

Understanding the eStreamer Application Protocol

The Secure Firewall System Event Streamer (eStreamer) uses a message-oriented protocol to stream events and host profile information to your client application. Your client can request event and host profile data from a Management Center, and intrusion event data only from a managed device. Your client application initiates the data stream by submitting request messages, which specify the data to be sent, and then controls the message flow from the Management Center or managed device after streaming begins.

Throughout this document, the eStreamer service on the Management Center or a managed device may be referred to as the eStreamer server or eStreamer.

The following sections describe requirements for connecting to the eStreamer service and introduce commands and data formats used in the eStreamer protocol:

- Connection Specifications, page 2-1 describes the communication flow between the eStreamer service and your client and describes how the client interacts with it.
- Understanding eStreamer Communication Stages, page 2-2 describes the communication protocol for client applications to submit data requests to the eStreamer server and for eStreamer to deliver the requested information to the client.
- Understanding eStreamer Message Types, page 2-8 describes the message types used in the eStreamer protocol; discusses the basic structure of data packets used by eStreamer to return intrusion event data, discovery event data, metadata, and host data to a client; and provides other information to help you write a client that can interpret eStreamer messages.

Connection Specifications

The eStreamer service:

- Communicates using TCP over an SSL connection (the client application must support SSL-based authentication).
- Accepts connection requests on port 8302.
- Waits for the client to initiate all communication sessions.
- Writes all message fields in network byte order (big endian).
- Encodes text in UTF-8.

Understanding eStreamer Communication Stages

There are four major stages of communication that occur between a client and the eStreamer service:

1. The client establishes a connection with the eStreamer server and the connection is authenticated by both parties.

   See Establishing an Authenticated Connection, page 2-2 for more information.

2. The client requests data from the eStreamer service and specifies the types of data to be streamed. A single event request message can specify any combination of available event data, including event metadata. A single host profile request can specify a single host or multiple hosts.

   Two request modes are available for requesting event data:

   - Event Stream Request — The client submits a message containing request flags that specify the requested event types and version of each type, and the eStreamer server responds by streaming the requested data.
   - Extended Request — The client submits a request with the same message format as for Event Stream requests but sets a flag for an extended request. This initiates a message interaction between client and eStreamer server through which the client requests additional information and version combinations not available via Event Stream requests.

   For information on requesting data, see Requesting Data from eStreamer, page 2-3.

3. eStreamer establishes the requested data stream to the client.

   See Accepting Data from eStreamer, page 2-7 for more information.

4. The connection terminates.
   See Terminating Connections, page 2-8 for more information.

Establishing an Authenticated Connection

Before a client can request data from eStreamer, the client must initiate an SSL-enabled TCP connection with the eStreamer service. The client can request on any configured management interface on the Management Center or managed device. Client connections do not enforce traffic channel configuration for management interfaces so that configuration can be ignored when choosing an interface for your connection. When the client initiates the connection, the eStreamer server responds, initiating an SSL handshake with the client. As part of the SSL handshake, the eStreamer server requests the client's authentication certificate, and verifies that the certificate is valid (signed by the Internal Certifying Authority [Internal CA] on the eStreamer server).

Note Cisco recommends that you also require your client to verify that the certificate presented by the eStreamer server has been signed by a trusted Certifying Authority. This is the Internal CA certificate included in the PKCS#12 file that Cisco provides when you register a new eStreamer client with the Management Center or managed device. See Adding Authentication for eStreamer Clients, page 6-3 for more information.

After the SSL session is established, the eStreamer server performs an additional post-connection verification of the certificate. This includes verifying that the client connection originates from the host specified in the certificate and that the subject name of the certificate contains the appropriate value. If either post-connection check fails, the eStreamer server closes the connection. If necessary, you can configure the eStreamer service so that it does not perform a client host name check (see eStreamer Service Options, page 6-4 for more information).

While the client is not required to perform post-connection verification, Cisco recommends that the client perform this verification step. The authentication certificate contains the following field values in the subject name of the certificate:

Table 2-1 Certificate Subject Name Fields

Field	Value
`title`	`eStreamer`
`generationQualifier`	`server`

After the post-connection verification is finished, the eStreamer server awaits a data request from the client.

Requesting Data from eStreamer

Your client performs the following high-level tasks in managing data requests:

- Initializing the request session — See Establishing a Session, page 2-3.
- Requesting events from the eStreamer event archive — Using Event Stream Requests and Extended Requests to Initiate Event Streaming, page 2-3.
- Requesting host data — See Requesting Host Data, page 2-7.
- Changing a request — See Changing a Request, page 2-7.
- Requesting Fully-Qualified Events — See Requesting Fully-Qualified Events, page 2-4

Establishing a Session

The client establishes a session by sending an initial Event Stream request to the eStreamer service.

In this initial message, you can either include data request flags or submit the data requests in a follow-on message. This initial Event Stream request message itself is a prerequisite for all eStreamer requests, whether for event data or for host data. For information about using the Event Stream request message, see Event Stream Request Message Format, page 2-12.

Note The eStreamer client can request on any configured management interface on the Management Center or managed device. Client connections do not enforce traffic channel configuration for management interfaces so that configuration can be ignored when choosing an interface for your connection.

Using Event Stream Requests and Extended Requests to Initiate Event Streaming

The eStreamer service provides two modes of requests for event streaming. Your request can combine modes. In both modes, your client starts the request with an Event Stream request message but sets the request flag bits differently. For details about the Event Stream message format, see Event Stream Request Message Format, page 2-12.

When eStreamer receives an Event Stream request message, it processes the client request as follows:

- If the request message does **not** set bit 30 in the request flag field, eStreamer begins streaming any events requested by other set bits in the request flag field. For information, see Submitting Event Stream Requests, page 2-4.

- If bit 30 **is** set in the Event Stream request, eStreamer provides extended request processing. Extended request flags must be sent if this bit is set. For information, see Submitting Extended Requests, page 2-4. Note that eStreamer resolves any duplicate requests. If you request multiple versions of the same data, either by multiple flags or multiple extended requests, the highest version is used. For example, if eStreamer receives flag requests for discovery events version 1 and 6 and an extended request for version 3, it sends version 6.

Submitting Event Stream Requests

Event stream requests use a simple process:

- Your client sends a request message to the eStreamer service with a start date and time and a request flag field that specifies the events and their version level to be included in the data stream.
- eStreamer streams events beginning at the specified time. For information about the streaming protocol, see Accepting Data from eStreamer, page 2-7.

For information on the format and content of the client's Event Stream request message, see Event Stream Request Message Format, page 2-12.

For information on the event types and versions of events that the client can request, see Table 2-6 on page 2-14.

Submitting Extended Requests

If you set bit 30 in the request flags field of an Event Stream Request message, you initiate an extended request, which starts a negotiation with the server. Extended request flags must be sent if this bit is set. For the event types available by extended request, see Table 2-22 on page 2-38.

The steps for extended requests are as follows:

- Your client sends an Event Streaming Request message to eStreamer with the request flags bit 30 set to `1`, which signals an extended request. See Event Stream Request Message Format, page 2-12 for message format details.
- eStreamer answers with a Streaming Information message that advertises the list of services available to the client. For details about the Streaming Information message, see Streaming Information Message Format, page 2-33.
- The client returns a Streaming Request message that indicates the service it wants to use, with a request list of event types and versions available from that service. The request list corresponds to setting bits in the request flag field when making a standard event stream request. For details about how to use the Streaming Request message to request events, see "Sample Extended Request Messages" section on page 2-40.
- eStreamer processes the client's Streaming Request message and begins streaming the data at the time specified in the message. For information about the streaming protocol, see Accepting Data from eStreamer, page 2-7.
-

Requesting Fully-Qualified Events

Instead of receiving events in the complicated binary format, we recommend that your client uses this option to request fully-qualified events in a text format such as JSON or CSV. When using this option, the majority of this document describing the binary format is irrelevant. In the SDK package, the python_client subdirectory provides sample code for using this option.

This option currently only supports requesting information for a few event types: connection events, intrusion events, intrusion packets, and file events. If you need to receive other event types in binary format, then separate client connections must be used for fully-qualified and binary event formats.

To request fully-qualified events, use the documented "Event Stream Request Message", and append a JSON-format configuration block at the end of the message. The request will include the usual five binary integers shown below, followed by the JSON-format configuration details, like:

```
<Header Version (1)>
<Message Type (2)>
<Message Length>
<Initial Timestamp>
<Request Flags>
<JSON-format Configuration Block>
```

The binary Message Length field must include the length of the binary header, plus the length of the JSON block. A terminating null character is optional after the JSON block, but if the null is included then the Message Length must account for the null character. For the Request Flags field, only bit 23 (extended event headers) is supported; all other bits should be zero, in particular bit 30 (extended request) must be zero.

After the client sends the request message, the eStreamer service will immediately start sending event data if the requested event types have been enabled on the server side UI eStreamer configuration page.

Format of the JSON File

This example can be also found in the `json_request.json` file in the eStreamer SDK.

```
{
"Events":
{
"ConnectionEvent":
{
"FieldSetDef":
{
"OutputFieldSet": ["HeaderFieldSet", "ConnectionKeySet", "DetailFieldSet"]
},
"Fields": ["OutputFieldSet"]
},
"IntrusionEvent":
{
"FieldSetDef":
{
"OutputFieldSet": ["HeaderFieldSet", "ConnectionKeySet", "DetailFieldSet", "Impact"]
},
"Fields": ["OutputFieldSet"]
},
"IntrusionPacket":
{
"FieldSetDef":
{
"OutputFieldSet": ["HeaderFieldSet", "DetailFieldSet"]
},
"Fields": ["OutputFieldSet"]
},
```

```
"FileEvent":
{
"FieldSetDef":
{
"OutputFieldSet": ["HeaderFieldSet", "ConnectionKeySet", "DetailFieldSet"]
},
"Fields": ["OutputFieldSet"]
}
},
"OutputFormat":
{
"Transform": "Text",
"TransformConfig": "JSON"
}
}
```

In the `Events` section, specify a block for each event type that you would like the client to receive (only the three example types are supported: `ConnectionEvent`, `IntrusionEvent`, `IntrusionPacket,` and `FileEvent`). The `FieldSetDef` section for each event must specify an `OutputFieldSet`, which lists the fields or field sets which will be included in the events for that event type. The sample file only specifies field sets, but you can use any combination of field names and field sets.

The list of available fields for each event type, and the predefined field sets, can be found on the Firepower Management Center in the file `/etc/sf/EventHandler/EventCatalog/EventCatalog.json`. In the Fields section towards the end of the file, look for the desired event type (such as `IntrusionEvent`), then see the `Fields` and `FieldSetDef` blocks to see what is available for that event type.

The `OutputFormat` section has settings for the output. The `Transform` field is always `Text`, and you specify the output transformation format with the `TransformConfig` field. The example shows `JSON`, but you can also specify `CSV`. Other text formats are available, as well as `FlatBuffer`, but you will need to request documentation for these formats.

When JSON output is specified in `TransformConfig`, the output will contain name-value pairs for each requested field, except any fields which are irrelevant to the event are skipped (e.g. if you requested SSL fields, and an event did not use SSL, then the output will not contain those fields).

When CSV output is specified in `TransformConfig`, the output will contain the desired fields in the order listed in the configuration. If a field is not relevant to the event then the CSV will only contain a comma for that field. Do not use predefined field sets when requesting CSV because the field sets may change between versions, making the CSV incompatible.

