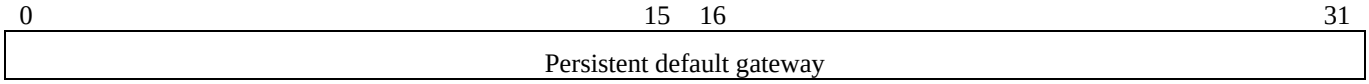
Bits	Name	Description
0 – 31	Persistent subnet mask	IPv4 persistent subnet mask for this network interface

27.20 Persistent Default Gateway Registers

This register indicates the persistent default gateway for this network interface. It is only used when the device boots with the Persistent IP configuration scheme.

[CR27-23d] 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



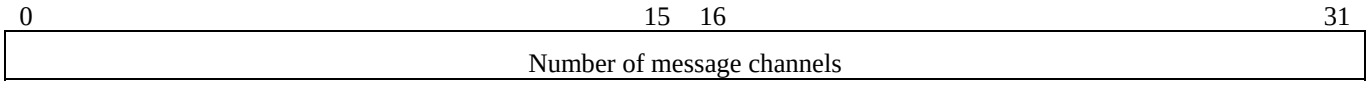
Bits	Name	Description
0 – 31	Persistent default gateway	IPv4 persistent default gateway for this network interface

27.21 Number of Message Channels Register

This register reports the number of message channel 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



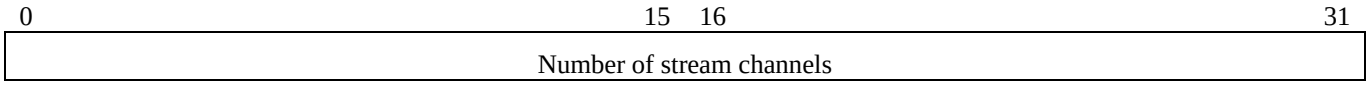
Bits	Name	Description
0 – 31	Nb Message channels	Number of message channels supported by this device. Can be 0 or 1.

27.22 Number of Stream Channels Register

This register reports the number of stream channels supported by this device.

A device needs to support at least one stream channel. A device can support up to 512 stream channels.

Address	0x0904
Length	4 bytes
Access type	Read
Factory Default	Device-specific

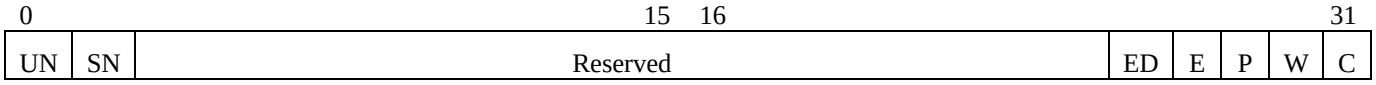


Bits	Name	Description
0 – 31	Nb Stream channels	Number of stream channels supported by this device. A value from 1 to 512.

27.23 GVCP Capability Register

This register reports the optional GVCP commands and bootstrap registers supported by this device.

Address	0x0934
Length	4 bytes
Access type	Read
Factory Default	Device-specific



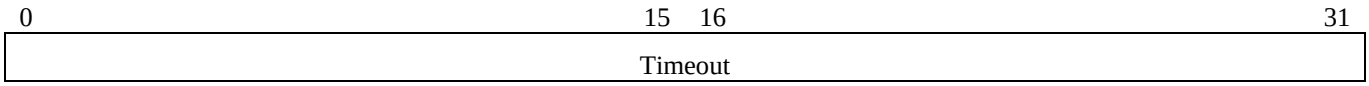
Bits	Name	Description
0	User-defined Name	User-defined name register is supported.
1	Serial Number	Serial number register is supported.
2 – 26	Reserved	Always 0.
27	EVENTDATA	EVENTDATA_CMD and EVENTDATA_ACK are supported.
28	EVENT	EVENT_CMD and EVENT_ACK are supported.
29	PACKETRESEND	PACKETRESEND_CMD is supported.
30	WRITEMEM	WRITEMEM_CMD and WRITEMEM_ACK are supported.
31	Concatenation	Multiple operations in a single message are supported.

27.24 Heartbeat Timeout Register

This register indicates the current heartbeat timeout in milliseconds.

[O27-24d] 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 msec. (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.

27.25 Timestamp Tick Frequency Register

This register indicates 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.

Note the timestamp counter is also used to compute stream channel inter-packet delay.

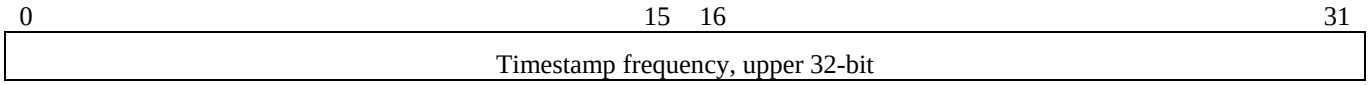
No timestamp is supported if this register is equal to 0.

[O27-25d] A device SHOULD support the “Timestamp Tick Frequency” register. 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.

An application need to access the high part before accessing the low part.

