EEVideo

Version 0.1 prerelease, 2026-03-16

Introduction

This is an **prerelease draft** of the Embedded Ethernet Video Specification. It is subject to significant changes in format and content before version 1.0. This version may contain fundamental errors and sections marked TBD. Comments and suggested changes are welcome.

Embedded Ethernet Video (EEVideo) is a protocol for delivering video data streams across Ethernet interfaces using standard Ethernet hardware and standard Internet IP protocols.

EEVideo is focused on embedded machine-vision applications that process live camera streams. It may also be useful to deliver sensor data in other applications.

Delivering video over Ethernet is now routine; many protocol stacks exist for different use cases. The choice often comes down to the toolchain needed to manage the protocol and process the data. EEVideo aims to provide an open-source, simplified toolchain for machine-vision applications.

Goals

Leverage Ethernet Hardware and Software Standards

Using commercial Ethernet devices provides a high-bandwidth, low-cost hardware interface platform for embedded systems.

Complying with existing Ethernet standards leverages established software tools and allows other applications to share the same network hardware infrastructure.

Low Latency

One key to high-performance machine vision is processing the freshest possible data.

Achieving the lowest latency requires:

- Raw (uncompressed or low-latency compressed) data delivery
- No frame-buffering requirements
- Minimal processing between the sensor and the network on the device
- Minimal processing between packet buffers and image-processing memory on the host

High Efficiency

Video streams can consume significant bandwidth. Minimizing protocol overhead is therefore critical.

Low Complexity

Simplifying protocol operation makes implementation both faster and less expensive.

EEVideo assumes the host (which receives the stream and controls the sensor) has significantly more processing power than the sensor device. A primary goal is therefore to keep sensor/camera complexity as low as possible.

Flexible Formats

Machine-vision applications vary widely in video formats, frame rates, and pixel characteristics. EEVideo must provide maximum flexibility for all these parameters.

Open Source

An open-source specification known to work in real applications gives developers a strong head start over ad-hoc protocols.

Many modern machine-vision applications need low-cost solutions to remain competitive. Legacy proprietary protocols and software stacks often do not fit.

Poorly documented legacy specifications, single-purpose solutions, and proprietary protocols with unclear licensing terms may present issues which may be avoided by adopting an open-source specification.

Specification Structure

The EEVideo protocol consists of two largely independent sections: the **Management Protocol** and the **Stream Protocol**. They have very different goals and therefore share little in common.

The Stream Protocol is kept as lightweight as possible. The Management Protocol is more complex to support a wide range of potential features.

Management traffic uses negligible bandwidth compared with the stream and has far less stringent latency requirements.

Versioning and History

This specification uses **semantic versioning** managed entirely through the Git repository:

- **Branches** — Development occurs on `vX.Y.Z` (unstable).
- **Antora** automatically builds each Git major.minor branch as a separate versioned site with version selector and navigation.

The authoritative revision history is maintained in the Git repository releases and the dedicated Changelog.

EEVideo Management Protocol

The Management Protocol enables a host to discover, configure, and monitor EEVideo streaming devices. The goals of the Management Protocol are limited-resource implementation and maximum flexibility.

At the minimal extreme, an EEVideo device may not require the Management Protocol. A simple device on a pre-configured embedded network can power up and begin streaming with no host interaction. More commonly, some configuration or device information is needed; these functions are provided by the Management Protocol.

CoAP Transactions

EEVideo Management uses the IETF Constrained Application Protocol (CoAP) to provide packet formats and transaction protocols that allow a host (client) to read (GET) and write (PUT) device (server) registers.

CoAP features are summarized at https://coap.space. The full specification is available here, https://datatracker.ietf.org/doc/html/rfc7252.

Implementation details of CoAP for EEVideo are in the CoAP Implementation section of this document.

EEVideo Device Registers

All host management interaction with an EEVideo device is done with 32-bit integer register-based reads and writes. The functionality of all the potential specified registers and their sub-fields is described in the Features section of this document.

This single address/value interface provides unified access to all the available features.

EEVideo Feature Discovery

EEVideo devices may offer a wide range of specified and custom features. The feature-discovery mechanism allows a host to read (from read-only registers) a complete list of all features supported by the device.

The list supplies identifiers and register addresses so the host can build a map for monitoring and control of each feature.

CoRE Device Discovery

EEVideo Discovery uses the IETF Constrained RESTful Environments (CoRE) Link Format carried in CoAP requests so hosts can multicast discovery requests to locate EEVideo-capable devices on the network. The CoRE specification is available here, https://datatracker.ietf.org/doc/html/rfc6690.

Implementation details of CoRE for EEVideo are in the Device Discovery section of this document.

IP Address Configuration

IP address configuration is not technically part of the Management Protocol, but a valid IP address must be assigned before the Management Protocol can be fully used. EEVideo offers several options for devices to establish their IP address.

EEVideo Streaming Protocol

The Streaming Protocol is used to deliver video data across the network. The goal of the streaming protocol is maximum efficiency in the device and host. It is designed for efficient use of bandwidth and minimal delivery latency.

Packet and data formats are described in the Stream section of this document.

Alternate Streaming Protocols

EEVideo devices may also support alternative streaming protocols. Most Ethernet video streaming protocols use IP/UDP packets and share more in common than they differ. The key is delivering high-volume data at low latency.

Changing a few header fields is a small modification that can enable legacy clients to process video packets using a familiar format.

Many legacy standards lack a separate formal control protocol or require more complex transactions.

The EEVideo Management Protocol can therefore add significant value to embedded systems that wish to retain compatibility with legacy streaming formats.

Normative References

- RFC 7252 — The Constrained Application Protocol (CoAP) https://datatracker.ietf.org/doc/html/rfc7252
- RFC 6690 — Constrained RESTful Environments (CoRE) Link Format https://datatracker.ietf.org/doc/html/rfc6690
- IEEE 802.3 — Ethernet standard (for physical and data-link layers)
- IANA CoAP Option Numbers registry (for final Binary Address and Register Access option numbers)

Management Protocol

The Management Protocol is used by a host to monitor and configure EEVideo devices.

In the most limited case, an EEVideo device may not require Management Protocol support. In a pre-configured embedded system the device can power up and begin streaming a standard format with no host interaction.

Conformance Levels

EEVideo defines two conformance levels for the Management Protocol:

- **Minimal** — Device powers up and streams with limited or no management interaction (hard-coded IP and default stream format). Full CoAP support is not required.
- **Full** — Device implements the complete register-based CoAP interface defined in this section and CoAP-Based Register Access. Required for discovery, dynamic configuration, and monitoring.

A device claiming “EEVideo Management support” must implement the Full level.

In most applications the host will need to configure the EEVideo camera and its stream parameters, query its capabilities and status, and/or trigger captures.

Typical Management Flow

A normal host–device interaction follows this sequence:

1. Host discovers the device (see Device Discovery).
2. Using the discovery response, the host reads register 0 (Device Capabilities) to determine supported features.
3. Host reads the feature list and builds a register map for the device.
4. Host configures stream parameters via register writes.
5. Host starts the stream.
6. Host may read status registers or trigger actions at any time.

All operations use the CoAP transactions defined in CoAP-Based Register Access.

Register-Based Transactions

All management functions for EEVideo are implemented as register transactions. Each transaction is initiated by a request packet from the host to the device and concluded by a response packet from the device to the host.

Register Data Width

All registers are 32 bits (4 bytes) wide. Not every register uses all 32 bits. Many registers are subdivided into fields of varying bit lengths.

Register Address Space

All registers are mapped into a 32-bit address space. Most devices will fit their registers into a small portion of that space.

The addresses are byte-based for maximum compatibility. Since each register is 4 bytes wide, the two least-significant bits of every register address are 0b00.

Register 0 — Device Capabilities (Required)

Every EEVideo device implementing the Management Protocol must implement register 0 as a read-only register containing a 32-bit capabilities bitmap. This provides the host with immediate information about supported features without requiring feature-discovery downloads.

The bit definitions are specified in the Features section.

Even minimal devices must implement register 0 so a host can confirm the device is EEVideo-compliant.

Security Considerations

The Management Protocol inherits the security model of CoAP. EEVideo ≤ 1.0 does not mandate DTLS. Devices operating on untrusted networks should enable CoAP DTLS as described in RFC 7252. All sensitive configuration (e.g. network keys) should be protected by write-once or authenticated registers.

CoAP-Based Register Access

EEVideo Management uses the Constrained Application Protocol (CoAP) packet formats and transaction model for control and status communication between hosts and devices.