Fully-Qualified Event Messages

Event messages are contained in bundles, as described in the eStreamer documentation for "Message Bundle Format", message type `4002`.

As documented, the client must acknowledge each received data bundle by sending a null message to the eStreamer server, indicating readiness to accept more data.

For all supported event types, the event data message starts with the binary header that is described in the eStreamer documentation for various event types, such as the "Correlation Record Header". The only difference is that the data block format is the requested format (JSON, CSV, etc.). For quick reference the basic structure is:

```
<Header Version (1)>
<Message Type (3)>
<Message Length>
<Record Type (with optional Netmap ID when requested)>
<Record Length>
<Timestamp (when request bit 23 is specified)>
<Reserved (when request bit 23 is specified)>
<Data>
```

Requesting Host Data

Once you have established a session, you can submit a request for host data at any time. eStreamer generates information for the requested hosts from the Secure Firewall System network map.

Changing a Request

To change request parameters for an established session, the client must disconnect and request a new session.

Accepting Data from eStreamer



Note The eStreamer server does not keep a history of the events it sends. Your client application must check for duplicate events, which can inadvertently occur for a number of reasons. For example, when starting up a new streaming session, the time specified by the client as the starting point for the new session can have multiple messages, some of which may have been sent in the previous session and some of which were not. eStreamer sends all message that meet the specified request criteria. Your application should detect any resulting duplicates.

During periods of inactivity, eStreamer sends periodic null messages to the client to keep the connection open. If it receives an error message from the client or an intermediate host, it closes the connection.

eStreamer transmits requested data to the client differently, depending on the request mode.

Event Stream Requests

If the client submits an event stream request, eStreamer returns data message by message. It may send multiple messages in a row without waiting for a client acknowledgment. At a certain point, it pauses and waits for the client. The client operating system buffers received data and lets the client process it at its own pace.

If the client request includes a request for metadata, eStreamer sends the metadata first. The client should store it in memory to be available when processing the event records that follow.

Extended Requests

If the client submits an extended request, eStreamer queues up messages and sends them in bundles. eStreamer may send multiple bundles in a row without waiting for a client acknowledgment. At a certain point, it pauses and waits for the client. The client operating system buffers received data and lets the client read it off at its own pace.

The client unpacks each bundle, message by message, and uses the lengths of the records and the blocks to parse each message. The overall message length in each message header can be used to calculate when the end of each message has been reached, and the overall bundle length can be used to know when the end of the bundle is reached. The bundle requires no index of its contents to be correctly parsed.

For information about the message bundling mechanism, see Message Bundle Format, page 2-41.

For information about the null message that the client can use for additional flow control, see Null Message Format, page 2-10.

Terminating Connections

The eStreamer server attempts to send an error message before closing the connection. For information on error messages, see Error Message Format, page 2-11.

The eStreamer server can close a client connection for the following reasons:

- Any time sending a message results in an error. This includes both event data messages and the null keep-alive message eStreamer sends during periods of inactivity.
- An error occurs while processing a client request.
- Client authentication fails (no error message is sent).
- eStreamer service is shutting down (no error message is sent).

Your client can close the connection to eStreamer server at any time and should attempt to use the error message format to notify the eStreamer server of the reason.

Understanding eStreamer Message Types

The eStreamer application protocol uses a simple message format that includes a standard message header and various sub-header fields followed by the record data which contains the message's payload. The message header is the same in all eStreamer message types; for more information, see eStreamer Message Header, page 2-9.

Table 2-2 eStreamer Message Types

Message Type	Name	Description
0	Null message	Both the eStreamer server and the client send null messages to control data flow. For information, see Null Message Format, page 2-10.
1	Error message	Both the eStreamer server and the client use error messages to indicate why a connection closed. For information, see Error Message Format, page 2-11.

Table 2-2 eStreamer Message Types (continued)

Message Type	Name	Description
2	Event Stream Request	A client sends this message type to the eStreamer service to initiate a new streaming session and request data. For information, see Event Stream Request Message Format, page 2-12.
4	Event Data	The eStreamer service uses this message type to send event data and metadata to the client. For information, see Event Data Message Format, page 2-19.
5	Host Data Request	A client sends this message type to the eStreamer service to request host data. A session must be started already via an Event Stream Request message. For information, see Host Request Message Format, page 2-27.
6	Single Host Data	The eStreamer service uses this message type to send single host data requested by the client. For information, see Host Data and Multiple Host Data Message Format, page 2-32.
7	Multiple Host Data	The eStreamer service uses this message type to send multiple host data requested by the client. For information, see Host Data and Multiple Host Data Message Format, page 2-32.
2049	Streaming Request	A client uses this message type in extended requests to specify which of the advertised events from the Stream Information message it wants. For information, see Sample Extended Request Messages, page 2-40.
2051	Streaming Information	The eStreamer service uses this message type in extended requests to advertise the list of services available to the client. For information, see Streaming Information Message Format, page 2-33.
4002	Message Bundle	The eStreamer service uses this message type to package messages that it streams to clients. For information, see Message Bundle Format, page 2-41.

eStreamer Message Header

All eStreamer messages start with the message header illustrated in the graphic below. The following table explains the fields.

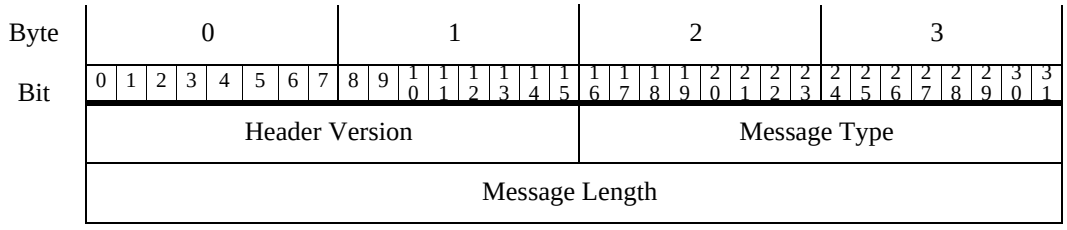


Table 2-3 Standard eStreamer Message Header Fields

Field	Data Type	Description
Header Version	uint16	Indicates the version of the header used on the message. For the current version of eStreamer, this value is always `1`.
Message Type	uint16	Indicates the type of message transmitted. For the list of current values, see Table 2-2 on page 2-8.
Message Length	uint32	Indicates the length of the content that follows, and excludes the bytes in the message header itself. A message with a header and no data has a message length of zero.

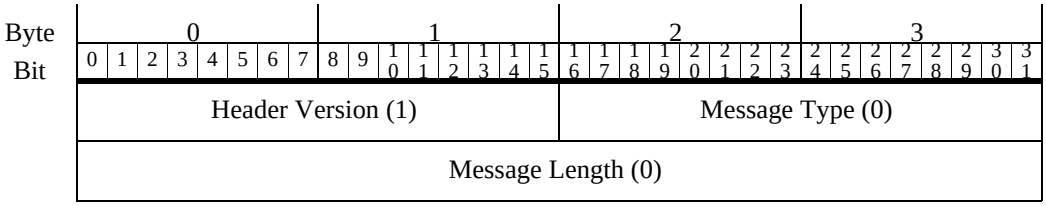
Null Message Format

Both the client application and the eStreamer service send null messages. The null message has a type of 0 and contains no data after the message header.

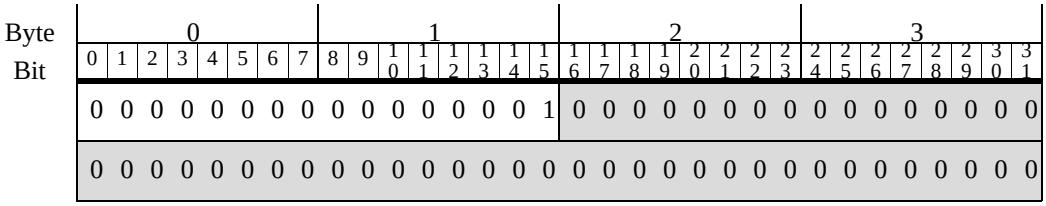
The client sends a null message to the eStreamer server to indicate readiness to accept more data. The eStreamer service sends null messages to the client to keep the connection alive when no data is being transmitted. The message length value for null messages is always set to `0`.

Tip In data structure diagrams in this book, integers in parentheses such as (`1`) or (`115`) represent constant field values. For example, Header Version (`1`) means that the field in the data structure under discussion always has a value of `1`.

The Null message format is shown below. The only non-zero value in the message is the header version.



An example of a null message in binary format follows. Notice that the only non-zero value is in the second byte, signifying a header version value of `1`. The message type and length fields (shaded) each have a value of `0`.



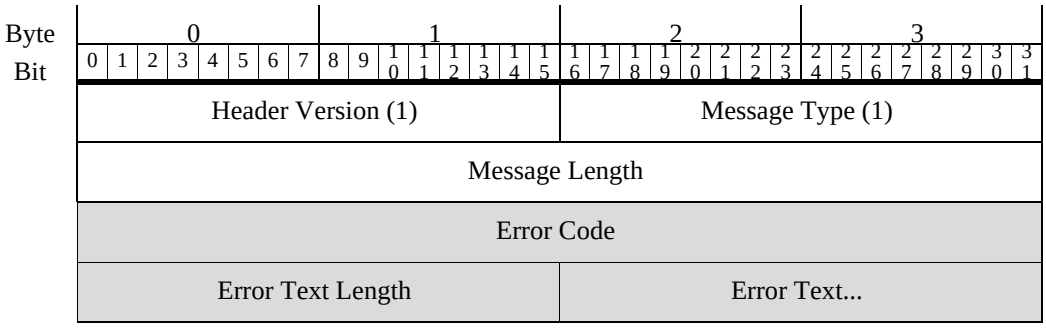
Tip Examples in this guide appear in binary format to clearly display which bits are set. This is important for some messages, such as the event request message and event impact fields.

Error Message Format

Both the client application and the eStreamer service use error messages. Error messages have a message type of 1 and contain a header, an error code, an error text length, and the actual error text. Error text can contain between 0 and 65,535 bytes.

When you create custom error messages for your client application, Cisco recommends using `-1` as the error code.

The following graphic illustrates the basic error message format. Shaded fields are specific to error messages.

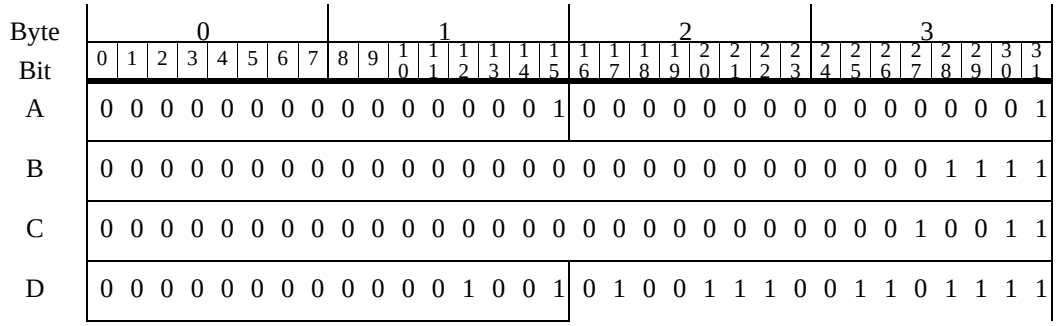


The following table describes each field in error code messages.

Table 2-4 Error Message Fields

Field	Data Type	Description
Error Code	int32	A number representing the error.
Error Text Length	uint16	The number of bytes included in the error text field.
Error Text	variable	The error message. Up to 65,535 bytes.