27.25.1 High Part

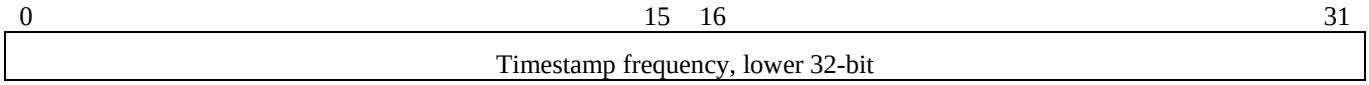
Address	0x093C
Length	4 bytes
Access type	Read
Factory Default	Device-specific



Bits	Name	Description
0 – 31	Freq - High	Timestamp frequency, upper 32-bit

27.25.2 Low Part

Address	0x0940
Length	4 bytes
Access type	Read
Factory Default	Device-specific



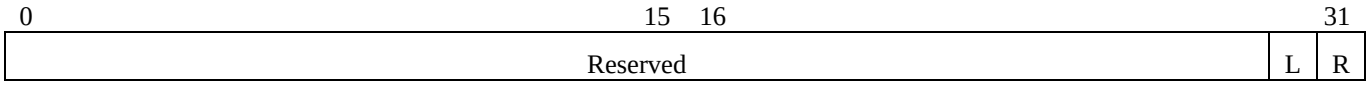
Bits	Name	Description
0 – 31	Freq – Low	Timestamp frequency, lower 32-bit

27.26 Timestamp Control

This register is used to control the timestamp counter. Writing to it has no effect if timestamp is not supported.

[CR27-26a] If a timestamp counter exists, an application MUST not attempt to read this register. This register is write-only.

Address	0x0944
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.

[CR27-27d] 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.

27.27 Timestamp Value

This register reports 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.

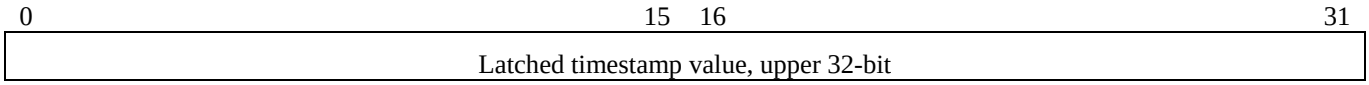
[CR27-28a] If a timestamp counter exists, an application MUST write 1 into bit 30 of Timestamp control register for the free-running timestamp counter to be copied into this register.

[CR27-29d] The timestamp value register MUST always be 0 if timestamp is not supported.

An application needs to access the high part before accessing the low part.

27.27.1 High Part

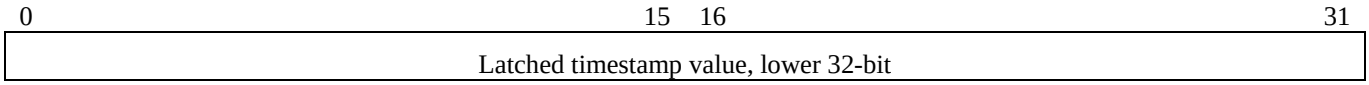
Address	0x0948
Length	4 bytes
Access type	Read
Factory Default	0



Bits	Name	Description
0 – 31	Time – High	Latched timestamp value, upper 32-bit

27.27.2 Low Part

Address	0x094C
Length	4 bytes
Access type	Read
Factory Default	0



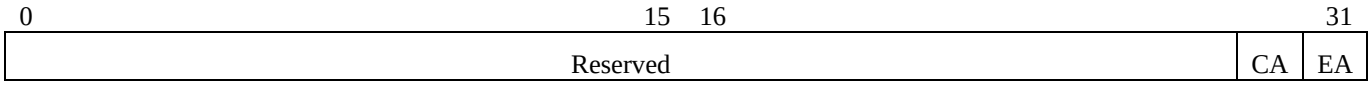
Bits	Name	Description
0 – 31	Time – Low	Latched timestamp value, lower 32-bit

27.28 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.

[R27-30a] 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 – 29	reserved	Always 0.
30	control_access	The application writing a 1 to this bit requests control access to the device. If a device grants control access, no other application can control the device. Other applications are still able to monitor the device.
31	exclusive_access	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.

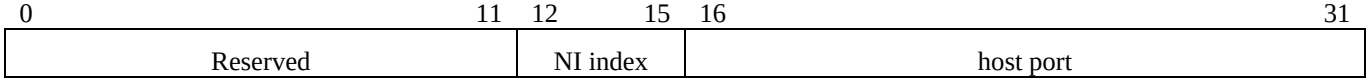
27.29 Message Channel Port Register (MCP)

This register provides port information about the message channel.

[CR27-31a] 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.

Address	0x0B00 *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



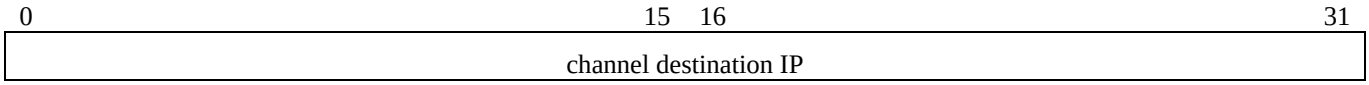
Bits	Name	Description
0 – 11	Reserved	Must be 0
12 – 15	network interface index	Always 0 in this version. Only the primary interface (#0) supports GVCP.
16 – 31	host_port	The port to which the device must send messages. Setting this value to 0 closes the message channel.