Implementing an established standard provides several benefits over an ad-hoc protocol:

- Mature, tested protocol
- Detailed specification for improved implementation compatibility
- Range of existing options that can be implemented with minimal additional specification:
  - Security features
  - Proxy features
  - Optional alternate interfaces
- Simplified integration with other CoAP-compliant services
- Source-code libraries already available for a wide range of languages and platforms

- Existing standardized port numbers and service strings for maximum Ethernet/IP compatibility

All packet and transaction details are fully specified in RFC 7252 and are already optimized for resource-constrained devices such as embedded video cameras. Those details are not repeated here.

CoAP provides maximum flexibility for a wide range of applications. Some of that flexibility can be trimmed for EEVideo to further reduce complexity. The specific CoAP application options chosen for EEVideo are described in this section.

EEVideo uses CoAP for two distinct functions: register access and device discovery. Device discovery is described later.

EEVideo to CoAP Mapping

Terminology

The vocabularies of digital video and Ethernet networking are not well aligned. In video systems a “server” usually means a central stream manager, while in networking it means the system providing a service. Both uses are correct, but context switching can be confusing. Therefore this specification avoids the term “server” entirely.

This specification aligns with digital video terminology. The following cross-reference may be helpful:

Terminology Cross Reference

Digital Video Term	CoAP Networking Term
Camera, Sensor, Device	Server
Host, Controller	Client
Response	Acknowledge with piggybacked response
Device or Host	Endpoint
Read	GET
Write	PUT

Network Capabilities

Video applications typically require high-bandwidth, low-error networks. The low-throughput, low-bandwidth assumptions and optimizations of CoAP are therefore not necessarily valid for EEVideo.

UDP

EEVideo <= 1.0 uses only UDP packets and asynchronous message exchanges.

Port

By default, EEVideo device and host management packets use port 5683. This is the IANA-assigned

port number for unsecured CoAP. .Management Port

Port	5683

Proxies

EEVideo <= 1.0 does not require proxy features but does not prohibit them.

Security

EEVideo <= 1.0 does not require CoAP DTLS security.

Confirmable Messages

EEVideo requests shall be confirmable.

Piggybacked Responses

EEVideo responses should be piggybacked on acknowledgements whenever possible.

One Request at a Time

To support resource-limited devices, a host must limit simultaneous outstanding interactions to one per device.

Congestion Control

NStart	1

Packet Size and UDP Fragmentation

To guarantee delivery on all Ethernet links, the total size of any EEVideo CoAP packet (including Ethernet, IP, and UDP headers) should not exceed 1472 bytes (1500-byte Ethernet MTU minus 28 bytes of headers).

A single Register Read or Register Write request shall therefore not request more than 360 registers (approximately) when using the maximum 4-byte Binary Address Option. Devices may return fewer registers than requested if the response would otherwise exceed this limit.

EEVideo Extensions to CoAP

Register Access

The primary purpose of CoAP transactions in EEVideo is to provide access to the status and control registers of an EEVideo device. A host must be able to read and write these registers, and the device must return appropriate responses. This results in four basic CoAP packet types:

Management Packet Types

Packet	Source	Receiver	Payload	Options
Register Read	Host	Device	None	Register Access, Binary Address

Packet	Source	Receiver	Payload	Options
Register Write	Host	Device	Write Data	Register Access, Binary Address
Read Response	Device	Host	Read Data	None
Write Response	Device	Host	None	None

All EEVideo device capabilities can be realized with these four packets. The goal is to keep each packet as simple and efficient as possible.

Binary Address Option

Register-based requests must include an address. The simplest method is a binary value; no hardware resources are needed to convert strings. The CoAP standard defines no suitable option for a binary register address, so EEVideo defines a new **Binary Address Option**.

Binary Address Option Properties

Request Meaning	Provides a binary address that directs the request to a specific register or memory location.
Response Meaning	Provides the binary address of the register used to create the response.
Critical or Elective	Critical - Endpoints that do not recognize the Binary Address Option should report an error. The requestor is expecting a response based on the address option. If the appropriate response can not be delivered, an error is helpful.
Safe-to-Forward	Safe - Proxies need not understand the Address Option to forward it to an endpoint that does.
Part of Cache-Key	Yes — Responses differ by address, so caches must treat them separately.
Format of Value	uint — Unsigned integer in network byte order.
Length of Value	1-4 bytes - EEVideo registers are limited to a 32-bit (4 byte) address space. Typically all the registers of a device will fit in a much smaller space allowing shorter addresses. Upper (most significant) bits not included in the value are assumed to be 0.
Single or Repeatable	Repeatable — A single packet may address multiple registers.
Default Value	0 - Reading and Writing register 0 is a good default option. EEVideo device register 0 is read only, so writes will have no effect. Register 0 is also specified to contain capability details of the device, so it is helpful to retrieve its value.

Option Number	0b???? ???? ???n nn01 — (nnn must not be 111; upper bits TBD pending expert review)

Register Access Option

Registers are read with GET and written with PUT using the register address. Efficiency is improved by the **Register Access Option**, which supports multi-register operations and bit-level writes in a single packet. The actual values for each mode are defined in the Register Read and Register Write sections.

Register Access Option Properties

Request Meaning	Indicates how the register request should be processed
Response Meaning	Indicates how the register request was processed
Critical or Elective	Critical - Endpoints that do not recognize the Register Access Option should report an error. The requestor is expecting processing and a response based on this option. Receiving an error will indicate the request was not processed as expected.
Safe-to-Forward	Safe - Proxies need not understand the Register Access Option to forward it to an endpoint that might.
Part of Cache-Key	Yes — Responses differ by access option, so caches must treat them separately
Format of Value	uint — Unsigned integer in network byte order
Length of Value	1 byte
Single or Repeatable	Single — All registers in one packet use the same access option
Default Value	0 - The simplest, most common, Register Access Option is 0 for register reads and writes. If the default value is 0, the Register Access Option will not be required by most requests.
Option Number	0b???? ???? ???n nn01 — (nnn must not be 111; upper bits TBD pending expert review)

Register Access Option Value

The Register Access Option Value is a single byte divided into two fields: **type** (3 bits) and **count** (5 bits), represented as `t.c`.

Register Access Count

Multiple registers may be accessed in one request either by repeating the Binary Address Option or

by using the Count field in the Access Option. A host shall use only one of these methods per request.

The Count field (values 1–31 = 1–31 registers; 0 = 32 registers) applies only to specific access types and is ignored for others.

Register Read Access Types

Read Access Types

Code	Description	Increment	Reg Count
0b000	Read Register	no	Count
0b001	Read FIFO,	no	Until empty detected or Max Regs
0b100	Read Register	yes	Count
0b101	Read String	yes	Until null byte detected or Max Regs

TBD: define Max Regs

Register Write Access Types

Write Access Types

Code	Description	Increment	Reg Count
0b000	Write	no	Count
0b001	Set	no	1
0b010	Clear	no	1
0b011	Toggle	no	1
0b100	Write	yes	Count
0b101	Masked Write	yes	Count
0b111	Reset/Advance	yes	1

CoAP Option Number Registration

The two custom options defined above require IANA-assigned option numbers.

Until official numbers are assigned, implementations shall use the following provisional numbers for interoperability testing:

- Binary Address Option: 0x1C01 (decimal 7169)
- Register Access Option: 0x1C03 (decimal 7171)

Final numbers will be registered with IANA and updated in a future revision of this specification.

EEVideo Management Packets

Register Read Request

Register Read Request Packet

Field		Value	Description
Ver		0b01	Standard CoAP
T		0b00	Confirmable
TKL		0b0000	No Token required
Code		0x01	0.01 - Request . GET
Message ID		0x????	Set by host, matched by device in response
Token		-	Optional, no token required
Option - Register Access	Delta	0xE	16 bit Delta required to reach custom option
	Length	0x1	1 byte Register Access Option Value
	D - 269	0x??	Pending final option number assignment
	Count	0b(cnt)	5 bit Register Access Count
	Type	0b(type)	3 bit Register Access Type
Option - Binary Address	Delta	0x4(?	(Expect Binary Address Option number to be near Register Access option)
	Length	0x1-0x4	For 1-4 byte address (addr)
	D - 269	0x??	Opt Delta extended value pending option number assignment
	Addr	0x(addr)	Binary Address
Payload		-	No Payload for GET
Total Length		9-11+ bytes	(plus Ethernet/IP/UDP headers and padding)

Register Write Request

Register Write Request Packet

Field	Value	Description
Ver	0b01	Standard CoAP
T	0b00	Confirmable
TKL	0b0000	No Token required
Code	0x03	0.03 - Request . PUT
Message ID	0x????	Set by host, matched by device in response
Token	-	Optional, no token required