The following diagram shows an example error message:



```
0 0 1 0 0 0 0 0 0 1 1 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 1
0 1 1 0 0 0 1 1 0 1 1 0 0 1 0 1
```

In the preceding example, the following information appears:

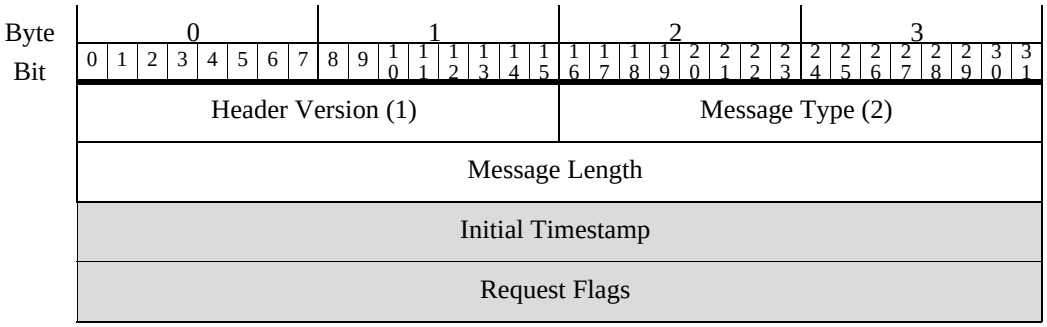
Letter	Description
A	The first two bytes indicate the standard header value of `1`. The second two bytes show a value of 1, which signifies that the transmission is an error message.
B	This line indicates the amount of message data that follows it. In this example, 15 bytes (in binary, `1111`) of data follow.
C	This line displays the error code. In this example, the message contains a value of `19` (`10011`). Therefore, error number 19 is transmitted in the message.
D	This line contains the number of bytes in the error message (`1001`, or nine bytes), and the error message itself follows in the next nine bytes. The error message value, when converted to ASCII text, equals “No space,” which is the error message that accompanies error code 19.

Event Stream Request Message Format

eStreamer clients use the Event Stream Request message to start a streaming session. The request message includes a start time and a bit flag field to specify the data the eStreamer service should include, which can be any combination of events, as well as intrusion event extra data and metadata. The Event Stream Request message can initiate both event stream requests and extended requests. The message type is 2.

You must submit an Event Stream Request message for all data requests, including a request exclusively for host profile information. In such a case, you first submit an Event Stream Request message, then a Host Request message (type 5) to specify the host data.

The following graphic illustrates the Event Stream Request message format. The message uses the standard header. The shaded fields are specific to the request message and are described in the following table.



The following table describes each field in Event Stream Request messages.

Table 2-5 Event Stream Request Message Fields

Field	Data Type	Description
Initial Timestamp	uint32	Defines the start of the session. To start at: • the time the client connects to eStreamer, set all timestamp bits to 1. • the oldest data available, set all timestamp bits to zero. • a given date and time, specify the UNIX timestamp (number of seconds since January 1, 1970). See Initial Timestamp, page 2-13 below for important information.
Request Flags	bits[32]	Specifies the types and versions of events and metadata to be returned in event stream requests. See Request Flags, page 2-13 for flag definitions. Setting bit 30 initiates an extended request, which can co-exist with event stream requests in the same message.

Initial Timestamp

Note Your client application should use the archival timestamp in the Initial Timestamp field when submitting an event stream request, as explained below. This ensures that you do not inadvertently exclude events. Devices transmit data to the Management Center using a "store and forward" mechanism with transmission delays. If you request events by the generation timestamp assigned by the device that detects it, delayed events may be missed.

When starting a session, a best practice is to start up from the archival timestamp (also known as the "server timestamp") of the last record in the previous session. It is not a technical requirement but is strongly recommended. By using the archival timestamp of the last record in the previous session, the eStreamer service will not resend prior records or metadata. Under certain circumstances, if you use the generation timestamp you can inadvertently exclude events from the new streaming session.

To include the archival timestamp in your streamed events, you must set bit 23 in the request flag field.

Note that only time-based events have archival timestamps. Events that eStreamer generates, such as metadata, have zero in this field when extended event headers have been requested with bit 23 set.

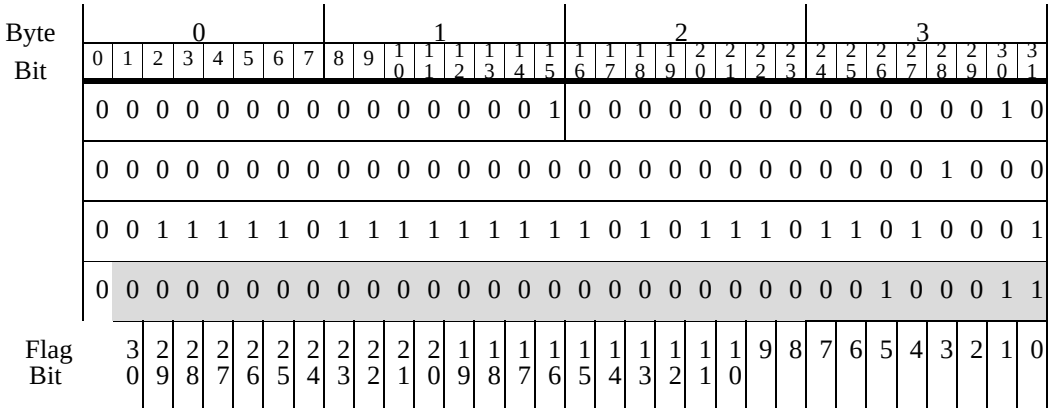
Request Flags

You set bits 0 through 29 in the event data request flag field to select the types of events you want eStreamer to send. You set bit 30 to activate the extended request mode. Setting bit 30 does not directly request any data. Extended request flags must be sent if this bit is set. Your client requests data during the server-client message dialog that follows submission of the Event Stream Request message. For information on extended requests, see Requesting Data from eStreamer, page 2-3.

See Table 2-6 on page 2-14 for definitions of the bit settings in the Request Flags field. Different flags request different versions of the event data. For example, to obtain data in Secure Firewall System 4.9 format instead of 4.10 format you set a different flag bit. For specific information on the flags to use when requesting data for particular product versions, see Table 2-7 on page 2-17.

Note that you request metadata by version, not by the individual metadata record. For information about each supported version of metadata, see Request Flags, page 2-13.

The following diagram shades the bits in the flags field that are currently used:



For information on each request flag bit, see the following table.

Table 2-6 Request Flags

Bit Field	Description
Bit 0	Requests the transmission of packet data associated with intrusion events. If set to `1`, packet data is transmitted with intrusion events. If set to `0`, packet data is not transmitted.
Bit 1	Requests the transmission of version 1 metadata associated with intrusion, discovery, correlation, and connection events. If set to `1`, version 1 metadata is transmitted with events. If set to `0`, version 1 metadata is not transmitted. You can use metadata to resolve coded and numeric fields in events. See Understanding Metadata, page 2-42 for general information on the way eStreamer transmits metadata to clients and how a client can use metadata.
Bit 2	Requests the transmission of intrusion events. If bit 2, bit 6, or both bit 2 and 6 are set to `1`, but the extended request flag, bit 30, is set to `0`, the system interprets this as a request from a Version 4.x client and record type 104/105 is sent. If no event type is specified when bit 2, bit 6, or both bit 2 and 6 are set to `1`, and bit 30 is set to `1`, the system interprets this as a request from a Version 5.0-5.1 client and record type 207/208 is sent. If bit 30 is set to `1`, and a specific event type is requested, intrusion events are sent regardless of bits 2 and 6. For details on requesting record types, see Submitting Extended Requests, page 2-4. If bit 2, bit 6, and bit 30 are all set to `0`, intrusion events are not sent. Bit 6 is used in a manner identical to bit 2. Either bit can be set to request intrusion events. Setting one of these bits to `0` will not override the other bit; setting bit 2 to `0` and bit 6 to `1`, or setting bit 2 to `1` and bit 6 to `0`, will be interpreted as a request for intrusion events.
Bit 3	Requests the transmission of discovery data version 1 (Management Center 3.2). If set to `0`, discovery data version 1 is not transmitted. For more information about discovery events, see Understanding Discovery & Connection Data Structures, page 4-1.
Bit 4	Requests the transmission of correlation data version 1 (Management Center 3.2). If set to `0`, correlation data version 1 is not transmitted.

Table 2-6 Request Flags (continued)

Bit Field	Description
Bit 5	Requests the transmission of impact correlation events (intrusion impact alerts). If set to `1`, intrusion impact alerts are transmitted. If set to `0`, intrusion impact alerts are not transmitted. See Intrusion Impact Alert Data 5.3+, page 3-18 for more information about intrusion impact alerts.
Bit 6	Bit 6 is used in a manner identical to bit 2. See Bit 2, page 2-14.
Bit 7	Requests the transmission of discovery data version 2 (Management Center 4.0 - 4.1) if set to `1`. If set to `0`, discovery data version 2 is not transmitted.
Bit 8	Requests the transmission of connection data version 1 (Management Center 4.0 - 4.1) if set to `1`. If set to `0`, connection data version 1 is not sent.
Bit 9	Requests the transmission of correlation data version 2 (Management Center 4.0 - 4.1.x) if set to `1`. If set to `0`, correlation policy data version 2 is not transmitted.
Bit 10	Requests the transmission of discovery data version 3 (Management Center 4.5 - 4.6.1) if set to `1`. If set to `0`, discovery data version 3 is not transmitted. For more information about legacy discovery events, see Legacy Discovery Data Structures, page B-119.
Bit 11	Disables transmission of events.
Bit 12	Requests the transmission of connection data version 3 (Management Center 4.5 - 4.6.1) if set to `1`. If set to `0`, connection data version 3 is not sent.
Bit 13	Requests the transmission of correlation data version 3 (Management Center 4.5 - 4.6.1). If set to `0`, correlation data version 3 is not transmitted.
Bit 14	Requests the transmission of version 2 metadata associated with intrusion, discovery, correlation, and connection events. If set to `1`, version 2 metadata is transmitted with events. If set to `0`, version 2 metadata is not transmitted. See Understanding Metadata, page 2-42 for general information on the way eStreamer transmits metadata to clients and how a client can use metadata.
Bit 15	Requests the transmission of version 3 metadata associated with intrusion, correlation, discovery, and connection events. If set to `1`, version 3 metadata is transmitted with events. If set to `0`, version 3 metadata is not transmitted. See Understanding Metadata, page 2-42 for general information on the way eStreamer transmits metadata to clients and how a client can use metadata.
Bit 16	Unused
Bit 17	Requests the transmission of discovery data version 4 (Management Center 4.7 - 4.8.x). If set to `0`, discovery data version 4 is not transmitted.
Bit 18	Requests the transmission of connection data version 4 (Management Center 4.7 - 4.9.0.x) if set to `1`. If set to `0`, connection data version 4 is not sent. See Connection Chunk Message, page 4-54 for more information.
Bit 19	Requests the transmission of correlation data version 4 (Management Center 4.7). If set to `0`, correlation data version 4 is not transmitted. See Legacy Correlation Event Data Structures, page B-328 for information about correlation events transmitted in Management Center 4.7 format.