27.30 Message Channel Destination Address Register (MCDA)

This 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.

Address	0x0B10 *[optional]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



Bits	Name	Description
0 – 31	Channel destination IP	Message channel destination IPv4 address. The destination address can be a multicast or a unicast address.

27.31 Message Channel Transmission Timeout Register (MCTT)

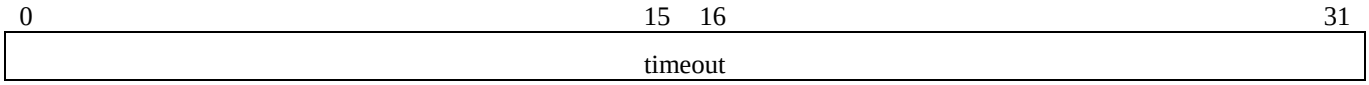
This register provides the transmission timeout value in milliseconds.

[CR27-32d] 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.

[CR27-33d] 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).

[CR27-34d] 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	0



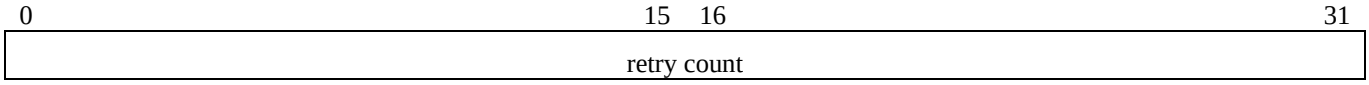
Bits	Name	Description
0 – 31	Timeout	Transmission timeout value in ms.

27.32 Message Channel Retry Count Register (MCRC)

This register indicates the number of retransmissions allowed when a message channel message times out.

[CR27-35d] 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	0



Bits	Name	Description
0 – 31	Retry count	Number of retransmissions allowed on the message channel.

27.33 Stream Channel Port Register (SCPx)

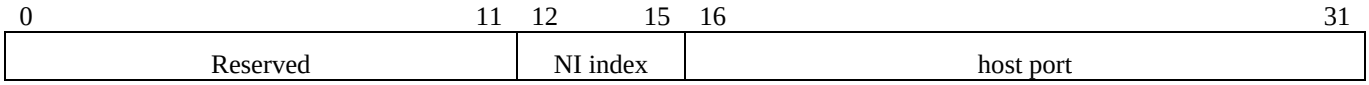
This register provides port information for this stream channel.

[R27-36a] 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.

Write access to this register is not possible while streaming is active on this channel.

Address	0x0D00 + 0x40 * x with 0 <= x < 512. *[optional for 0 < x < 512]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



Bits	Name	Description
0 – 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 the device must send data stream. Setting this value to 0 closes the stream channel.

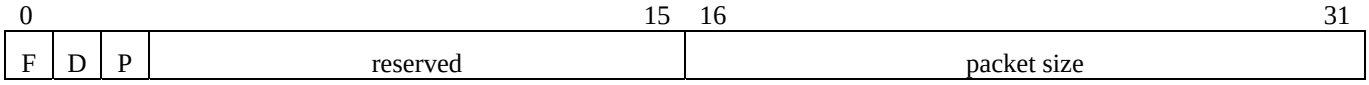
27.34 Stream Channel Packet Size Register (SCPSx)

This register indicates the packet size in bytes for this stream channel. This is the total packet size, including all headers (IP, UDP and GVSP). It also provides a way to set the IP header “do not fragment” bit and to send stream test packet to the application.

The SCPSx register returns an error to the application if the corresponding stream channel is not supported and the application tries to access it.

Write access to this register is not possible while streaming is active on this channel.

Address	0x0D04 + 0x40 * x with 0 <= x < 512. *[optional for 0 < x < 512]*
Length	4 bytes
Access type	Read/Write
Factory Default	0x40000240



Bits	Name	Description
0	Fire test packet	When this bit is set, the device 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.
1	Do Not Fragment	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.
2	Pixel Endianess	Endianess of multi-byte pixel data for this stream. 0: little endian 1: big endian This is an optional feature. A device 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	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 device cannot support the requested packet_size, then it must not fire a test packet when requested to do so.

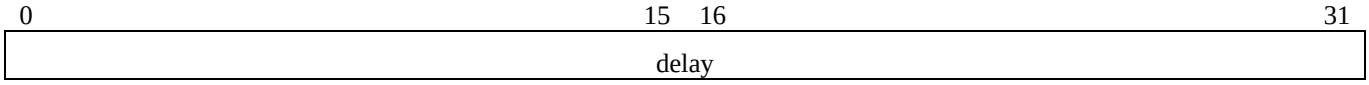
27.35 Stream Channel Packet Delay Register (SCPDx)

This register indicates the delay (in timestamp counter unit) to insert between each packet for this stream channel. This can be used as a crude flow-control mechanism if the application 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 not supported and the application tries to access it.

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]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



Bits	Name	Description
0 – 31	Delay	Inter-packet delay in timestamp tick.