Field		Value	Description
Option - Register Access	Delta	0xE	16 bit Delta required to reach custom option
	Length	0x1	1 byte Register Access Option Value
	D - 269	0x??	Option Delta extended value pending option number assignment
	Count	0b(cnt)	5 bit Register Access Count
	Type	0b(type)	3 bit Register Access Type
Option - Binary Address	Delta	0x4(?)	(Expect Binary Address Option number to be near Register Access option)
	Length	0x1-0x4	For 1-4 byte for address (addr)
	D - 269	0x??	Delta value pending option number assignment
	Addr	0x(addr)	Binary Address
Payload Marker		0xFF	
Payload		0xVVVVVVVV	Contiguous 32-bit write values for requested registers in order of request
Total Length		13+ bytes	(plus Ethernet/IP/UDP headers and padding)

Register Read Response (successful)

Register Read Response Packet (successful)

Field	Value	Description
Ver	0b01	Standard CoAP
T	0b00	Confirmable
TKL	-	Matches request token length
Code	0x15	2.05 - Success.Content
Message ID	0x????	Matches request Message ID
Token	-	Matches request token content
Payload Marker	0xFF	
Payload	0xVVVVVVVV	List of 32-bit register values for requested registers, in order of request
Total Length	9+ bytes	(plus Ethernet/IP/UDP headers and padding)

Register Write Response (successful)

Register Write Response Packet (successful)

Field	Value	Description
Ver	0b01	Standard CoAP

Field	Value	Description
T	0b00	Confirmable
TKL	-	Matches request token length
Code	0x15	2.04 - Success.Changed
Message ID	0x????	Matches request Message ID
Token	-	Matches request token content
Total Length	4 bytes	(plus Ethernet/IP/UDP headers and padding)

Error Responses

All error responses shall follow standard CoAP response codes and shall be confirmable. The following codes are defined for EEVideo:

Error Response Codes

Code	Name	EEVideo Meaning
0x80	4.00 Bad Request	Malformed packet, unknown option, or invalid Register Access Type/Count
0x84	4.04 Not Found	Register address is outside the supported range or device does not implement the requested feature
0x85	4.05 Method Not Allowed	Attempted write to a read-only register
0x8F	4.15 Unsupported Content-Format	(reserved for future use)
0xA0	5.00 Internal Server Error	Device-side error (e.g. FIFO underflow during read)
0xA1	5.01 Not Implemented	Requested feature or access type not supported in this firmware version

The response may include a human-readable diagnostic payload (UTF-8 string). The Binary Address Option shall be echoed back in the error response so the host knows which register caused the failure.

Examples

Reading a Single Register

Host → Device (GET):

- Register Access Option = 0b000.00001 (type 0, count 1)
- Binary Address Option = 0x00000010 (register 0x10)

Device → Host (2.05 Content):

- Payload = 32-bit value of register 0x10

Writing Two Contiguous Registers

Host → Device (PUT):

- Register Access Option = 0b100.00010 (type 4 = incrementing write, count 2)
- Binary Address Option = 0x00000020
- Payload = [0x12345678, 0x9ABCDEF0]

Device → Host (2.04 Changed): - empty payload

Error Example (write to read-only register)

Device → Host (4.05 Method Not Allowed):

- Code = 0x85
- Payload = "Register 0x00000000 is read-only"
- Binary Address Option echoed

Registers

Register Map

EEVideo register allocation is flexible to support low-capability devices. One structure that simplifies implementation is to divide the register map into **static** (read-only, constant) and **dynamic** registers. This allows static registers to be served directly from compact ROM starting at address 0, with write requests to that space safely ignored.

Not all "read-only" registers are static. Some read-only registers change during normal operation. Mapping those dynamic read-only registers into ROM does not reduce complexity.

Read-Only Registers

Register 0 (Required)

EEVideo Management requires only one mandatory register. All others are optional.

Register 0 is a read-only (static) register containing a 16-bit EEVideo identifier and a 16-bit capabilities bitmap.

Register 0

```
(defattrs :hex {:font-family "Source Code Pro", :font-size 10})
(defattrs :plain {:font-family "Source Code Pro", :font-size 10})
```

```
(defattrs :vertical [:plain {:writing-mode "vertical-rl"}])
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def left-margin 1)
(def row-height 26)
(def box-width 16)
(def font-family "Source Code Pro")
(draw-column-headers {:font-family "Source Code Pro" :font-size 7.5})
  (draw-box "ID=0xE71D" {:span 16})
  (draw-box "c0")
  (draw-box "c1")
  (draw-box "c2")
  (draw-box "c3")
  (draw-box "c4")
  (draw-box "c5")
  (draw-box "c6")
  (draw-box "c7")
  (draw-box "c8")
  (draw-box "c9")
  (draw-box "cA")
  (draw-box "cB")
  (draw-box "cC")
  (draw-box "cD")
  (draw-box "cE")
  (draw-box "cF")
(draw-bottom)
```

Register 0 Fields

Field	Description
ID	Common identifier for EEVideo Devices. May be used to validate bit and byte endian.
c0	1 = Feature Declaration List Available
c1	(reserved)
c2	(reserved)
c3	(reserved)
c4	1 = Multiple Binary Address Options supported
c5	1 = String Type Read available
c6	1 = FIFO Read Type available
c7	1 = Read & Reset Type available
c8	1 = Masked Write available
c9	1 = Bit Toggle Write available
cA	1 = Bit Set Write available
cB	1 = Bit Clear Write available

Field	Description
cC	1 = Static IP configuration supported
cD	1 = Link-Local Address supported
cE	1 = DHCP supported
cF	1 = Multicast CoAP discovery supported

The ID field (0xE71D) is stored in network byte order (big-endian). A host may read Register 0 as a single 32-bit word and check that the high 16 bits equal `0xE71D` to confirm both byte order and that the device is EEVideo-compliant.

Registers 0x04, 0x08, 0x0C (Reserved)

These three registers are reserved for future static read-only values.

Feature Declaration List (0x10–)

The Feature Declaration List declares every supported feature and the register addresses used by that feature.

If a device implements it, the list **must** start at register 0x10.

Hosts read this list to build a complete register map. The list ends with the **End of Features** marker (defined in the Features section).

The format of these declarations is described below.

Additional Static Registers

Individual features may define additional static registers. Their addresses are specified by pointers in the Feature Declaration List.

Dynamic Registers

Dynamic registers begin after the last static register. For clarity, implementations should align the first dynamic register to subsequent power-of-two boundary.

This convention lets a host determine whether a register is static (constant) or dynamic (may change) simply by inspecting the upper address bits.

Typically, the dynamic registers may not be actual memory locations, but rather represent specific function calls.

If there are regions of writable registers that represent physical memory locations, it may be helpful to group them together in address blocks. Detailing these blocks is outside the scope of this specification.

Feature Discovery

EEVideo defines a set of standardized optional features. A device only implements the features it

needs; no resources are wasted on unused features.

Multiple instances of a given feature may be declared separately.

Each feature is listed contiguously in the Feature Declaration List beginning at register 0x10. Features may be listed in any order, but the **End of Features** marker must be last.

The only required declaration (if the list is present) is the End of Features marker.

Feature Declaration Format

```
(defattrs :hex {:font-family "Source Code Pro", :font-size 10})
(defattrs :plain {:font-family "Source Code Pro", :font-size 10})
(defattrs :vertical [:plain {:writing-mode "vertical-rl"}])
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def left-margin 1)
(def row-height 26)
(def box-width 16)
(def font-family "Source Code Pro")
(draw-column-headers {:font-family "Source Code Pro" :font-size 7.5})
  (draw-box "Feature Id" {:span 12})
  (draw-box "Feature Version" {:span 12})
  (draw-box "Pointer Count" {:span 8})
  (draw-box "Pointer 0 (if Pointer Count > 0)" {:span 32})
  (draw-gap "Additional Pointers (if Pointer Count > 1)")
(draw-bottom)
```

Each feature in the list follows this format.

Feature Declaration Fields

Field	Bits	Description
Feature ID	12	Feature identification number (see Features)
Feature Version	12	Version of the implemented feature
Pointer Count	8	Number of register pointers (0-255)
Pointer 0	32	Address of the first register for this feature
Pointer 1-n	32	Additional register addresses (if required)

The number of 32-bit registers occupied by each feature declaration is `Pointer Count + 1`.

Parsing rule: The list must be parsed by reading each declaration's length (based on Pointer Count). Simply scanning for the End of Features marker can give incorrect results.

Declared features are not required to implement every register listed in the feature specification. The Pointer Count indicates exactly how many registers are provided. Pointers to unimplemented registers must point to `0x00000000`. A host seeing a pointer of `0x0` must treat that register as not implemented.

Pointers appear in the exact order defined in the feature specification (required registers first, optional registers last). A single pointer may refer to the start of a block of related registers.