Table 2-6 Request Flags (continued)

Bit Field	Description
Bit 20	Requests the transmission of version 4 metadata associated with intrusion, discovery, user activity, correlation, and connection events. If set to `1`, version 4 metadata is transmitted with events. If set to `0`, version 4 metadata is not transmitted. Version 4 metadata includes the following: • correlation (compliance) rule information • correlation (compliance) policy information • fingerprint records • client application records • client application type records • vulnerability records • host criticality records • network protocol records • host attribute records • scan type records • user records • service detection device (version 2) records • event classification (version 2) records • priority records • rule information (version 2) • malware information If you request bit 20 with bit 22, user metadata is also sent. See Understanding Metadata, page 2-42 for general information on the way eStreamer transmits metadata to clients and how a client can use metadata.
Bit 21	Requests the transmission of version 1 user events. For more information on user events, see User Record, page 4-19.
Bit 22	Requests the transmission of correlation data version 5 (Management Center 4.8.0.2 - 4.9.1). If set to `0`, correlation data version 5 is not transmitted. If you request bit 20 with bit 22, user metadata is also sent. For more information about legacy correlation (compliance) events, see Legacy Correlation Event Data Structures, page B-328.
Bit 23	Requests extended event headers. If set to `1`, events are transmitted with the timestamp applied when the event was archived for the eStreamer server to process and four bytes reserved for future use. If this field is set to `0`, events are sent with a standard event header that only includes the record type and record length. See eStreamer Message Header, page 2-9 for information about the event message header.
Bit 24	Requests the transmission of discovery data version 5 (Management Center 4.9.0.x). If set to `0`, discovery data version 5 is not transmitted. For more information about discovery events, see Understanding Discovery & Connection Data Structures, page 4-1.

Table 2-6 Request Flags (continued)

Bit Field	Description
Bit 25	Requests the transmission of discovery data version 6 (Management Center 4.9.1+). If set to `0`, discovery data version 6 is not transmitted. For more information about discovery events, see Understanding Discovery & Connection Data Structures, page 4-1.
Bit 26	Requests the transmission of connection data version 5 (Management Center 4.9.1 - 4.10.x) if set to `1`. If set to `0`, connection data version 5 is not sent. See Connection Chunk Message, page 4-54 for more information.
Bit 27	Requests event extra data associated with an intrusion event in an Extra Data record. For more information about event data, see Table B-11Intrusion Event Extra Data Data Block Fields, page B-66.
Bit 28	Requests the transmission of discovery data version 7 (Management Center 4.10.0+). If set to `0`, discovery data version 7 is not transmitted. For more information about discovery events, see Understanding Discovery & Connection Data Structures, page 4-1.
Bit 29	Requests the transmission of correlation data version 6 (Management Center 4.10 - 4.10.x). If set to `0`, correlation policy data version 6 is not transmitted. If you request bit 20 with bit 29, user metadata is also sent. For more information about correlation events, see earlier versions of the product.
Bit 30	Indicates an extended request to eStreamer. Extended request flags must be sent if this bit is set. For information about extended requests, see Submitting Extended Requests, page 2-4.

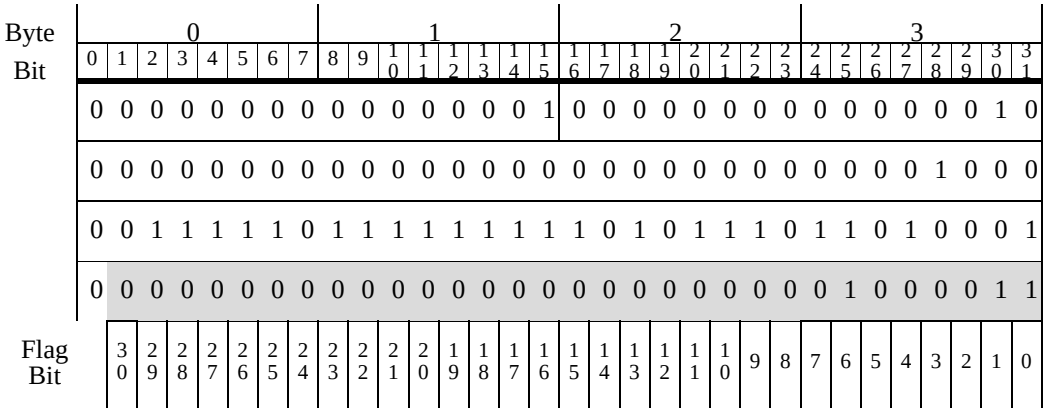
To help you decide which flags to use to request data for a particular version, see the following table. For Version 5.0 and later, see Submitting Extended Requests, page 2-4 for more information about using Bit 30.

Table 2-7 Event Request Flags by Product Version

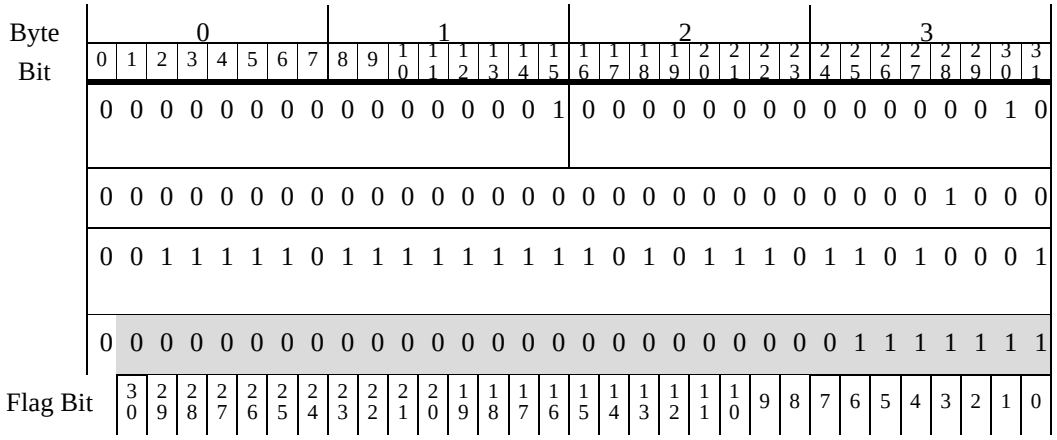
Type of Requested Data	4.9.0.x	4.9.1.x	4.10.x	5.0+	5.1	5.1.1+
packet data	Bit 0	Bit 0	Bit 0	Bit 0	Bit 0	Bit 0
intrusion events	Bit 2	Bit 2	Bit 2	Bit 2	Bit 2	Bit 30
metadata	Bit 20	Bit 20	Bit 20	Bit 20	Bit 20	Bit 20
discovery events	Bit 24	Bit 25	Bit 28	Bit 30	Bit 30	Bit 30
correlation events	Bit 22	Bit 22	Bit 29	Bit 30	Bit 30	Bit 30
event extra data	—	—	Bit 27	Bit 27	Bit 27	Bit 27
impact event alerts	Bit 5	Bit 5	Bit 5	Bit 5	Bit 5	Bit 5
connection data	Bit 18	Bit 26	Bit 26	Bit 30	Bit 30	Bit 30
user events	Bit 21	Bit 21	Bit 21	Bit 30	Bit 30	Bit 30
malware events	—	—	—	—	—	Bit 30
file events	—	—	—	—	—	Bit 30

Caution In all event types, prior to version 5.x, the reference client labels `detection engine ID` fields as `sensor ID`.

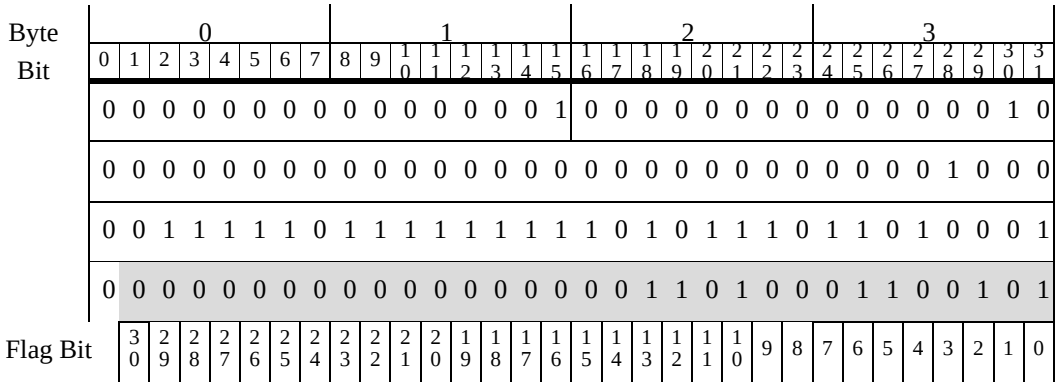
The following example requests intrusion events of type 7 (compatible with Secure Firewall System 3.2+) with both version 1 metadata and packet flags:



To request only data compatible with Secure Firewall System 3.2 (including intrusion events, packets, metadata, impact alerts, policy violation events, and version 2.0 events), use the following:



To request intrusion impact alerts, correlation events, discovery events, connection events, and intrusion events of type 7 with packets and version 3 metadata in Management Center 4.6.1+ format, use the following:



Event Data Message Format

The eStreamer service transmits event data and related metadata to clients when it receives an event request. Event data messages have a message type of 3. Each message contains a single data record with either event data or metadata.

Note that type 3 messages carry only event data and metadata. eStreamer transmits host information in type 6 (single-host) and type 7 (multiple-host) messages. See Host Data and Multiple Host Data Message Format, page 2-32 for information on host message formats.

Understanding the Organization of Event Data Messages

The event data and metadata messages that eStreamer sends contain the following sections:

- eStreamer message header — The standard message header defined at eStreamer Message Header, page 2-9.
- Event-specific sub-headers — Sets of fields that vary by event type, with codes that describe additional event details and determine the structure of the payload data that follows.
- Data record — Fixed-length fields and a data block.

Note The client should unpack all messages on the basis of field length.

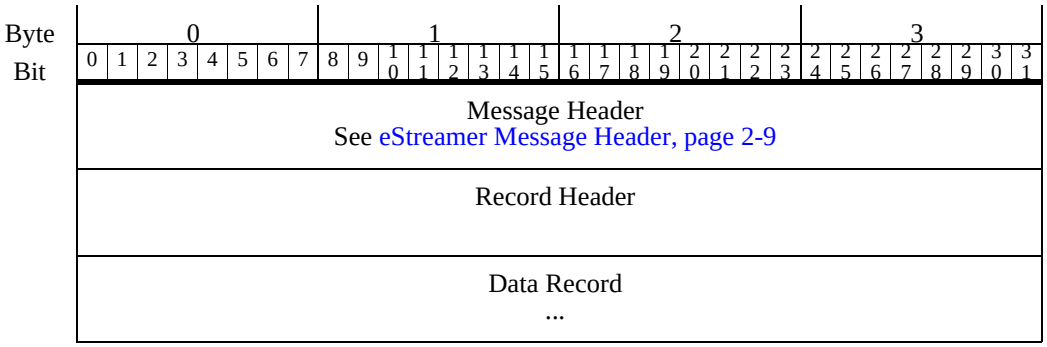
For the event message formats by event type, see the following:

- Intrusion Event and Metadata Message Format, page 2-20 for intrusion event data records and all metadata records. These messages have fixed-length fields.
- Discovery Event Message Format, page 2-21 for messages with discovery event or user event data. In addition to the standard eStreamer message header and a record header similar to the intrusion event message, discovery messages have a distinctive discovery event header with an event type and subtype field. The data record in discovery event messages is packaged in a series 1 block that can have variable length fields and multiple layers of encapsulated blocks.
- Connection Event Message Format, page 2-23 for messages with connection statistics. Their general structure is identical to discovery event messages. Their data block types, however, are specific for connection statistics.