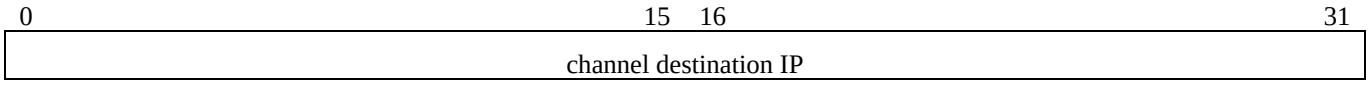
27.36 Stream Channel Destination Address Register (SCDAx)

This register indicates the destination IP address for this stream channel.

The SCDAx register returns an error to the application if the corresponding stream channel is not supported and the application tries to access it.

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]*
Length	4 bytes
Access type	Read/Write
Factory Default	0



Bits	Name	Description
0 – 31	Channel destination IP	Stream channel destination IPv4 address. The destination address can be a multicast or a unicast address.

28 Standard Feature List for Camera

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.

[R28-1d] 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 appendix. 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 appendix.

Some other features are listed in a separate document: “Standard Feature 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 “Standard Feature 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

[CR28-2d] 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. 117)
“PayloadSize”	IInteger	R	Bytes	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 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 expel an infinite sequence of image acquisition 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	W	N/A	Start image acquisition using the specified acquisition mode.
"AcquisitionStop"	ICommand	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 because most of them are relevant to the transport layer only. The XML description file is intended to expose only those 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 "Standard Feature Naming Convention". But only Width and Height are mandatory.

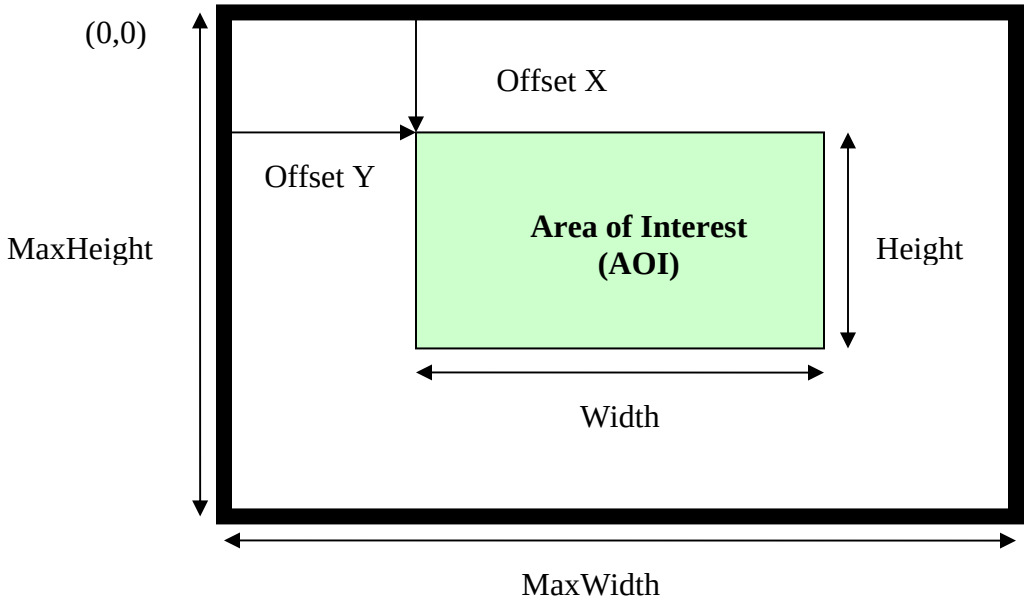


Figure 28-1: Width and Height Features

[CR28-3d] Width and Height features MUST be represented using the IInteger interface of GenICam (64-bit integer).

[CR28-4d] Width and Height features MUST be expressed in pixels.

Width can take any value ranging from 1 up to MaxWidth.

[CR28-5d] Width feature MUST take only positive values.

[CR28-6d] 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.

[CR28-7d] Height feature MUST take only positive values.

[CR28-8d] 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. 117).

[CR28-9d] 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.

[CR28-11d] 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
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_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_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_RGB10V1_PACKED	“RGB10V1Packed”	0x0220001C
GVSP_PIX_RGB10V2_PACKED	“RGB10V2Packed”	0x0220001D
GVSP_PIX_YUV411_PACKED	“YUV411Packed”	0x020C001E
GVSP_PIX_YUV422_PACKED	“YUV422Packed”	0x0210001F
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 formats represent 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 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 should be provided by PayloadSize.

[CR28-12d] PayloadSize feature MUST be represented using the IInteger interface of GenICam (64-bit integer).

[CR28-13d] PayloadSize feature MUST be expressed in bytes (8-bit value).

[CR28-14d] 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.

[CR28-15d] AcquisitionMode feature MUST be readable.

[CR28-16d] When AcquisitionMode provides multiple values in its enumeration, then it has to be writable.

[CR28-17d] 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 "Standard 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.

[CR28-18d] 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.

[CR28-19d] AcquisitionStart feature MUST be realized using the ICommand feature of GenICam.

[CR28-20d] 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.

[CR28-21d] 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.

[CR28-22d] AcquisitionStop feature MUST be realized using the ICommand feature of GenICam.