Device Discovery

Device discovery may not be required for all EEVideo systems. Some systems may preconfigure their cameras, controllers, and networks so they all know how to communicate on power up without any additional configuration.

Other systems may use off-the-shelf EEVideo cameras delivered with a standard configuration from the factory or use devices that set their own dynamic IP addresses using DHCP or ZeroConf protocols. In these cases and many others, a method of searching the network for cameras is helpful.

CoAP Multicast Discovery

EEVideo leverages the Multi-cast Dicovery features alredy specified by the Constrained Application Protocol (CoAP RFC7252) using the Constrained RESTful Environments (CoRE) Link Format (RFC6690) . These protocols work together to build a standard method to find specific devices and services on Ethernet networks.

Request Implementation

Each discovery-capable EEVideo device shall acknowledge request packets with the following parameters.

Discovery IP Destination Address

EEVideo devices supporting Device Discovery shall monitor the standard CoAP IPv4 multicast address: 224.0.1.187.

Discovery Destination Port

EEVideo devices supporting Device Discovery shall open the standard CoAP UDP port: 5683.

Discovery URL

EEVideo devices supporting Device Discovery shall respond to GET requests with the following URI options.

Discovery Request Options

Field	String	Description
Uri-Path	".well-known"	Specifies the well-known locations scheme, RFC5785
Uri-Path	"core"	Specifies the CoRE Link Format, RFC6690

Field	String	Description
query	"rt=eev.cam"	Specifies the EEVideo Camera resource type (a requested CoRE Parameter Resource Type)
query	"if=eev.reg"	Specifies the EEVideo Register based service type (a requested CoRE Parameter Link Target Attribute)

Response Implementation

Discovery Response Destination Fields

EEVideo devices supporting Device Discovery shall deliver responses to the MAC address, IP address, and UDP port contained in the source fields of the request packet headers. These fields are included in the header destination fields of the CoAP UDP IPv4 acknowledge packet.

Discovery Response Source Fields

EEVideo devices supporting Device Discovery shall deliver responses with their MAC address, IP address, and CoAP UDP port in the source fields of the CoAP UDP IPv4 acknowledge packet.

Discovery Response Payload

EEVideo devices supporting Device Discovery may include the 32 bit value of register 0 in the Payload of a Piggybacked Response. This supports the default Binary Address option = 0 in the GET request

IP Address Configuration

To communicate on an Ethernet IP network, each device must have an IP address that is unique on its subnet. On simple embedded networks without routers, the address must also belong to the same subnet as the host.

It is not typically practical to deliver cameras pre-configured with a proper IP address for their intended application. Until a proper address is established, a host cannot communicate with the device to configure an address with the standard register write process.

The EEVideo specification supports several options for configuring an IP network address.

Persistent Static IP

EEVideo Devices should provide programmable non-volitile registers for the following Ethernet Interface Feature registers: * Static IP Address * Static Subnet Mask * Static Gateway Address

The process to configure these non-volitile registers is outside the scope of this specification. NVM updates should be perfomed by an NVM update feature.

Static address assignment reduces the need for re-discovery.

DHCP

EEVideo devices may implement DHCP protocols to self assign Ethernet configuration parameters if static parameters are not implmemented or set to 0.

DHCP assigned addresses may change and require re-discovery of the device.

Link Local (ZeroConf)

EEVideo devices may implment Link Local configuation protocols to self assign Ethernet configuration parameters if static parameters are not available or if DHCP servers are not available on the network.

These devices will be discoverable at their Link Local address.

The Link Local address will not be persistant and may require re-discovery after power cycles.

Multicast Register Configuration

EEVideo devices with unconfigured Ethernet address may be discoverable via the multicast discovery process. They will report 0 for these unconfigured addresses.

EEVideo devices may implment a multicast IP configuration feature for remote configuration by an EEVideo host.

Hosts are responsible for maintaing repeatable addressing using Multicast register configuration

Streaming Protocols

Stream Protocol

Stream Protocol Introduction

Stream Packets

Stream Packet Header

Packet Format

- Start of Frame Packet
- Pixel Data Packet
- End of Frame Packet
- generic data, h.264 multi-part data, genDC data?

Data Packet Header Size Options

Packet Fmt	Description	Fmt Bytes	Flag Bytes	Frame ID Bytes	Packet ID Bytes	Additional Bytes	Total
00000110 0x06	SoF Short	1	0	3 0-16M	4 0-4B	16	24
00001010 0x0A	SoF Medium	1	1	6 0-2T	8	24	40
00001101 0x0D	SoF Long	1	3	8	8	32	52
00010010 0x12	Data Short	1	0	3	4	0	8
00010100 0x14	Data Medium	1	1	6	8	0	16
00010101 0x15	Data Long	1	3	8	8	0	20
00000010 0x02	EoF Short	1	0	3	4	0	8
00000100 0x04	EoF Medium	1	1	6	8	0	16
00000101 0x05	EoF Long	1	3	8	8	0	20

SoF Fields

Packet Fmt	Description	Pix Fmt	X Y Res	TimeStamp	X Y Off	X Y Pad	Total
00000110 0x06	SoF Short	2	2*2 0-65k	6 48 bits	2*2 0-65k	0	16
00001010 0x0A	SoF Medium	2	4*2 0-4B	8 64 bits	2*2	1*2 0-255	24
00001101 0x0D	SoF Long	4	4*2	8 64 bits	4*2	1*2	32

Pixel Format

- Bits Per Pixel 4 bits 2-32 bits per pixel
- Color Space 4 bits 16 options
  - Mono
  - RGB
  - YUV4:4:4
  - YUV4:2:2

Flags

- Resend Packet
- Previous block dropped?
- Device specific
- Transmit flag?

Frame ID

Packet ID

Start of Frame Packet

- payload type specific - payload type
- timestamp
- pixel format
- frame format field_id, field_count?
- size x
- size y
- offset x
- offset y
- padding x
- padding y

Start of Frame Short

```
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def box-width 16)
(draw-column-headers)
  (draw-box "PT = 0x06" {:span 8} )
  (draw-box "Frame ID" {:span 24})
  (draw-box "Packet ID" {:span 32})
  (draw-box "Time Stamp ms" {:span 32})
  (draw-box "Time Stamp ls" {:span 16})
```

```
  (draw-box "Pixel Format" {:span 16})
  (draw-box "Size X" {:span 16})
  (draw-box "Size Y" {:span 16})
  (draw-box "Offset X" {:span 16})
  (draw-box "OffSet Y" {:span 16})
```

Start of Frame Medium

```
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def box-width 16)
(draw-column-headers)
  (draw-box "PT = 0x0A" {:span 8} )
  (draw-box "Flags" {:span 8} )
  (draw-box "Frame ID (ms)" {:span 16})
  (draw-box "Frame ID (ls)" {:span 32})
  (draw-box "Packet ID (ms)" {:span 32})
  (draw-box "Packet ID (ls)" {:span 32})
  (draw-box "Time Stamp ms" {:span 32})
  (draw-box "Time Stamp ls" {:span 32})
  (draw-box "Pixel Format" {:span 16})
  (draw-box "Pad X" {:span 8})
  (draw-box "Pad Y" {:span 8})
  (draw-box "Size X" {:span 32})
  (draw-box "Size Y" {:span 32})
  (draw-box "Offset X" {:span 16})
  (draw-box "OffSet Y" {:span 16})
```

Start of Frame Long

```
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def box-width 16)
(draw-column-headers)
  (draw-box "PT = 0x0D" {:span 8} )
  (draw-box "Flags" {:span 24} )
  (draw-box "Frame ID (ms)" {:span 32})
  (draw-box "Frame ID (ls)" {:span 32})
  (draw-box "Packet ID (ms)" {:span 32})
  (draw-box "Packet ID (ls)" {:span 32})
  (draw-box "Time Stamp ms" {:span 32})
  (draw-box "Time Stamp ls" {:span 32})
  (draw-box "Pixel Format" {:span 32})
  (draw-box "Pad X" {:span 16})
  (draw-box "Pad Y" {:span 16})
  (draw-box "Size X" {:span 32})
  (draw-box "Size Y" {:span 32})
  (draw-box "Offset X" {:span 32})
  (draw-box "OffSet Y" {:span 32})
```

Pixel Data Packet

- Size of Header 4, 4-64 bytes 0.5 bytes
- Packet type 4, 16 packet type options 0.5 bytes
- Frame ID 24 bits 3 bytes
- Packet ID 32 bits 4 bytes == 8 Bytes minimum

Data Packet Header Size Options

Size	Packet Fmt	Flag Bits	Frame ID Bits	Packet ID bits
8	1	0	3 (24 bit)	4 (32 bit)
16	1	1	6 (48 bit)	8 (64 bit)
20	1	3	8 (64 bit)	8 (64 bit)