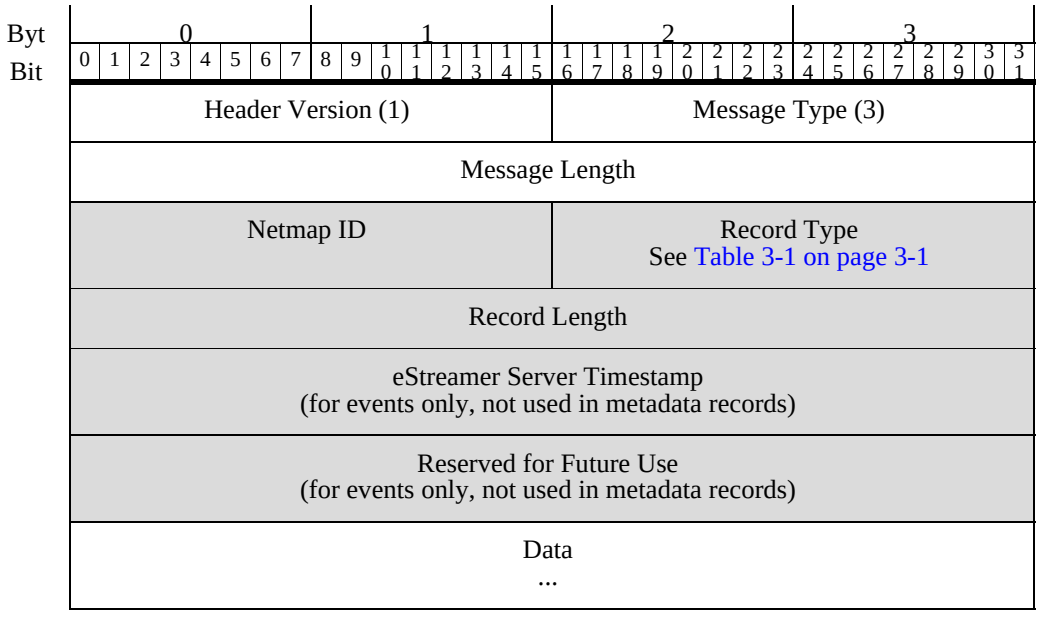
- Correlation Event Message Format, page 2-23 for messages with correlation (compliance) event data. The headers in these messages are the same as in intrusion event messages but the data blocks are series 1 blocks.
- Event Extra Data Message Format, page 2-25 for a series of messages that deliver intrusion-related record types with variable-length fields and multiple layers of nested data blocks such as intrusion event extra data. See Event Extra Data Message Format, page 2-25 for general information on the structure of this message series. See Data Block Header, page 2-26 for information about the structures of this series of blocks which are similar to series 1 blocks but numbered separately.

Intrusion Event and Metadata Message Format

The graphic below shows the general structure of intrusion event and metadata messages.



The following graphic shows the details of the record header portion of the intrusion event and metadata message format. The record header fields are shaded. The table that follows defines the fields.



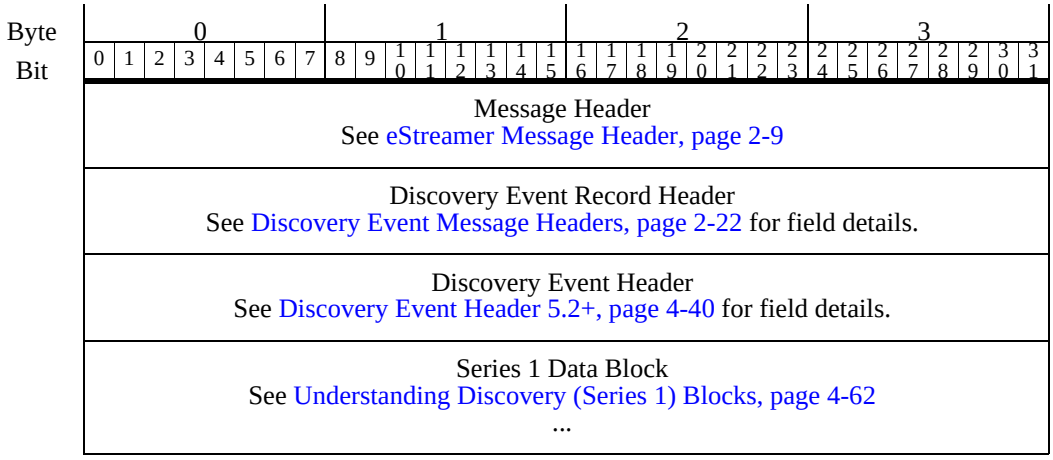
The following table describes each field in the header of intrusion events and metadata messages.

Table 2-8 Intrusion Event and Metadata Record Header Fields

Field	Data Type	Description
Netmap ID	uint16	The first bit of this field is a flag indicating whether the header is an extended header containing an archive timestamp. The remaining 15 bits are an optional field containing the Netmap ID for the domain on which the event was detected. If this field is not used, it is left empty. Netmap IDs map to domains as provided in metadata.
Record Type	uint16	Identifies the data record content type. See Table 3-1Intrusion Event and General Metadata Record Types, page 3-1 for the list of record types.
Record Length	uint32	Length of the content of the message after the record header. Does not include the 8 or 16 bytes of the record header. (Record Length plus the length of the record header equals Message Length.)
eStreamer Server Timestamp	uint32	Indicates the timestamp applied when the event was archived by the eStreamer server. Also called the archival timestamp. Field present only if bit 23 is set in the request message flags.
Reserved for future use	uint32	Reserved for future use. Field present only if bit 23 is set in the request message flags.

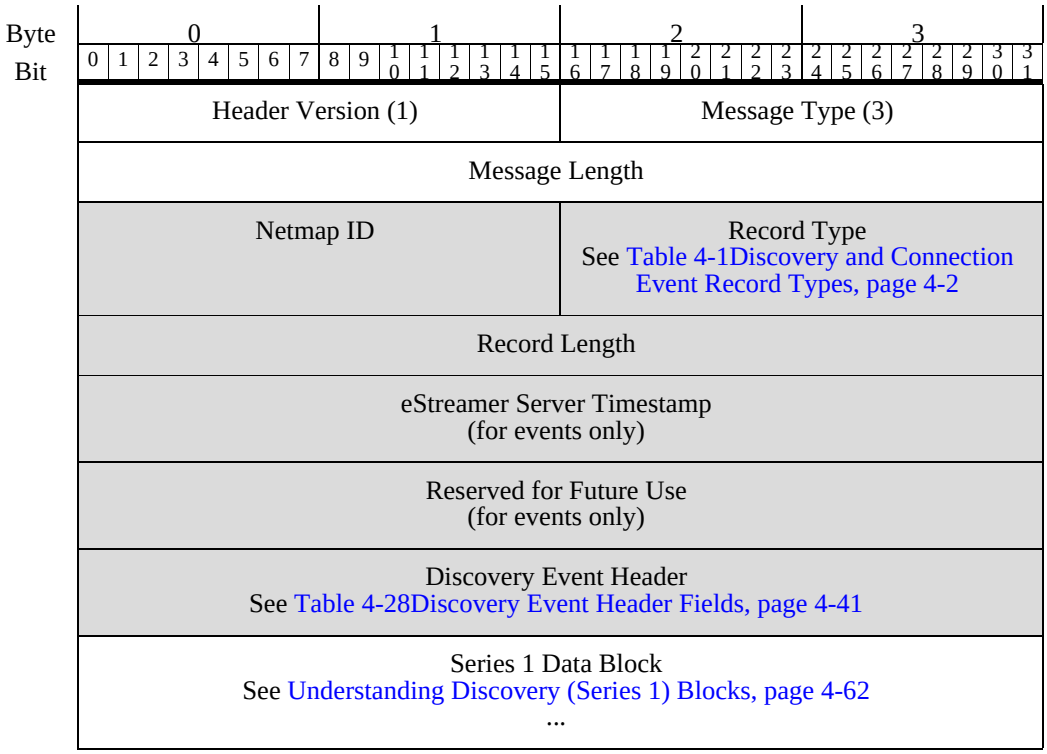
Discovery Event Message Format

The graphic below shows the structure of discovery event messages. The standard eStreamer message header and event record header are followed by a discovery event header used only in discovery and user event messages. The discovery event header section of the message contains the discovery event type and subtype fields, which together form a key to the data block that follows. For the current discovery event types and subtypes, see Table 4-29Discovery and Connection Events by Type and Subtype, page 4-42.



Discovery Event Message Headers

The shaded section in the following graphic shows the fields of the record header in the discovery event data message format, and shows the location of the event header that follows it. The following table defines the fields of the discovery event message headers.



The following table describes the fields in the record header and the event header of the discovery event message.

Table 2-9 Discovery Event Message Header Fields

Field	Data Type	Description
Netmap ID	uint16	The first bit of this field is a flag indicating whether the header is an extended header containing an archive timestamp. The remaining 15 bits are an optional field containing the Netmap ID for the domain on which the event was detected. If this field is not used, it is left empty. Netmap IDs map to domains as provided in metadata.
Record Type	uint16	Identifies the data record content type. See Table 4-1Discovery and Connection Event Record Types, page 4-2 for the list of record types.
Record Length	uint32	Length of the content of the message after the record header. Does not include the 8 or 16 bytes of the record header. (Record Length plus the length of the record header equals Message Length.)

Table 2-9 Discovery Event Message Header Fields (continued)

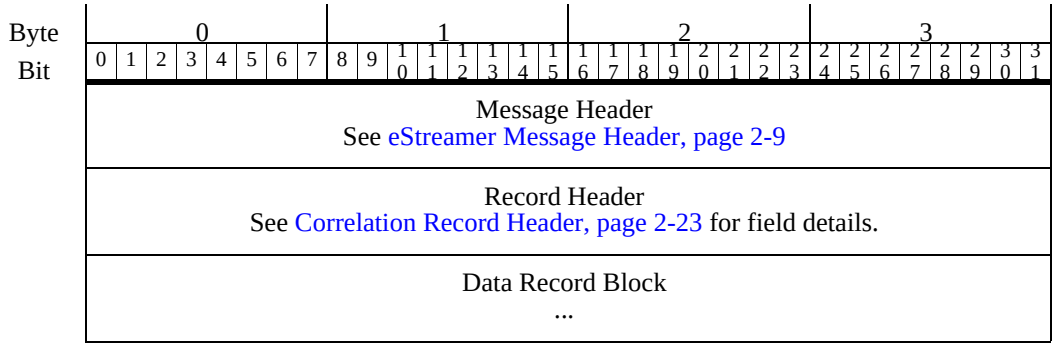
Field	Data Type	Description
eStreamer Server Timestamp	uint32	Indicates the timestamp applied when the event was archived by the eStreamer server. Also called the archival timestamp. Field present only if bit 23 is set in the request flags field of the event stream request.
Reserved for future use	uint32	Reserved for future use. Field present only if bit 23 is set in the request message flags.
Discovery Event Header	Varied	Contains a number of fields, including the event type and subtype, which together form a unique key to the data structure that follows. See Discovery Event Header 5.2+, page 4-40 for definitions of fields in the discovery event header.

Connection Event Message Format

Messages with connection statistics have a structure identical to discovery event messages. See Discovery Event Message Format, page 2-21 for general message format information. Connection event messages are distinct in terms of the data block types they incorporate.