[CR28-23d] Image acquisition MUST stop as soon as possible on a frame boundary (after the data trailer) when issuing the AcquisitionStop command.

[CR28-24d] 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 Minimal XML Description File

The following is an example of a minimal XML description file which provides no functionality apart from the mandatory features defined in this section. Camera vendors might use this file as a starting point to create their own camera description files. For details see the GenICam standard text (download from www.genicam.org).

```
<?xml version="1.0" encoding="utf-8"?>
<RegisterDescription ModelName="GigEVisionMandatory" VendorName="Generic" ToolTip="Features which MUST be
supplied by GigE Vision compliant cameras" StandardNameSpace="GEV" SchemaMajorVersion="1" SchemaMinorVersion="0"
SchemaSubMinorVersion="1" MajorVersion="1" MinorVersion="0" SubMinorVersion="1" ProductGuid="CB9328DB-1273-
42a3-90C7-EB3224537C39" VersionGuid="321DAFF5-A243-42ae-8B5D-B5BD18C31488"
xmlns="http://www.genicam.org/GenApi/Version_1_0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.genicam.org/GenApi/Version_1_0
../../GenApi/GenApiSchema_Version_1_0.xsd">
        <Category Name="Root" NameSpace="Standard">
                <ToolTip>Minimum set of features a GigE Vision camera must supply</ToolTip>
                <pFeature>Width</pFeature>
                <pFeature>Height</pFeature>
                <pFeature>PixelFormat</pFeature>
                <pFeature>PayloadSize</pFeature>
                <pFeature>AcquisitionMode</pFeature>
                <pFeature>AcquistionStart</pFeature>
                <pFeature>AcquisitionStop</pFeature>
        </Category>
        <Group Comment="GigE Vision Mandatory Features">
                <Integer Name="Width" NameSpace="Standard">
                        <ToolTip>Width of the image in pixel</ToolTip>
                        <Value>1</Value>
                        <Min>1</Min>
                </Integer>
```

```
                <Integer Name="Height" NameSpace="Standard">
                        <ToolTip>Height of the image in pixel</ToolTip>
                        <Description>For linescan, this represents the height of the frame created by combining
consecutive lines together.</Description>
                        <Value>1</Value>
                        <Min>1</Min>
                </Integer>
                <Enumeration Name="PixelFormat" NameSpace="Standard">
                        <ToolTip>Pixel Format as specified in GigE Vision standard</ToolTip>
                        <Description>Refer to the Pixel format section (p. 117)</Description>
                        <EnumEntry Name="Mono8" NameSpace="Standard">
                                <Value>17301505</Value>
                        </EnumEntry>
                        <EnumEntry Name="Mono8Signed" NameSpace="Standard">
                                <Value>17301506</Value>
                        </EnumEntry>
                        <EnumEntry Name="Mono10" NameSpace="Standard">
                                <Value>17825795</Value>
                        </EnumEntry>
                        <EnumEntry Name="Mono10Packed" NameSpace="Standard">
                                <Value>17563652</Value>
                        </EnumEntry>
                        <EnumEntry Name="Mono12" NameSpace="Standard">
                                <Value>17825797</Value>
                        </EnumEntry>
                        <EnumEntry Name="Mono12Packed" NameSpace="Standard">
                                <Value>17563654</Value>
                        </EnumEntry>
                        <EnumEntry Name="Mono16" NameSpace="Standard">
                                <Value>17825799</Value>
                        </EnumEntry>
                        <EnumEntry Name="BayerGR8" NameSpace="Standard">
                                <Value>17301512</Value>
                        </EnumEntry>
                        <EnumEntry Name="BayerRG8" NameSpace="Standard">
                                <Value>17301513</Value>
                        </EnumEntry>
                        <EnumEntry Name="BayerGB8" NameSpace="Standard">
                                <Value>17301514</Value>
                        </EnumEntry>
                        <EnumEntry Name="BayerBG8" NameSpace="Standard">
                                <Value>17301515</Value>
                        </EnumEntry>
                        <EnumEntry Name="BayerGR10" NameSpace="Standard">
                                <Value>17825804</Value>
                        </EnumEntry>
                        <EnumEntry Name="BayerRG10" NameSpace="Standard">
                                <Value>17825805</Value>
                        </EnumEntry>
                        <EnumEntry Name="BayerGB10" NameSpace="Standard">
                                <Value>17825806</Value>
                        </EnumEntry>