Pixel Data Short

```
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def box-width 16)
(draw-column-headers)
  (draw-box "PT = 0x102" {:span 8} )
  (draw-box "Frame ID" {:span 24})
  (draw-box "Packet ID" {:span 32})
  (draw-gap "Pixel Data")
  (draw-bottom)
```

Pixel Data Medium

```
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def box-width 16)
(draw-column-headers)
  (draw-box "PT = 0x104" {:span 8} )
  (draw-box "Flags" {:span 8} )
  (draw-box "Frame ID (ms)" {:span 16})
  (draw-box "Frame ID (ls)" {:span 32})
  (draw-box "Packet ID (ms)" {:span 32})
  (draw-box "Packet ID (ls)" {:span 32})
  (draw-gap "Pixel Data")
  (draw-bottom)
```

Pixel Data Long

```
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def box-width 16)
```

```
(draw-column-headers)
  (draw-box "PT = 0x105" {:span 8} )
  (draw-box "Flags" {:span 24} )
  (draw-box "Frame ID (ms)" {:span 32})
  (draw-box "Frame ID (ls)" {:span 32})
  (draw-box "Packet ID (ms)" {:span 32})
  (draw-box "Packet ID (ls)" {:span 32})
  (draw-gap "Pixel Data")
  (draw-bottom)
```

End of Frame Packet

- Frame ID
- Packet ID

End of Frame Short

```
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def box-width 16)
(draw-column-headers)
  (draw-box "PT = 0x02" {:span 8} )
  (draw-box "Frame ID" {:span 24})
  (draw-box "Packet ID" {:span 32})
```

End of Frame Medium

```
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def box-width 16)
(draw-column-headers)
  (draw-box "PT = 0x04" {:span 8} )
  (draw-box "Flags" {:span 8} )
  (draw-box "Frame ID (ms)" {:span 16})
  (draw-box "Frame ID (ls)" {:span 32})
  (draw-box "Packet ID (ms)" {:span 32})
  (draw-box "Packet ID (ls)" {:span 32})
```

End of Frame Long

```
(def column-labels (mapv #(number-as-hex % 2) (range 32)))
(def boxes-per-row 32)
(def box-width 16)
(draw-column-headers)
  (draw-box "PT = 0x05" {:span 8} )
  (draw-box "Flags" {:span 24} )
  (draw-box "Frame ID (ms)" {:span 32})
  (draw-box "Frame ID (ls)" {:span 32})
  (draw-box "Packet ID (ms)" {:span 32})
```

```
  (draw-box "Packet ID (ls)" {:span 32})
```

Full Frame Single Packet

TBD

Device Feature Specifications

Optional features are described here. The process for discovering which features are implemented and where their registers are located is described in Register Map.

Each standardized feature has its own unique Feature ID and version.

Features are grouped into the following categories

Feature Groups

Group	ID Start	Description
System	0x000	Device identification and management
Ethernet	0x100	Etherenet configuration
Stream	0x200	Video stream configuration
Sources	0x300	Video source management
Processing	0x400	Image Processing
Timing	0x500	Timing management
Peripheral	0x600	Controllers for embedded peripherals
Markers	0x700	Markers for the Feature Discovery List
Custom	0x800	User specified features

Feature List

Features

Group	ID	Feature Name
System	0x001	Identification Strings
	0x002	Register Detail Embedded
	0x003	Register Detail URL
	0x010	Test Register Read/Write
	0x0??	Custom Strings
	0x0??	Command Script
	0x0??	NVM Backup
	0x0??	Reset / Sleep
	0x0??	Unsolicited Response
	0x0??	Watchdog
	0x0??	Write Enable

Group	ID	Feature Name
Ethernet	0x101	Etherenet Interface
	0x102	Ethernet PHY
	0x104	Packet Resend
Stream	0x201	Video Stream
	0x202	Meta-Data Stream
	0x210	Parametric Pixel Format
Sources	0x305	MIPI Monitor
	0x310	Test Pattern Generator
Processing	0x400	TBD
Timing	0x501	Timestamp
	0x502	NTP Protocol
	0x503	PTP Protocol
	0x504	GPS Synchronization
	0x505	Timer / Trigger
Peripheral	0x601	I2C Controller
	0x602	SPI Controller
	0x603	UART Controller
	0x604	SMI (MDIO) Controller
	0x605	CAN Controller
	0x606	PWM Controller
	0x607	GPIO Port
Markers	0x7FF	End of Features Marker

System Features

Identification Strings Feature (0x001)

The Identification Strings Feature provides pointers to registers containing strings that describe the device.

Identification Strings

Feature ID	0x001	
Version	0x001	(0.1)
Pointer Count	7	
Bootstrap ID	0x00100107	(Max pointers)

Pointer 0	String	Manufacturer Name
Pointer 1	String	Device Model Name
Pointer 2	String	Device Version
Pointer 3	String	Serial Number
Pointer 4	String	Manufacturer Info
Pointer 5	String	User Defined Name
Pointer 6	String	Device Webpage (URL)

A string pointer points to the first register of a variable-length, null-terminated string.

Manufacturer Name

Typically static.

Device Model Name

Typically static. May be used for mDNS service discovery.

Device Version

Typically static.

Serial Number

Should be unique per model. May be used for mDNS.

Manufacturer Info

Typically static.

User Defined Name

User-writable, stored in non-volatile memory. Must hold at least 16 characters (including null terminator).

Device Webpage

URL to documentation or manual.

Register Detail Embedded File Feature (0x002)

An EEVideo device may offer structured metadata files containing additional information on the device registers. These files are stored on the device and are typically compressed.

Embedded Register Detail File

Feature ID	0x002	
Version	0x001	(0.1)

Pointer Count	1	
Bootstrap ID	0x00200101	
Pointer 0	File	File Length / Name / Contents

Register Detail File

The pointer points to the first of a set of registers. The number of registers depends on the name string length and the file content.

Register Detail File Registers

Offset 0	4 Bytes	Length of file in bytes (excluding length and name)
Offset 1	String	File name and extension
Offset n+1	Data	File contents

The format of the file is determined by its extension (detailed in a future section).

Register Detail File URL Feature (0x003)

Register Detail Files may also be made available at a URL.

Embedded Register Detail File URL

Feature ID	0x003	
Version	0x001	(0.1)
Pointer Count	1	
Bootstrap ID	0x00300101	
Pointer 0	String	URL for register detail file

URL for Register Detail File

A string containing a full URL where the register detail file can be downloaded.

NVM Backup Feature (0x0??)

The NVM Backup Control feature provides pointers to registers that store and recall specific blocks of registers to and from non-volatile memory included in the device.

TBD

Command Script Feature (0x0??)

The Command Script Feature provides pointers to registers that allow building and executing lists of register commands. These lists may be used for initialization or mode switching with a single register write.

TBD

Watchdog / Heartbeat Feature (0x0??)

The Watchdog / Heartbeat Feature provides pointers to registers with timer functionality to indicate the device is functioning properly and reset the device if timeouts occur or communication with the host is lost.

TBD

Reset / Sleep Feature (0x0??)

The Reset / Sleep Feature provides pointers to registers that reset internal device processes and put the device into reduced-functionality, lower-power modes.

TBD

Unsolicited Response Feature (0x0??)

The Unsolicited Response Feature provides pointers to registers that allow the device to deliver unsolicited read response packets to a host when specified events occur.

TBD

Write Enable Feature (0x0??)

TBD

Custom features may be implemented, included in the Feature Manifest, and further defined in Register Detail Files.

Custom features **must** use a Feature ID value ≥ 0x800 and < 0x900 (included in their Bootstrap ID).

Ethernet Features

Ethernet Interface Feature (0x101)

The Ethernet Interface Feature provides pointers to registers containing configuration information about an Ethernet interface on the device.

Ethernet Interface

Feature ID	0x101	
Version	0x001	(0.1)

Pointer Count	8	
Bootstrap ID	0x10100108	(Max pointers)
Pointer 0	String	Name
Pointer 1	Reg 64	MAC Address
Pointer 2	Reg 32	IP Address
Pointer 3	Reg 32	Static IP Address
Pointer 4	Reg 32	Subnet Mask
Pointer 5	Reg 32	Static Subnet Mask
Pointer 6	Reg 32	Gateway Address
Pointer 7	Reg 32	Static Gateway Address

Name (Pointer 0)

Human-readable null-terminated string name for this interface.

Ethernet MAC Address (Pointer 1)

The 48-bit Ethernet MAC address.

Typically a unique static value assigned during manufacture from a block of MAC addresses assigned to the manufacurer by the IEEE.