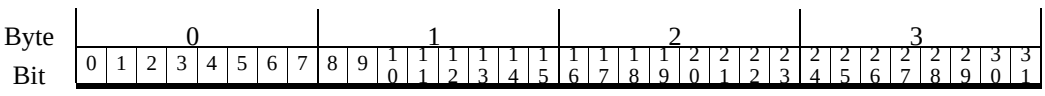
Correlation Event Message Format

The graphic below shows the general structure of correlation (compliance) event messages. The standard eStreamer message header and record header are followed immediately by a data block in the data record section of the message. Correlation messages use Series 1 data blocks.



Correlation Record Header

The shaded section of the following graphic shows the fields of the record header in correlation event messages. Note that correlation messages use series 1 data blocks; however, they do not have the discovery header that appears in discovery event messages. Their header fields resemble those of intrusion event messages. The table that follows the graphic below defines the record header fields for correlation events.



Header Version (1)	Message Type (3)
Message Length	
Netmap ID	Record Type See Table 3-1Intrusion Event and General Metadata Record Types, page 3-1
Record Length	
eStreamer Server Timestamp (for events only, not used in metadata records)	
Reserved for Future Use (for events only, not used in metadata records)	
Data Record Block Uses Series 1 block, see Understanding Discovery (Series 1) Blocks, page 4-62 ...	

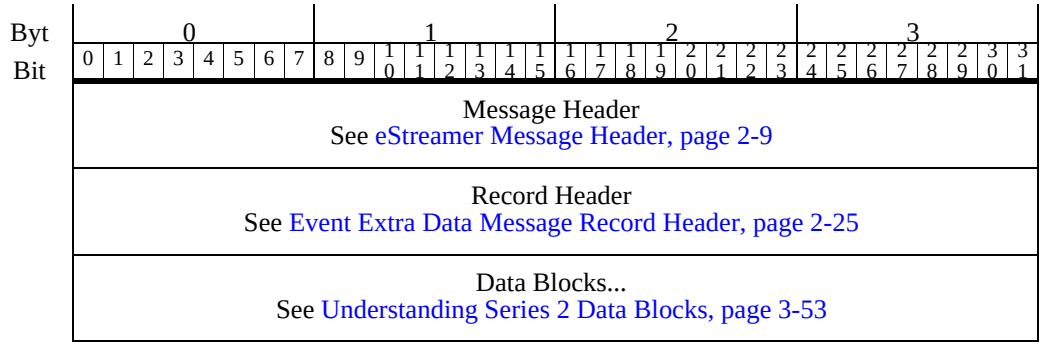
The following table describes each field in the record header of correlation event messages.

Table 2-10 Correlation Event Message Record Header Fields

Field	Data Type	Description
Netmap ID	uint16	The first bit of this field is a flag indicating whether the header is an extended header containing an archive timestamp. The remaining 15 bits are an optional field containing the Netmap ID for the domain on which the event was detected. If this field is not used, it is left empty. Netmap IDs map to domains as provided in metadata.
Record Type	uint16	Identifies the data record content type. See Table 3-1 on page 3-1 for the list of intrusion, correlation, and metadata record types.
Record Length	uint32	Length of the content of the message after the record header. Does not include the 8 or 16 bytes of the record header. (Record Length plus the length of the record header equals Message Length.)
eStreamer Server Timestamp	uint32	Indicates the timestamp applied when the event was archived by the eStreamer server. Also called the archival timestamp. Field present only if bit 23 is set in the request message flags. Field is zero for data generated by the Management Center such as host profiles and metadata.
Reserved for future use	uint32	Reserved for future use. Field present only if bit 23 is set in the request message flags.

Event Extra Data Message Format

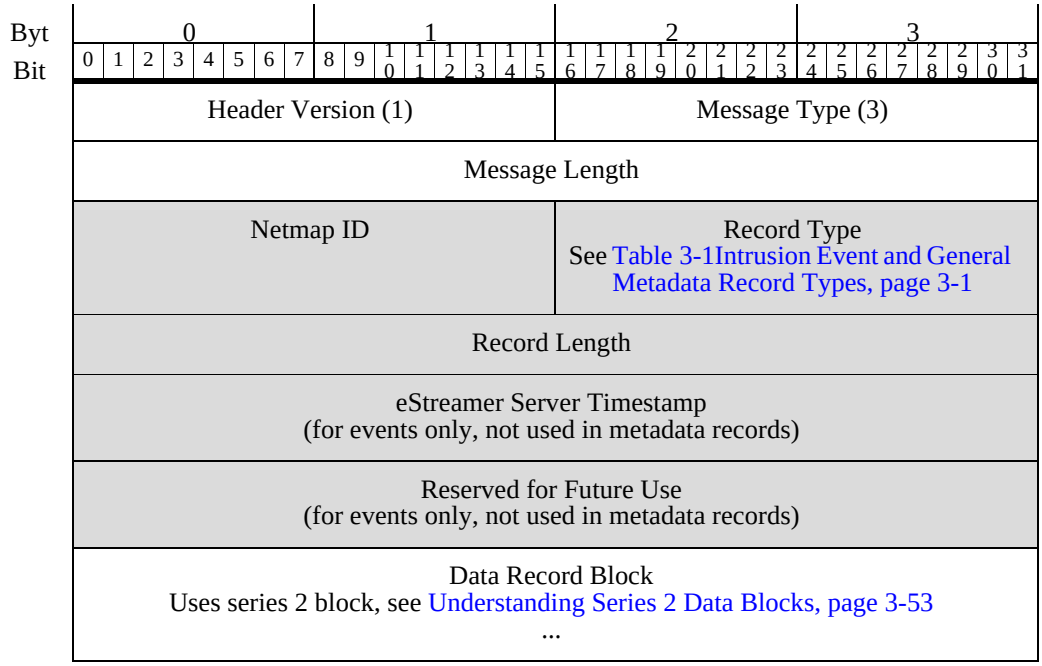
The graphic below shows the structure of event extra data messages. The Intrusion Event Extra Data message is an example of this message group.



Event extra data messages have the same format as correlation event messages, with a data block directly after the record header. Unlike correlation messages, they use series 2 data blocks, not series 1 data blocks, which have a separate numbering sequence. For information about series 2 block types, see Understanding Series 2 Data Blocks, page 3-53.

Event Extra Data Message Record Header

The shaded section of the following graphic shows the fields of the record header in event extra data messages. The table that follows defines the record header fields for event extra data messages.



The following table describes each field in the record header of event extra data messages.

Table 2-11 Event Extra Data Message Record Header Fields

Field	Data Type	Description
Netmap ID	uint16	The first bit of this field is a flag indicating whether the header is an extended header containing an archive timestamp. The remaining 15 bits are an optional field containing the Netmap ID for the domain on which the event was detected. If this field is not used, it is left empty. Netmap IDs map to domains as provided in metadata.
Record Type	uint16	Identifies the data record content type. See Table 3-1Intrusion Event and General Metadata Record Types, page 3-1 for the list of event extra data record types.
Record Length	uint32	Length of the content of the message after the record header. Does not include the 8 or 16 bytes of the record header. (Record Length plus the length of the record header equals Message Length.)
eStreamer Server Timestamp	uint32	Indicates the timestamp applied when the event was archived by the eStreamer server. Also called the archival timestamp. Field present only if bit 23 is set in the request message flags. Field is not present for events generated by the Management Center.
Reserved for future use	uint32	Reserved for future use. Field present only if bit 23 is set in the request message flags. Field is not present for events generated by the Management Center.

Data Block Header

Series 1 blocks and series 2 blocks have similar structures but distinct numbering. These blocks can appear anywhere in the data portion of a discovery, correlation, connection, or event extra data message. These blocks encapsulate other blocks at multiple levels of nesting.

The data blocks in both the first and second series begin with the header structure shown in the graphic below. The following table provides information about the header fields. The header is followed immediately by the data structure associated with the data block type.

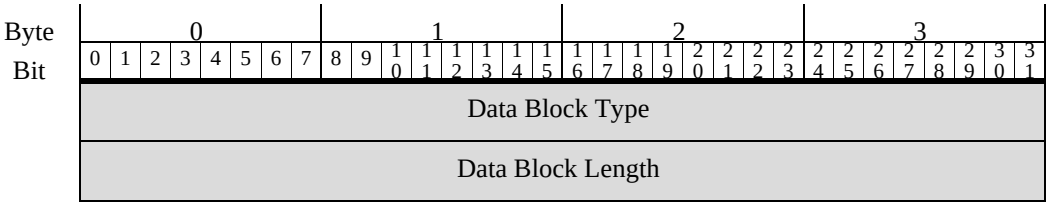


Table 2-12

Field	Data Type	Description
Data Block Type	uint32	For series 1 block types, see Understanding Discovery (Series 1) Blocks, page 4-62. For series 2 block types, see Table 3-24Series 2 Block Types, page 3-53.
Data Block Length	uint32	Length of the data block. Includes the number of bytes of data plus the 8 bytes in the two data block header fields.

Host Request Message Format

To receive host profiles, you submit Host Request messages. You can request data for a single host or multiple hosts defined by an IP address range.

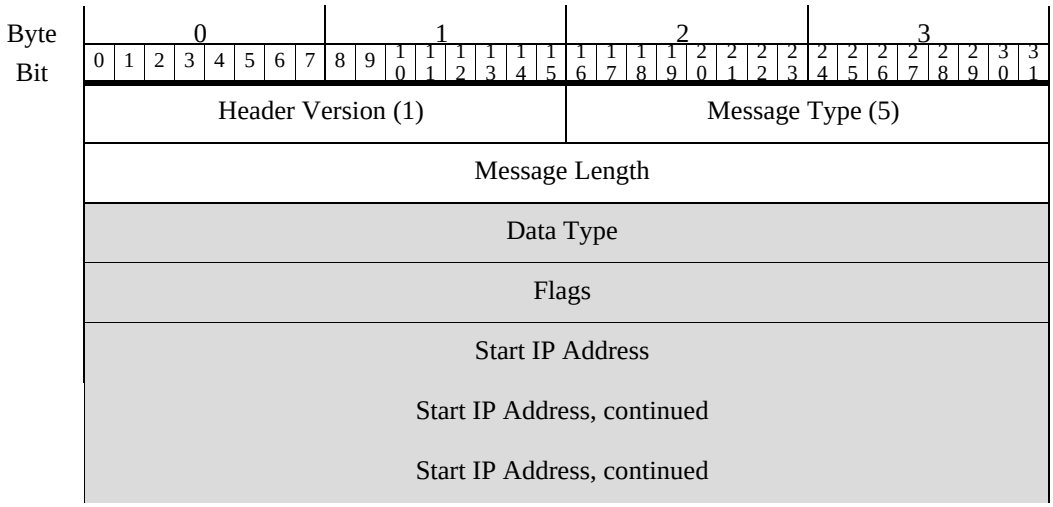
Note that it is mandatory for all data requests, including requests for host profile information, to first initialize the session by submitting an Event Stream Request message. To set up for streaming host data only, you can use any of the following request flag settings in your initial Event Stream Request message:

- set the bit for the appropriate version of metadata (this can be beneficial when streaming host data)
- set no request flags
- set bit 11 (to suppress any default event streaming if using legacy versions of eStreamer)

After the initial message, you then use a Host Request message (type 5) to specify the hosts.

Note For legacy eStreamer versions with default event streaming, if you want to stream only host profile data, you need to suppress the default event messages. First send the server an Event Stream Request message with bit 11 in the Request Flags field set to `1`; then, send the Host Request message.