                        <EnumEntry Name="BayerBG10" NameSpace="Standard">
                                <Value>17825807</Value>
                        </EnumEntry>
```

```
<EnumEntry Name="BayerGR12" NameSpace="Standard">
        <Value>17825808</Value>
</EnumEntry>
<EnumEntry Name="BayerRG12" NameSpace="Standard">
        <Value>17825809</Value>
</EnumEntry>
<EnumEntry Name="BayerGB12" NameSpace="Standard">
        <Value>17825810</Value>
</EnumEntry>
<EnumEntry Name="BayerBG12" NameSpace="Standard">
        <Value>17825811</Value>
</EnumEntry>
<EnumEntry Name="RGB8Packed" NameSpace="Standard">
        <Value>35127316</Value>
</EnumEntry>
<EnumEntry Name="BGR8Packed" NameSpace="Standard">
        <Value>35127317</Value>
</EnumEntry>
<EnumEntry Name="RGBA8Packed" NameSpace="Standard">
        <Value>35651606</Value>
</EnumEntry>
<EnumEntry Name="BGRA8Packed" NameSpace="Standard">
        <Value>35651607</Value>
</EnumEntry>
<EnumEntry Name="RGB10Packed" NameSpace="Standard">
        <Value>36700184</Value>
</EnumEntry>
<EnumEntry Name="BGR10Packed" NameSpace="Standard">
        <Value>36700185</Value>
</EnumEntry>
<EnumEntry Name="RGB12Packed" NameSpace="Standard">
        <Value>36700186</Value>
</EnumEntry>
<EnumEntry Name="BGR12Packed" NameSpace="Standard">
        <Value>36700187</Value>
</EnumEntry>
<EnumEntry Name="RGB10V1Packed" NameSpace="Standard">
        <Value>35651612</Value>
</EnumEntry>
<EnumEntry Name="RGB10V2Packed" NameSpace="Standard">
        <Value>35651613</Value>
</EnumEntry>
<EnumEntry Name="YUV411Packed" NameSpace="Standard">
        <Value>34340894</Value>
</EnumEntry>
<EnumEntry Name="YUV422Packed" NameSpace="Standard">
        <Value>34603039</Value>
</EnumEntry>
<EnumEntry Name="YUV444Packed" NameSpace="Standard">
        <Value>35127328</Value>
</EnumEntry>
<EnumEntry Name="RGB8Plannar" NameSpace="Standard">
        <Value>35127329</Value>
</EnumEntry>
<EnumEntry Name="RGB10Plannar" NameSpace="Standard">
```

```
                                        <Value>36700194</Value>
                                </EnumEntry>
                                <EnumEntry Name="RGB12Plannar" NameSpace="Standard">
                                        <Value>36700195</Value>
                                </EnumEntry>
                                <EnumEntry Name="RGB16Plannar" NameSpace="Standard">
                                        <Value>36700196</Value>
                                </EnumEntry>
                                <Value>17301505</Value>
                        </Enumeration>
                        <Integer Name="PayloadSize" NameSpace="Standard">
                                <ToolTip>Size of the largest buffer possible for current mode of operation</ToolTip>
                                <Description>Number of bytes transferred for each image on the stream channel, including any
end-of-line, end-offrame statistics or other stamp data. This is the 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 equal to (width x height x pixelSize) in bytes. It is strongly recommended to retrieve PayloadSize from
the camera instead of relying on the above formula.</Description>
                                <Value>1</Value>
                                <Min>1</Min>
                        </Integer>
                        <Enumeration Name="AcquisitionMode" NameSpace="Standard">
                                <ToolTip>Use by application software to set the mode of acquisition</ToolTip>
                                <Description>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, â€¦)</Description>
                                <EnumEntry Name="Continuous" NameSpace="Standard">
                                        <ToolTip>Acquires an infinite sequence of images</ToolTip>
                                        <Description>Configures the camera to start an infinite sequence of image
acquisition with the AcquisitionStart command that must be stopped explicitly by the application with the AcquisitionStop
command.</Description>
                                        <Value>1</Value>
                                </EnumEntry>
                                <Value>0</Value>
                        </Enumeration>
                        <Command Name="AcquisitionStart" NameSpace="Standard">
                                <ToolTip>Start image acquisition using the specified acquisition mode.</ToolTip>
                                <Value>0</Value>
                                <CommandValue>1</CommandValue>
                        </Command>
                        <Command Name="AcquisitionStop" NameSpace="Standard">
                                <ToolTip>Stop image acquisition using the specified acquisition mode.</ToolTip>
                                <Value>0</Value>
                                <CommandValue>0</CommandValue>
                        </Command>
                </Group>
</RegisterDescription>
```