IP Address (Pointer 2)

IPv4 address used by the interface (may be 0 if uninitialized).

Static IP Address (Pointer 3)

The user assigned IPv4 address used to be used by the interface.

If set to 0 or not implemented, another method is required to assign the operating device IP address.

Subnet Mask (Pointer 4)

32-bit IPv4 subnet mask.

Static Subnet Mask (Pointer 5)

The user assigned 32-bit IPv4 subnet mask.

If set to 0 or not implemented, another method may be required to assign the operating device Subnet Mask.

Gateway Address (Pointer 6)

32-bit Ethernet gateway address.

Static Gateway Address (Pointer 7)

The user assigned 32-bit Ethernet gateway address.

If set to 0 or not implemented, another method may be required to assign the operating device Ethernet gateway address.

Ethernet PHY Feature (0x102)

The Ethernet PHY Feature offers pointers to values describing and controlling one Ethernet PHY connected to the device. Multiple Ethernet PHY features may be implemented.

Ethernet PHY

Feature ID	0x102	
Version	0x001	(0.1)
Pointer Count	3	
Bootstrap ID	0x10200103	(Max pointers)
Pointer 0	Reg 32	Link Status
Pointer 1	Reg 32	PHY Capability
Pointer 2	Reg 32	SMI Port / Address

Link Status

TBD Register table

PHY Capability

TBD Register table

SMI Port / Address

TBD Register table

Packet Resend Feature (0x104)

EEVideo devices may implement functionality to retransmit video stream packets that were lost or damaged. The host requests retransmission using the registers pointed to by this feature.

The device must maintain memory buffers long enough for the host to request retransmission.

Packet Resend

Feature ID	0x104	

Version	0x001	(0.1)
Pointer Count	4	
Bootstrap ID	0x10400104	(Max pointers)
Pointer 0	Reg 32	Video Channel
Pointer 1	Reg 32	Block ID
Pointer 2	Reg 32	Packet ID
Pointer 3	Reg 32	Packet Availability

The response to a resend request includes the Packet Availability field.

Stream Features

Video Stream Feature (0x201)

Video Stream

Feature ID	0x201	
Version	0x001	(0.1)
Pointer Count	12	
Bootstrap ID	0x2010010C	(Max pointers)
Pointer 0	String	Description
Pointer 1	Reg 32	Stream Config
Pointer 2	Reg 32	Delay
Pointer 3	Reg 64	Dest MAC Addr (MSBs)
Pointer 4	Reg 32	Dest IP Addr
Pointer 5	Reg 32	Dest Port
Pointer 6	Reg 32	Source Port
Pointer 7	Reg 32	Pixels Per Line (Width)
Pointer 8	Reg 32	Lines Per Frame (Height)
Pointer 9	Reg 32	Pixel Format
Pointer 10	Reg 32	X Offset
Pointer 11	Reg 32	Y Offset

Description (Pointer 0)

Stream Config (Pointer 1)

Stream Config Register

Field	bits	msb
Fire Test Pkt	1	31
Stream Format	4	23
Enable Stream	1	16
Max Pkt Size	15	15

Fire Test Packet Field

Set to 1 to launch a single test packet of the Max Packet Size.

Self clearing.

Stream Format

Selects Stream Format option for the stream.

Enable Stream

Set to 1 to enable the stream.

Max Packet Size

Set to limit the maximum packet size.

Max Packet size may also be restricted by an internal hardware limit.

Setting Max Packet Size to 0 will disable the stream.

Delay (Pointer 2)

Count of clocks to delay between stream packets. May be 0. Ethernet minimum interpacket gaps must be enforced by logic.

Destination MAC Address (Pointer 3)

Not required for EEVideo devices with ARP capability.

Destination IP Address (Pointer 4)

32-bit IPv4 address.

Destination Port (Pointer 5)

Destination UDP Port as required by host.

Source Port (Pointer 6)

Source UDP Port as required by host.

Pixels per Line (Width) (Pointer 7)

Lines per Frame (Height) (Pointer 8)

Pixel Format (Pointer 9)

Index of the selected Parametric Pixel Format feature.

X Offset (Pointer 10)

Pixel offset from left side of original frame.

Y Offset (Pointer 11)

Pixel offset from top side of original frame.

Parametric Pixel Format (0x210)

EEVideo pixel formats are declared with Parametric Pixel Format descriptions.

Embedded systems require maximum flexibility for pixel formats. Traditional standards limit formats to predefined legacy video types. Modern embedded applications increasingly use non-visible wavelengths, depth sensing, hyper-spectral imaging, and other custom data that do not fit classic video formats.

EEVideo uses a **parametric** description system instead of a fixed list. This enables the widest possible range of formats — including legacy ones (HDR monochrome, RGBA, sub-sampled YUV, Bayer) and future ones (hyper-spectral, 3D, event-based) — while allowing drivers and host processors to understand buffering requirements and frame sizes without knowing every component detail.

This also simplifies device implementation. Stream generators can use parametric fields to format the output stream rather than maintain large implementation specific look-up tables.

Parametric Pixel Format

Feature ID	0x210	a
Version	0x001	(0.1)
Pointer Count	4	b
Bootstrap ID	0x21000104	(Max pointers)
Pointer 0	String	Description
Pointer 1	Reg 32	Pixel Group Format Register
Pointer 2	Reg 32	Valid Streams (bit-field)
Pointer 3	Reg 32	Stream XRef Count

Description (Pointer 0)

User-defined null-terminated string that describes the format.

Pixel Group Format Register (Pointer 1)

The pixel Format register points to the first register of the list of registers that define the format.

The parametric format is built around repeating **sample groups**. A group is the smallest set of samples that repeats across the frame. * For monochrome: the group is one sample. * For RGB or YUV 4:4:4: the group contains 3 samples. * For Bayer or YUV 4:2:2: the group is more complex but still fully described.

The **Pixel Group Format Register** defines the relationship between sample groups and image pixels.

Pixel Group Format Register

Field Name	Bits	Description
Y Samples per Group	4	Number of sample lines required for each sample group
X Samples per Group	4	Number of samples on each line included in the group
Y Pixels per Group	4	Number of pixel lines represented by the group
X Pixels per Group	4	Number of pixels in each line represented by the group
Interleaved Columns	4	Number of column bands (multi-tap sensors)
Interleaved Rows	4	Number of row bands (multi-tap sensors)
Bits per Pixel	8	Total bits required to represent one image pixel pixel

Standard Format Examples

Standard Format Examples

Format	Y s/g	X s/g	Y p/g	X p/g	ICols	IRows	BPP	Comps
Monochrome 8	1	1	1	1	1	1	8	1
Monochrome 12	1	1	1	1	1	1	12	1
RGB 8:8:8	1	3	1	1	1	1	24	3
RGB 5:6:5	1	3	1	1	1	1	16	3
RGBA 10-bit	1	4	1	1	1	1	40	4
YUV 4:4:4 10	1	3	1	1	1	1	30	3
YUV 4:2:2 12	1	4	1	2	1	1	24	4
YUV 4:1:1 8	2	3	2	2	1	1	12	6
Bayer GRBG 10	2	2	2	2	1	1	10	4

Field Descriptions

Y / X Samples per Group

Number of raw samples in each repeating group (vertical × horizontal).

For single sample (Mono) formats, they are always 1.

For RGB, RGBA, YUV 4:4:4, and 4:2:2, Y is still one because the samples are the same for each line. X indicates the number of samples of different types (i.e. 3 for RGB, 4 for YUV 4:2:2 which is delivered U,Y,V,Y).

Y Samples/Group is two for YUV 4:1:1 and Bayer patterns because those patterns are not the same on consecutive lines, but repeat every 2 lines.

Y / X Pixels per Group

Number of actual image pixels represented by one sample group. This allows correct frame-size calculation even when multiple samples map to one pixel (e.g. Bayer or 4:2:2).

Y and X Samples/Pixel indicates the number pixels represented by each sample group. Frame sizes are specified by X and Y pixel dimensions, but the data stream may deliver multiple samples per pixel. These Pixels/Group provides the group to pixel mapping.

For single sample formats Y and X Pixels/Group are always 1. For many multi-sample formats Y is still 1 and X represents the sample count, just like the group fields. These fields may differ from the Samples/Group settings when individual pixels are represented by different sets of samples.

Bayer and half-band chroma patterns are good examples. Each pixel in YUV 4:2:2 requires 1 Y, 2 X subgroup. For Bayer patterns, each pixel requires only 1 X and 1 Y sample out of the 2x2 group so each group provide 2x2 pixels.

Interleaved Columns / Rows

Number of column or row bands delivered by multi-tap sensors (usually 1,1 for non-interleaved delivery).