The graphic below shows the format for the Host Request message. The shaded fields are specific to the Host Request message format and are defined in the following table. The preceding three fields are the standard message header.



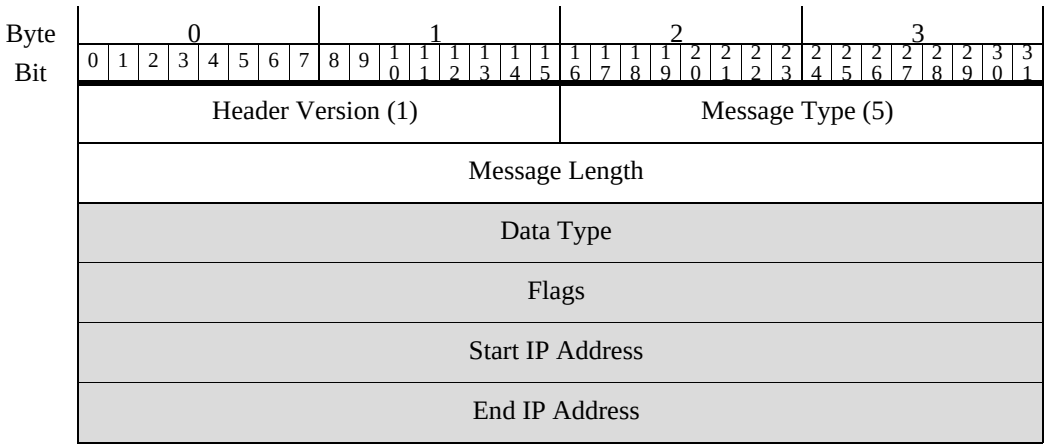
Start IP Address, continued
End IP Address
End IP Address, continued
End IP Address, continued
End IP Address, continued

The following table explains the message fields.

Table 2-13 Host Request Message Fields

Field	Data Type	Description
Data Type	uint32	Requests data for a single host or multiple hosts, using the following codes: • `0` — Version 3.5 - 4.6 for a single host. • `1` — Version 3.5 - 4.6 for multiple hosts (uses block 34). • `2` — Version 4.7 - 4.8 for a single host (uses block 47). • `3` — Version 4.7 - 4.8 for multiple hosts (uses block 47). • `4` — Version 4.9 - 4.10 for a single host (uses block 92). • `5` — Version 4.9 - 4.10 for multiple hosts (uses block 92). • `6` — Version 5.0.x data for a single host (uses block 111, see Full Host Profile Data Block 5.0 - 5.0.2, page B-344). • `7` — Version 5.0.x data for multiple hosts (uses block 111, see Full Host Profile Data Block 5.0 - 5.0.2, page B-344). • `8` — Version 5.1.x data for multiple hosts (uses block 111, see Full Host Profile Data Block 5.1.1, page B-353). • `9` — Version 5.1.x data for multiple hosts (uses block 111, see Full Host Profile Data Block 5.1.1, page B-353). • `10` — Rule documentation data (uses block 27, see Rule Documentation Message Format, page 2-31) • `11` — Version 5.2x data for multiple hosts (uses block 111, see Full Host Profile Data Block 5.2.x, page B-361). • `12` — Version 5.2.x data for multiple hosts (uses block 111, see Full Host Profile Data Block 5.2.x, page B-361). • `13` — Version 5.3+ data for multiple hosts (uses block 111, see Full Host Profile Data Block 5.3+, page 5-1). • `14` — Version 5.3+ data for multiple hosts (uses block 111, see Full Host Profile Data Block 5.3+, page 5-1).
Flags	32-bit field	• `0x00000001` — Causes the Notes field of the host profile to be populated (with user-defined information about the host stored in the Secure Firewall System). • `0x00000002` — Causes the Banner field of the service block to be populated (with the first 256 bytes of the first packet detected for the service). Banners are disabled by default and available only if configured.
Start IP Address	uint8[16]	IP address of the host whose data should be returned (if request is for a single host), or the starting address in an IP address range (if request is for multiple hosts). Can be either an IPv4 or IPv6 address.
End IP Address	uint8[16]	Ending address in an IP address range (if request is for multiple hosts), or the Start IP Address value (if request is for single host). Can be either an IPv4 or IPv6 address.

The graphic below shows the format for the legacy Host Request message. eStreamer will still respond to this request. The only difference from the current request is the smaller IPv4 address fields. The shaded fields are specific to the Host Request message format and are defined in the following table. The preceding three fields are the standard message header.



The following table explains the message fields.

Table 2-14 Host Request Message Fields

Field	Data Type	Description
Data Type	uint32	Requests data for a single host or multiple hosts, using the following codes: • `0` — Version 3.5 - 4.6 for a single host. • `1` — Version 3.5 - 4.6 for multiple hosts (uses block 34). • `2` — Version 4.7 - 4.8 for a single host (uses block 47). • `3` — Version 4.7 - 4.8 for multiple hosts (uses block 47). • `4` — Version 4.9 - 4.10 for a single host (uses block 92). • `5` — Version 4.9 - 4.10 for multiple hosts (uses block 92). • `6` — Version 5.0+ data for a single host (uses block 111, see Full Host Profile Data Block 5.3+, page 5-1). • `7` — Version 5.0+ data for multiple hosts (uses block 111, see Full Host Profile Data Block 5.3+, page 5-1).
Flags	32-bit field	• `0x00000001` — Causes the Notes field of the host profile to be populated (with user-defined information about the host stored in the Secure Firewall System). • `0x00000002` — Causes the Banner field of the service block to be populated (with the first 256 bytes of the first packet detected for the service). Banners are disabled by default and available only if configured.

Table 2-14 Host Request Message Fields (continued)

Field	Data Type	Description
Start IP Address	uint8[4]	IP address of the host whose data should be returned (if request is for a single host), or the starting address in an IP address range (if request is for multiple hosts). Specify the address in IP address octets.
End IP Address	uint8[4]	Ending address in an IP address range (if request is for multiple hosts), or the Start IP Address value (if request is for single host).

Rule Documentation Message Format

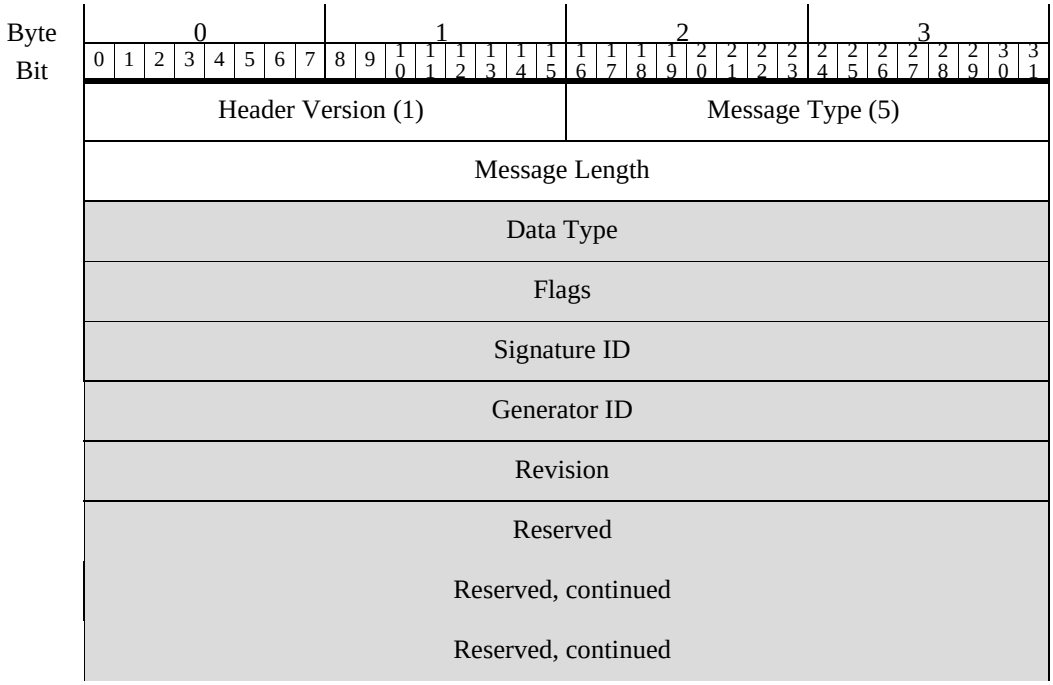
To receive rule documentation profiles, you submit Rule Documentation messages. You request these by generator ID, signature ID, and revision.

Note that it is mandatory for all data requests, including requests for rule documentation information, to first initialize the session by submitting an Event Stream Request message. To set up for streaming host data only, you can use any of the following request flag settings in your initial Event Stream Request message:

- set the bit for the appropriate version of metadata (this can be beneficial when streaming host data)
- set no request flags
- set bit 11 (to suppress any default event streaming if using legacy versions of eStreamer)

After the initial message, you then use a Rule Documentation message (type 10) to specify the rule.

The graphic below shows the format for the Rule Documentation message. The shaded fields are specific to the Rule Documentation message format and are defined in the following table. The preceding three fields are the standard message header.



Reserved, continued
Reserved, continued

The following table explains the message fields.

Table 2-15 Rule Documentation Message Fields

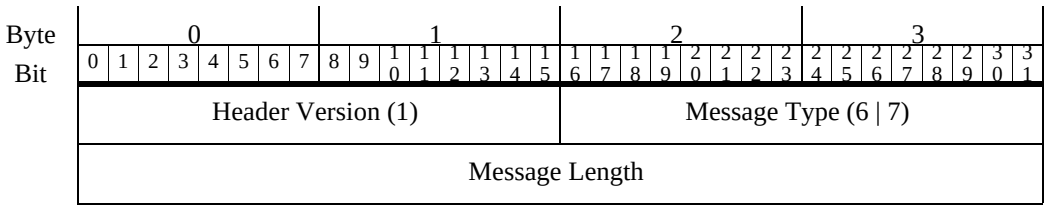
Field	Data Type	Description
Data Type	uint32	Requests data for a Rule Documentation Data Block. This value is always `10`. See Rule Documentation Data Block for 5.2+, page 3-103.
Flags	32-bit field	• `0x00000001` — Causes the Notes field of the Rule Documentation data block to be populated (with user-defined information about the host stored in the Secure Firewall System). • `0x00000002` — Causes the Banner field of the service block to be populated (with the first 256 bytes of the first packet detected for the service). Banners are disabled by default and available only if configured.
Signature ID	uint32	Identification number of the requested rule.
Generator ID	uint32	Identification number of the Secure Firewall System preprocessor for the requested rule.
Rule Revision	uint32	Rule revision number.
Reserved	uint8[20]	This field is not currently used.

Host Data and Multiple Host Data Message Format

eStreamer responds to host requests by sending host data messages, each with a full host profile data block. eStreamer sends one host data message for each host specified in the request. eStreamer uses the type 6 message to respond to requests for a single host profile, and uses the type 7 message to respond to requests for multiple hosts. The formats of the type 6 and type 7 messages are identical, only the message type is different.

Host data messages do not have a record type field. The structure of the message is communicated by the message type and the data block type of the full host profile included in the message. Full host profile data blocks are in the series a group of blocks.