Use the XML schema file which comes with the GenICam standard to ensure the correct syntax of your camera description file: most XML editors support checking the syntax of an XML file by means of a

schema file. In addition there is a validation tool available from the GenICam committee which will apply additional checks to the camera description file which cannot be expressed in the XML schema syntax.

28.13 Link to Naming Convention

This appendix only describes the mandatory features provided in the XML camera description file. Other recommended features are covered in “Standard Feature Naming Convention” (available in the GigE Vision page of http://www.machinevisiononline.org). This naming convention should be respected when implementing features not described in this appendix.

Proposal for new recommended feature names shall be addressed to the GigE Vision technical committee at www.machinevisiononline.org.

29 Appendix 1 – Compliancy Matrix

The Compliancy Matrix must be filled by manufacturer for each device or application software in order to obtain certification.

For each requirement and objective, please indicate the level of compliancy from the following list:

FC = Fully Compliant

NC = Not Compliant

NS = Not Supported (for Conditional requirements, Objectives and Conditional Objectives only)

This specification defines two types of component: transmitter and receiver. The transmitter is the component that emits stream data (typically a camera, referred to as a device in this specification). The receiver is the component that receives stream data (typically application software and/or frame grabber, referred to as application on this specification).

29.1 Matrix for GigE Vision Transmitter (Device)

The following table lists the requirements (mandatory and optional) that a transmitter device needs to fulfill. Requirements not applicable to a transmitter device are not listed. Manufacturer of a GigE Vision transmitter device needs to fill this matrix for each device.

Table 29-1: Transmitter Compliancy Matrix

Req.	Compliant	Comments
[R3-1d]		
[R3-2d]		
[CR3-3d]		
[CR3-4d]		
[R3-5d]		
[R3-6d]		
[R3-7d]		
[CR3-8d]		
[CR3-9d]		
[CO3-10d]		
[CO3-11d]		
[R3-12d]		
[O3-13d]		
[CR3-14d]		
[CR3-15d]		

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

[CR3-16d]		
[O3-17d]		
[O3-18d]		
[R3-19d]		
[R3-20d]		
[R3-21d]		
[O3-22d]		
[R4-1d]		
[R4-2d]		
[R4-3d]		
[O4-4d]		
[R4-5d]		
[R4-6d]		
[O4-7d]		
[R7-1]		
[R7-2]		
[R7-3]		
[R7-4]		
[O7-5]		
[R7-6]		
[R7-7]		
[R7-13d]		
[R7-14d]		
[O7-15d]		
[R7-16]		
[R8-1d]		
[R8-2d]		
[R9-1d]		
[O9-2d]		
[R9-4d]		
[R9-5d]		
[R9-6d]		
[CO9-7d]		
[CR9-8d]		
[R9-9d]		
[R9-10d]		
[R9-11d]		

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

[R9-12d]		
[R9-14d]		
[R9-15d]		
[R9-16d]		
[R10-1d]		
[R10-2d]		
[O10-3d]		
[R10-5d]		
[R10-6d]		
[CR10-7]		
[CR11-1d]		
[CO11-2d]		
[CR11-4d]		
[CR11-6d]		
[R12-1d]		
[R12-2d]		
[CR12-3d]		
[O12-4d]		
[R12-5d]		
[R12-6]		
[R12-7d]		
[R12-8d]		
[CR12-10]		
[R13-1]		
[O13-2]		
[R13-3]		
[O13-4d]		
[O13-5d]		
[R13-6]		
[O13-7]		
[R13-9]		
[R14-1d]		
[R14-2d]		
[R14-3]		
[R14-4d]		
[R14-5d]		
[O14-6d]		

[R14-7]		
[R14-8d]		
[R14-9d]		
[R14-11d]		
[R14-12d]		
[R14-13d]		
[R14-14d]		
[R14-15]		
[R14-16]		
[R14-17d]		
[R14-18d]		
[CR14-19d]		
[CR14-20d]		
[R14-21d]		
[R14-22d]		
[R14-23]		
[R14-24]		
[R14-25d]		
[R14-26d]		
[R14-27d]		
[CR14-28d]		
[CR14-29d]		
[R14-30d]		
[R14-31d]		
[R14-32d]		
[R14-33]		
[R14-34]		
[R14-35d]		
[R14-36d]		
[R14-37]		
[R14-38]		
[R14-39d]		
[R14-40d]		
[R14-41]		
[R14-42]		
[R14-43d]		
[CR14-45]		

[CR14-46]		
[CR14-47d]		
[CR14-48d]		
[CR14-49]		
[CR14-50]		
[CR14-51d]		
[CR14-52d]		
[O14-53d]		
[CR14-56d]		
[CO14-57d]		
[R14-58d]		
[CR14-59]		
[CR14-60d]		
[CR14-61d]		
[CR15-1d]		
[CO15-2d]		
[O15-3d]		
[CR15-5d]		
[CO15-6d]		
[CR15-7]		
[CR15-8]		
[CR15-10d]		
[CR15-11d]		
[CO15-12d]		
[CR15-13]		
[CR15-14]		
[R16-1]		
[R17-1d]		
[CR17-2d]		
[R17-5d]		
[O18-1d]		
[R18-2d]		
[CR19-1d]		
[R20-1d]		
[R22-1]		
[R22-2]		
[R22-4d]		

[R23-1]		
[R23-2d]		
[CR23-3d]		
[CR23-4d]		
[CR23-5d]		
[CO23-6d]		
[CR23-7d]		
[CR23-8d]		
[CR23-9d]		
[CO23-10d]		
[CO23-11d]		
[R24-1]		
[R24-2]		
[R24-3d]		
[R24-4d]		
[CR24-5]		
[CR24-6]		
[CR24-7]		
[CR24-8]		
[CR24-9]		
[R24-10d]		
[CR24-11d]		
[R24-12d]		
[CR24-13]		
[CR24-14]		
[CR24-15]		
[CR24-16]		
[CR24-17]		
[R24-18d]		
[R24-19d]		
[R24-20d]		
[R24-21]		
[CR24-22]		
[CR24-23]		
[CR24-24]		
[CR24-25]		
[CR24-26]		