Bits Per Pixel

Total of bits needed to represent one image pixel pixel. Used with frame width/height to calculate buffer size.

Sample Type Registers (Pointer 1 + x)

Immediately following the Pixel Group Format Register are one or more **Sample Type Registers** (one per sample in the group).

Sample types must be listed in the order the samples arrive.

Sample Type Registers

Field	Bits	Description
Type	8	Top-level sample category
Sub-type	12	Specific variant or standard
Pad Bits	4	Padding bits for byte alignment
Bit Depth	8	Valid data bits per sample

Sample Type Field (8-bit)

The 8 bit sample type field contains an index to the EEVideo sample type list.

The sample type list is short and only defines the most general category of the sample. Each Type has a list of sub-types that more specifically define the qualities of the actual sample. In many cases, the simple Type identifier will be adequate for classifying the format and selecting processing options.

Sample types are defined in the appendix.

Sample Sub-type Field

Specific assignments are TBD. Sub-types can express conformance to particular video standards like sRGB, SMPTE, or ITU-R BT component specifications. Sub-types may indicate raw, unprocessed, sensor data. Sub-types may indicate linear or non-linear (gamma) scales. Sub-types may indicate bandwidth limiting (half,full) of the sample.

Sub-type assignment lists are unique for each top level Sample Type.

User defined sub-types are also available.

Sub-type lists are expected to grow over the life of the EEVideo specification.

Pad Bits Field (packing)

By default samples in EEVideo streams are packed together, not byte aligned. This allows 4x 12 bit pixels to fit in a 6 byte (48 bit) window. This increases channel efficiency, but may require unpacking effort from the host.

If byte aligned samples are desired, pad bits may be added in front of the sample data. For example, adding 4 pad bits would extend a 12 bit sample to 16 bits (2 bytes) in the stream data. Now those same 4 pixels require 8 bytes of stream bandwidth, but the host will not be responsible for bit manipulations to separate the samples.

Bit Depth Field

The Bit Depth Field reports the bit resolution of each sample type. Samples frequently use the same resolution for all sample types, but this is not required. RGB 565 is an example.

Valid Streams (Pointer 2)

Bit-field indicating which streams support this Pixel Format (bit 0 = stream 0, etc.).

An EEVideo device may support multiple streams and all streams may not support all pixel formats.

Stream XRef Count (Pointer 3)

EEVideo format streams include the Format Feature index in Start of Frame packets to identify the pixel format.

Other streaming formats use other identifiers.

The Stream XRef Count register indicates the number of alternate stream packet formats able to use this pixel format.

Stream XRef Format (Pointer 3 + 1 + n×2)

Indicates to which alternate stream packet feature this cross reference applies.

Stream XRef Value (Pointer 3 + 2 + n×2)

Provides the pixel format identifier for the alternate stream packet protocol.

Source Features

MIPI Monitor Feature (0x305)

MIPI Monitor Declaration

Feature ID	0x305	
Version	0x001	(0.1)
Pointer Count	2	
Bootstrap ID	0x30500102	(Max pointers)
Pointer 0	Reg Block	MIPI Monitor Block
Pointer 1	String	Description

MIPI Monitor Registers

Pointer 0 points to a block of 4 contiguous registers.

Register Offset 0 — Activity

MIPI Activity Register

PCx16	Unused	CActive	CCont	L0A	L1A	L2A	L3A	V0A	V1A	V2A	V3A

Register Offset 4 — LineInfo

MIPI LineInfo Register

PixelClockCount	Bytes Per Line

Register Offset 8 — LineCount

MIPI LineCount Register

Unused	LineCount

Register Offset 0xC — DataType

MIPI DataType Register

U	DType0	U	DType1	U	DType2	U	DType3

Test Pattern Generator (0x310)

Test Pattern Generator

Feature ID	0x310	
Version	0x001	(0.1)
Pointer Count	8	
Bootstrap ID	0x31000108	(Max pointers)
Pointer 0	String	Description
Pointer 1	Reg 32	Applicable Streams (bit-field)
Pointer 2	Reg 32	Test Pattern Mode Register
Pointer 3	Reg 32	Test Pattern Mode Count
Pointer 4	Strings	Test Pattern Mode 0 Description
Pointer 5	Reg 32	Test Pattern Param Count
Pointer 6	Strings	Test Pattern Param 0 Description
Pointer 7	Reg 32	Test Pattern Parameter 0

Description (Pointer 0)

User-defined null-terminated string that describes this test pattern generator.

Applicable Streams (Pointer 1)

Bit-field indicating which streams include this test pattern generator (bit 0 = stream 0, etc.).

All streams marked available use the same register settings for this generator.

Test Pattern Mode Register (Pointer 2)

Read / Write Register to set the test pattern generator mode.

0 always disables the generator and enables the alternative stream.

Test Pattern Mode Count (Pointer 3)

Count of functional test pattern modes.

Test Pattern Mode x Description (Pointer 4 + offset from prior string)

User-defined list of null-terminated 32-bit word-aligned strings that describe each of the test pattern modes.

The mode 0 description should indicate "off" or "bypass".

Test Pattern Parameter Count (Pointer 5)

Count of available test pattern parameters.

Test Pattern Parameter Descriptions (Pointer 6 + offset from prior string)

User-defined null-terminated 32-bit word-aligned strings that describe each available parameter.

Test Pattern Parameter Registers (Pointer 7 + n)

Read / Write registers for the specified test pattern parameters.

Processing Features

TBD Processing features like color correction control, gamma control, JPEG control

Timing Features

Timestamp Feature (0x501)

Timestamp

Feature ID	0x501	
Version	0x001	(0.1)
Pointer Count	4	
Bootstrap ID	0x50100104	(Max pointers)
Pointer 0	Reg 32	
Pointer 1	Reg 32	
Pointer 2	Reg 64	

NTP Feature (0x502)

EEVideo devices may implement Network Time Protocol (NTP) for Time Stamp synchronization.

Network Time Protocol

Feature ID	0x502	
Version	0x001	(0.1)
Pointer Count	3	
Bootstrap ID	0x50200103	(Max pointers)

Pointer 0	Reg 32	NTP Configuration
Pointer 1	Reg 32	NTP Status
Pointer 2	Reg 32	NTP Server IP Address

TBD pointer descriptions

PTP (IEEE-1588) Feature (0x0503)

EEVideo devices may implement an IEEE-1588 Precision Time Protocol (PTP) for higher precision Time Stamp synchronization.

Precision Time Protocol

Feature ID	0x503	
Version	0x001	(0.1)
Pointer Count	3	
Bootstrap ID	0x50300103	(Max pointers)
Pointer 0	Reg 32	PTP Configuration
Pointer 1	Reg 32	PTP Status
Pointer 2	Reg 32	PTP Server IP Address

TBD pointer descriptions

GPS Synchronization Feature (0x504)

EEVideo devices may implement an interface to synchronize the time stamp value with signals from a GPS receiver.

GPS Synchronization

Feature ID	0x504	
Version	0x001	(0.1)
Pointer Count	3	

Bootstrap ID	0x50400103	(Max pointers)
Pointer 0	Reg 32	GPS Configuration
Pointer 1	Reg 32	GPS Status
Pointer 2	Reg 32	GPS UART port

TBD pointer descriptions

Timer / Trigger Feature (0x505)

EEVideo devices may implement timers and triggers driven by the time stamp value.

Timer / Trigger

Feature ID	0x505	
Version	0x001	(0.1)
Pointer Count	6	
Bootstrap ID	0x50500106	(Max pointers)
Pointer 0	Reg 32	Timer Configuration
Pointer 1	Reg 32	Timer Status
Pointer 2	Reg 64	Timer stamp A
Pointer 3	Reg 64	Timer stamp B
Pointer 4	Reg 64	Timer stamp Out
Pointer 5	Reg 32	Timer I/O ports

TBD pointer descriptions

Peripheral Features

I2C Controller Feature (0x601)

The I2C Controller Feature provides a bridge to manage devices on an I2C bus. This is useful for interfacing to MIPI video devices that provide an I2C port for management.

The controller can manage multiple I2C devices on the same bus if they have unique addresses.