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

[R24-27]		
[R24-28]		
[O25-1d]		
[R25-2]		
[O25-3d]		
[CR25-4]		
[CR25-5]		
[CR25-6]		
[CR25-7]		
[CR25-8]		
[CR25-9]		
[CR25-10]		
[CR25-11d]		
[CR25-12]		
[CR25-13]		
[CR25-14]		
[CR25-15]		
[CR25-16]		
[CR25-17]		
[CR25-18]		
[CR25-19]		
[CR25-20]		
[CR25-21]		
[CR25-22]		
[CR25-23]		
[CR25-24]		
[CR25-25]		
[CR25-26]		
[CR25-27]		
[CR25-28]		
[CR25-29]		
[CR25-30]		
[CR25-31]		
[CR25-32]		
[CR25-33]		
[CR25-34]		
[CR25-35]		

[CR25-36]		
[CR25-37]		
[CR25-38]		
[CR25-39]		
[CR25-40]		
[R27-1d]		
[R27-2d]		
[R27-5d]		
[R27-6d]		
[O27-7d]		
[O27-8d]		
[R27-10d]		
[R27-11d]		
[R27-12d]		
[R27-13d]		
[R27-14d]		
[R27-15d]		
[CO27-16d]		
[CO27-17d]		
[CR27-18d]		
[R27-19d]		
[R27-20d]		
[CR27-21d]		
[CR27-22d]		
[CR27-23d]		
[O27-24d]		
[O27-25d]		
[CR27-27d]		
[CR27-29d]		
[CR27-32d]		
[CR27-33d]		
[CR27-34d]		
[CR27-35d]		
[R28-1d]		
[CR28-2d]		
[CR28-3d]		
[CR28-4d]		

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

[CR28-5d]		
[CR28-6d]		
[CR28-7d]		
[CR28-8d]		
[CR28-9d]		
[CR28-10d]		
[CR28-11d]		
[CR28-12d]		
[CR28-13d]		
[CR28-14d]		
[CR28-15d]		
[CR28-16d]		
[CR28-17d]		
[CR28-18d]		
[CR28-19d]		
[CR28-20d]		
[CR28-21d]		
[CR28-22d]		
[CR28-23d]		
[CR28-24d]		

29.2 Matrix for GigE Vision Receiver (Application Software)

The following table lists the requirements (mandatory and optional) that receiver application needs to fulfill. Requirements not applicable to a receiver application are not listed. Manufacturer of a GigE Vision receiver application needs to fill this matrix for each application software.

Table 29-2: Receiver Compliancy Matrix

Req.	Compliant	Comments
[O5-1a]		
[R7-1]		
[R7-2]		
[R7-3]		
[R7-4]		
[O7-5]		
[R7-6]		
[R7-7]		

[O7-8a]		
[R7-9a]		
[O7-10a]		
[O7-11a]		
[R7-12a]		
[R7-16]		
[R7-17a]		
[R7-18a]		
[R7-19a]		
[R8-3a]		
[R9-3a]		
[R9-13a]		
[R10-4a]		
[CR10-7]		
[CR11-3a]		
[CR11-5a]		
[R12-6]		
[R12-9a]		
[CR12-10]		
[R13-1]		
[O13-2]		
[R13-3]		
[R13-6]		
[O13-7]		
[O13-8a]		
[R13-9]		
[R14-3]		
[R14-7]		
[R14-10a]		
[R14-15]		
[R14-16]		
[R14-23]		
[R14-24]		
[R14-33]		
[R14-34]		
[R14-37]		
[R14-38]		

[R14-41]		
[R14-42]		
[O14-44a]		
[CR14-45]		
[CR14-46]		
[CR14-49]		
[CR14-50]		
[O14-54a]		
[R14-55a]		
[CR14-59]		
[O15-4a]		
[CR15-7]		
[CR15-8]		
[O15-9a]		
[CR15-13]		
[CR15-14]		
[R16-1]		
[R17-3a]		
[R17-4a]		
[O17-6a]		
[R22-1]		
[R22-2]		
[R22-3a]		
[R23-1]		
[R24-1]		
[R24-2]		
[CR24-5]		
[CR24-6]		
[CR24-7]		
[R24-8]		
[CR24-9]		
[CR24-13]		
[CR24-14]		
[CR24-15]		
[R24-16]		
[CR24-17]		
[R24-21]		

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

[CR24-22]		
[CR24-23]		
[CR24-24]		
[CR24-25]		
[CR24-26]		
[R24-27]		
[R24-28]		
[R25-2]		
[CR25-4]		
[CR25-5]		
[CR25-6]		
[CR25-7]		
[CR25-8]		
[CR25-9]		
[CR25-10]		
[CR25-12]		
[CR25-13]		
[CR25-14]		
[CR25-15]		
[CR25-16]		
[CR25-17]		
[CR25-18]		
[CR25-19]		
[CR25-20]		
[CR25-21]		
[CR25-22]		
[CR25-23]		
[CR25-24]		
[CR25-25]		
[CR25-26]		
[CR25-27]		
[CR25-28]		
[CR25-29]		
[CR25-30]		
[CR25-31]		
[CR25-32]		
[CR25-33]		

[CR25-34]		
[CR25-35]		
[CR25-36]		
[CR25-37]		
[CR25-38]		
[CR25-39]		
[CR25-40]		
[O27-3a]		
[R27-4a]		
[O27-9a]		
[CR27-26a]		
[CR27-28a]		
[R27-30a]		
[CR27-31a]		
[R27-36a]		