I2C Controller Declaration

Feature ID	0x601	
Version	0x001	(0.1)
Pointer Count	2	
Bootstrap ID	0x60100102	(Max pointers)
Pointer 0	Reg Block	I2C Peripheral Block
Pointer 1	String	Description

I2C Controller Pointers

I2C Controller Registers

Register Offset 0 — Status

I2C Status Register

Done	Holding	Unused	Rate	Max	TxLevel	RxLevel

I2C Rx FIFO Register Fields

Field	Access	msb	len	Description
Done	RO	31	1	'0' indicates Controller is Idle
Holding	RO	30	1	'1' indicates Controller is holding the bus
Rate	RO	23	4	0x1=100kbps, 0x4=400kbps, 0x8=1Mbps
MaxLevel	RO	19	4	Maximum FIFO level
TxLevel	RO	15	8	Current entries in Tx FIFO
RxLevel	RO	7	8	Current entries in Rx FIFO

Register Offset 4 — Config

I2C Config Register

Unused	Reset

I2C Config Fields

Field	Access	msb	len	Description
Reset	RWSC	0	1	Resets I2C controller and FIFOs, self-clearing

Register Offset 8 — Tx Data FIFO

I2C Tx FIFO Register Field Map

Unused	Stop	Hold	Ack	Data

I2C Tx FIFO Register Fields

Field	Access	msb	len	Description
Stop	WO	10	1	'1' adds a Stop condition after the data byte
Hold	WO	9	1	'1' holds the bus for the next byte
Ack	WO	8	1	Ack value to set after data byte
Data	WO	7	8	Data byte to transfer

Register Offset 0xC — Rx Data FIFO

I2C Rx FIFO Register Field Map

Unused	Ack	Data

I2C Rx FIFO Register Fields

Field	Access	msb	len	Description
Ack	RO-AoW	8	1	Ack bit captured after the byte
Data	RO-AoW	7	8	Byte value captured from the I2C bus

Pointer 1 — Port Description

Pointer to a null-terminated string describing the function of this I2C Controller (e.g. corresponding MIPI port).

SPI Controller Feature (0x602)

The SPI Controller Feature provides a bridge to manage devices on a SPI bus. Useful for updating SPI Flash memories containing FPGA and microcontroller firmware.

SPI Controller

Feature ID	0x602	
Version	0x001	(0.1)
Pointer Count	4	
Bootstrap ID	0x60200104	(Max pointers)

Pointer 0	Reg 32	Config
Pointer 1	Reg 32	Control
Pointer 2	Reg 32	SPI Tx Buffer
Pointer 3	Reg 32	SPI Rx Buffer

TBD pointer descriptions

UART Controller Feature (0x603)

UART Controller

Feature ID	0x603	
Version	0x001	(0.1)
Pointer Count	4	
Bootstrap ID	0x60300104	(Max pointers)
Pointer 0	Reg 32	UART Configuration
Pointer 1	Reg 32	UART Status
Pointer 2	Reg 32	UART Tx Buffer
Pointer 3	Reg 32	UART Rx Buffer

TBD pointer descriptions

SMI (MDIO/MDC) Controller Feature (0x604)

The SMI Controller Feature offers pointers to SMI (MDIO/MDC) controller registers, typically connected to Ethernet PHYs.

SMI Controller

Feature ID	0x604	
Version	0x001	(0.1)

Pointer Count	4	
Bootstrap ID	0x60400104	(Max pointers)
Pointer 0	Reg 32	SMI Configuration
Pointer 1	Reg 32	SMI Status
Pointer 2	Reg 32	SMI Tx Buffer
Pointer 3	Reg 32	SMI Rx Buffer

TBD pointer descriptions

CAN Controller Feature (0x605)

TBD pointer table

TBD pointer descriptions

PWM Port Feature (0x606)

TBD pointer table

TBD pointer descriptions

GPIO Port Feature (0x607)

EEVideo devices may implement register-driven GPIO pins.

GPIO Controller

Feature ID	0x607	
Version	0x001	(0.1)
Pointer Count	4	
Bootstrap ID	0x60700104	(Max pointers)
Pointer 0	Reg 32	GPIO Port Config
Pointer 1	Reg 32	GPIO Direction

Pointer 2	Reg 32	GPIO In
Pointer 3	Reg 32	GPIO Out

TBD pointer descriptions

Marker Features

End of Features Marker (0x7FF)

The End of Features marker **must** be the last entry in the Feature Declaration List. It indicates to the host that there are no additional features implemented in the device.

End of Features

Feature ID	0x7FF	
Version	0x001	(0.1)
Pointer Count	2	
Bootstrap ID	0x7FF00102	(Max pointers)
Pointer 0	Reg 32	Last Static ROM Address
Pointer 1	Reg 32	Last Register Address

Last Static ROM Address

Points to the highest address register in the static ROM space. This register may not contain real data. It tells the host that all registers below this address are static/cacheable and do not need to be re-read.

Last Register Address

Points to the highest register number implemented in the device.

Registers above this address are invalid.

Custom Features

There will be application requirements that do not fit any of the standardized EEVideo features.

Custom features may be implemented, included in the Feature Manifest, and further defined in Register Detail Files.

Custom features **must** use a Feature ID value ≥ 0x800.

Custom features may follow the feature group numbering scheme using group + 0x800 for identifiers.

Changelog

This document records all notable changes to the EEVideo specification, following the Keep a Changelog format.

Unreleased

Added

- ...

Changed

- ...

Fixed

- ...

0.1.alpha-1 - 2025-10-02

Added

- Initial public alpha draft
- CoAP-based management protocol
- Lightweight streaming protocol

0.1.0 - 2026-03-13

Added

- Changelog
- Seperate Device Features heading
- Changed Device Features group numbering
- Added individual files for each feature
- Added feature groups
- Added test pattern feature
- Added Parametric Pixel Format option and details
- Added Appendix with sample type / sub-type definitions
- Refined CoAP option definitions

Fixed

- Some TBDs
- Grammer / Presentation

Template for future releases

When cutting a new release: 1. Update this file on the release branch. 2. Tag the commit: `git tag -a vX.Y.Z -m "Release vX.Y.Z"` 3. Push the tag: `git push origin vX.Y.Z`

License

MIT Open Source License

This Embedded Ethernet Video Protocol Specification (EEVideo) is licensed for use under the MIT license,

Appendices

This section contains reference material that supports the main specification.

Sample Types

Sample Types

This appendix defines the top-level sample type categories used throughout the EEVideo specification, particularly in the Parametric Pixel Format.

Sample Types

Type #	Name	Short	Typical use
0	Undefined	?	—
1	Monochrome	M	A sample from a specific range of the visible spectrum
2	Red	R	A sample from the red region of the spectrum
3	Green	G	A sample from the green region of the spectrum
4	Blue	B	A sample from the blue region of the spectrum
5	Luminance	Y	A composite sample generated from multiple raw samples
6	Red Chrominance	U	A red based color difference sample
7	Blue Chrominance	V	A blue based color difference sample
8	Alpha	A	A metadata component, common for compositing
9	Ultraviolet	UV	A sample from the ultraviolet region of the spectrum
10	Near IR	NI	A sample from the infrared region of the spectrum
11	SWIR	SI	A sample from the infrared region of the spectrum
12	MWIR	MI	A sample from the infrared region of the spectrum
13	LWIR	LI	A sample from the infrared region of the spectrum
14	Distance	DS	A distance measurement
15	Distance Conf	DC	A confidence indicator of the distance measurement
16-22	reserved		

Type #	Name	Short	Typical use
24-31	User Type 0-7	UT	User defined sample type
Others	reserved		

Sample Sub-Types

Type	Subtype	Definition
0 Undefined	0x000	Unknown sample type
1 Monochrome	0x000	Undefined
	0x001	Raw from sensor (linear)
	0x002	Gamma Corrected
2 Red	0x000	Undefined
	0x001	Raw from sensor (linear)
	0x002	Gamma Corrected
	0xF01	ITU-R BT.709 (sRGB)
3 Green	0x000	Undefined
	0x001	Raw from sensor (linear)
	0x002	Gamma Corrected
	0xF01	ITU-R BT.709 (sRGB)
4 Blue	0x000	Undefined
	0x001	Raw from sensor (linear)
	0x002	Gamma Corrected
	0xF01	ITU-R BT.709 (sRGB)
5 Luminance	0x000	Undefined
	0xF01	Y' ITU-R BT.709
6 Red Chrominance	0x000	Undefined
	0xF01	Cr ITU-R BT.709
	0xF02	Cr ITU-R BT.709 Half band Horizontal
	0xF03	Cr ITU-R BT.709 Half band H & V
7 Blue Chrominance	0x000	Undefined
	0xF01	Cb ITU-R BT.709
	0xF02	Cb ITU-R BT.709 Half band Horizontal
	0xF03	Cb ITU-R BT.709 Half band H & V

Type	Subtype	Definition
8 Alpha	0x000	Undefined
	0x001	Linear
9 Ultraviolet	0x000	Undefined
10 Near IR	0x000	Undefined
11 SWIR	0x000	Undefined
12 MWIR	0x000	Undefined
13 LWIR	0x000	Undefined
14 Distance	0x000	Undefined
15 Distance Conf	0x000	Undefined