The graphic below shows the format of the host data message and the table that follows defines the shaded fields:



Full Host Profile Data Block Type See Table 4-30Host Discovery and Connection Data Block Types, page 4-63
Length
Full Host Profile Data Block

The fields specific to the Host Request message are:

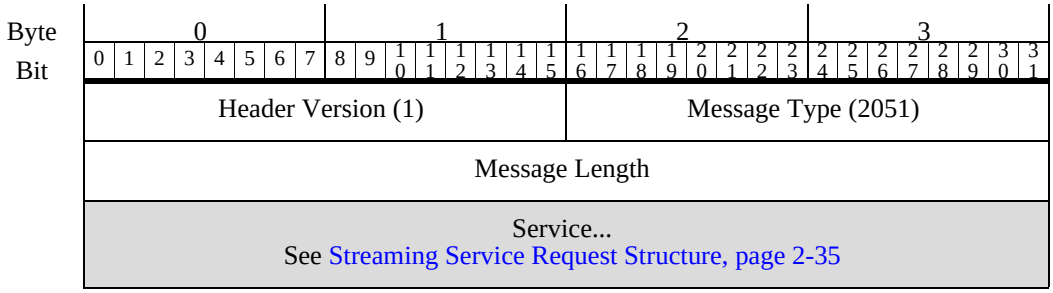
Table 2-16

Field	Data Type	Description
Full Host Profile Data Block Type	uint32	Specifies the block type for the full host profile data included in the message. See Table 4-30Host Discovery and Connection Data Block Types, page 4-63.
Length	uint32	Length of the full host profile data in the message.
Full Host Profile Data Block	variable	The host data. For links to the definitions of current full host profile data blocks, see Table 4-30Host Discovery and Connection Data Block Types, page 4-63.

Streaming Information Message Format

When the eStreamer service receives a request for an extended request, it sends the client the Streaming Information message described below. This message advertises the server's list of available services. Currently, the only relevant option is the eStreamer service (6667), although the message can list other services, which should be ignored. Each advertised service is represented by a Streaming Service Request structure described in Streaming Service Request Structure, page 2-35.

The graphic below illustrates the format for the Streaming Information message. The shaded field is specific to this message type. The preceding three fields are the standard message header.



The fields of the Streaming Information message are:

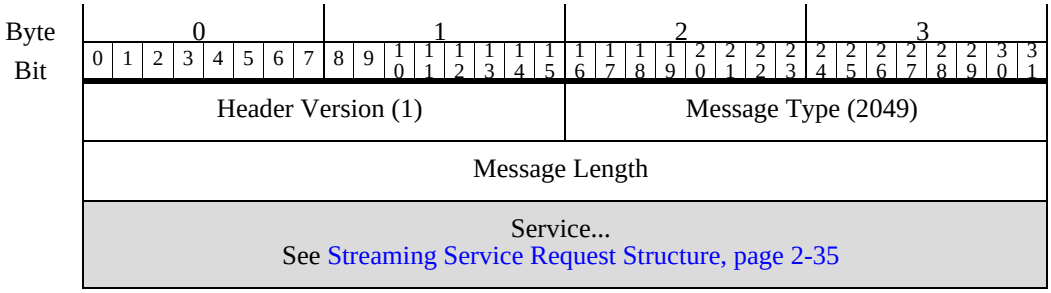
Table 2-17 Streaming Information Message Fields

Field	Data Type	Description
Header Version	uint16	Set to `1`.
Message Type	uint16	eStreamer message type. Set to `2051` for Streaming Request messages.
Message Length	uint32	Length of the content of the message after the message header. Does not include the bytes in the Header Version, Message Type, and Message Length fields.
Service[]	array	List of available services. See Streaming Service Request Structure, page 2-35.

Streaming Request Message Format

The client uses the Streaming Request message to specify to eStreamer the service in the Streaming Information message that it wants to use, followed by a set of requests for event types and versions to be streamed. The graphic below shows the message structure and the following table defines the fields. The requested service is represented by a Streaming Service Request structure described in Streaming Service Request Structure, page 2-35.

The graphic below illustrates the format for the Streaming Information message. The shaded field is specific to this message type. The preceding three fields are the standard message header.



The fields of the Streaming Request message are:

Table 2-18 Streaming Request Message Fields

Field	Data Type	Description
Header Version	uint16	Set to `1`.
Message Type	uint16	eStreamer message type. Set to `2049` for Streaming Request messages.
Message Length	uint32	Length of the content of the message after the message header. Does not include the bytes in the Header Version, Message Type, and Message Length fields.
Service[]	array	List of requested service structures. See Streaming Service Request Structure, page 2-35.

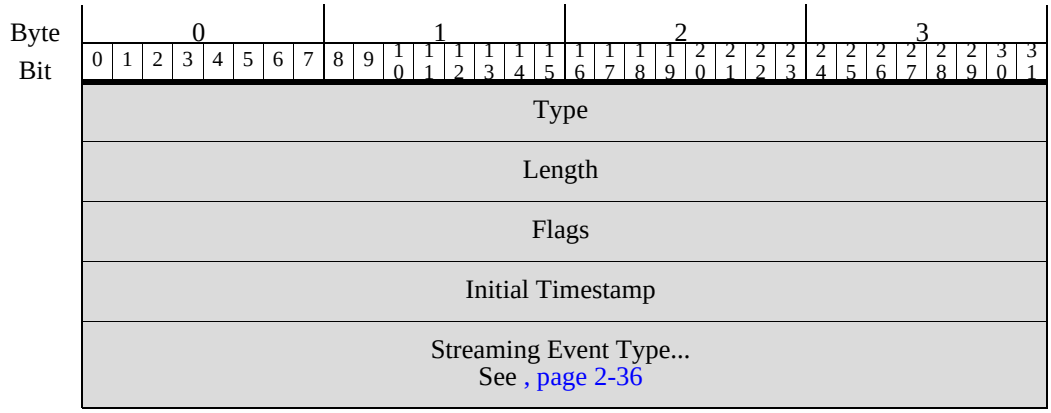
Streaming Service Request Structure

The eStreamer service sends one Streaming Service Request data structure in the Streaming Information message for each service it advertises. The eStreamer service does not use the last field of the Streaming Service Request, which provides for a list of event types to be included.

The client processes the Streaming Service Request structure from eStreamer and uses the same structure in the response it returns to the server. In the Streaming Service Request that the client sends to the server, it includes, first, a request for the service advertised by eStreamer, and, second, a list of Streaming Event Type structures, which specify the requested event types the client wants to receive.

Each Streaming Event Type structure contains two fields to specify the event type and version for each requested event type. For information on the Streaming Event Type structure, see , page 2-36.

The graphic below shows the fields of the Streaming Service Request structure. The table that follows defines the fields.



The fields of the Streaming Service Request structure are:

Table 2-19 Streaming Service Request Fields

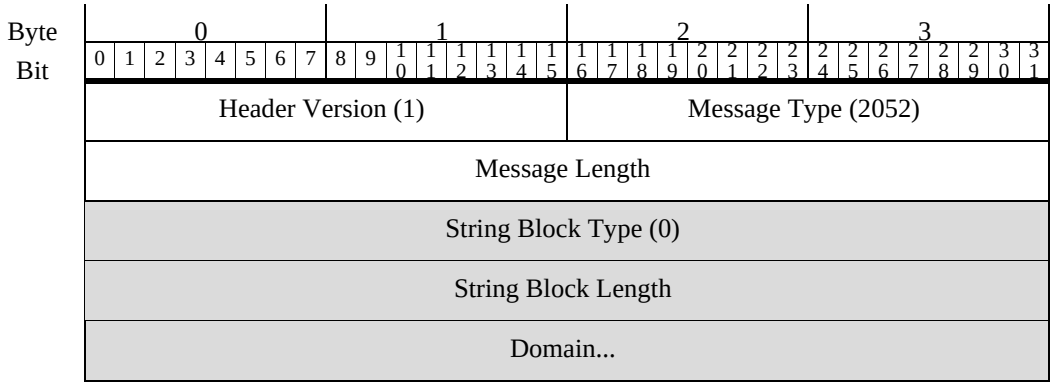
Field	Data Type	Description
Type	uint32	Service ID. In eStreamer server messages, this advertises an available service. In client messages, it specifies a requested service. Current valid options: • `6667` (for eStreamer service)
Length	uint32	Service request length. Describes the length of the service request, including Type and Length. Note that Length must include all the Streaming Event Type records in the message, plus the terminating one.
Flags	uint32	In eStreamer's Streaming Information messages: Always `0`. In client's Streaming Request message: replicates the flag settings in the original Event Stream Request message.

Table 2-19 Streaming Service Request Fields (continued)

Field	Data Type	Description
Initial Timestamp	uint32	In eStreamer's Streaming Information messages: Always `0`. In client's Streaming Request message: replicates the timestamp in the original Event Stream Request message.
Streaming Event Type	array	In eStreamer's Streaming Information message: • Reserved for future use. Has `0` length. In client's Streaming Request message: • One Streaming Event Type entry for each requested event type. See , page 2-36. • Terminate the request list with a `0` Event Type entry, with both Event Type and Version set to `0`. See , page 2-36.

Domain Streaming Request Message Format

The client uses the Domain Streaming Request message to request events from a specific domain from eStreamer. The graphic below shows the message structure and the following table defines the fields. The shaded fields are specific to this message type. The preceding three fields are the standard message header.



The fields of the Domain Streaming Request message are:

Table 2-20 Domain Streaming Request Message Fields

Field	Data Type	Description
Header Version	uint16	Set to `1`.
Message Type	uint16	eStreamer message type. Set to `2052` for Domain Streaming Request messages.
Message Length	uint32	Length of the content of the message after the message header. Does not include the bytes in the Header Version, Message Type, and Message Length fields.
String Block Type	uint32	Initiates a String data block containing the domain. This value is always `0`.
String Block Length	uint32	The number of bytes included in the domain String data block, including eight bytes for the block type and header fields plus the number of bytes in the domain.
Domain	string	Domain from which streaming events are requested. If left blank, the service will stream events for all domains to which the client has access.

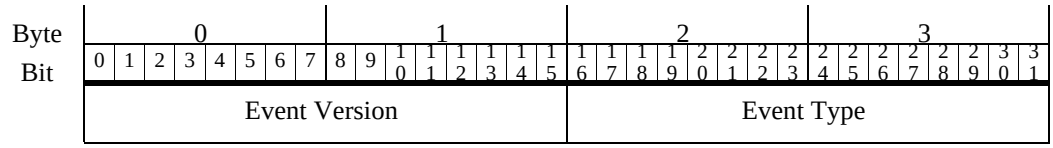
Streaming Event Type Structure

eStreamer clients use the Streaming Event Type structure to specify an event's version and version. Each event version/type combination is a request for an event stream.

Lists of Streaming Event Type structures must be terminated with a structure with all fields set to zero. That is:

```
Event Version = 0
Event Type = 0
```

The following diagram illustrates the format for the Streaming Event Type structure.



The fields of the Streaming Event Type structure are:

Table 2-21 Streaming Event Type Fields

Field	Data Type	Description
Event Version	uint16	Version number of event type. For list of versions supported for each event type, see Table 2-22Event Types and Versions for Extended Request, page 2-38.
Event Type	uint16	Code for requested event type. For the current list of valid event types and version codes, see Table 2-22Event Types and Versions for Extended Request, page 2-38. List of event types should be terminated with a zero event type and zero event version.

The following table lists the event types and versions that clients can specify in extended requests. The table indicates the Management Center software versions that correspond to each event type version. For example, to request the correlation events that were supported by the Management Center in version 4.8.0.2 - 4.9.1, you should request Event Type 31, Version 5. If an event was recorded with a different event type, it will be upgraded or downgraded to match the format of the requested event type.

Table 2-22 Event Types and Versions for Extended Request

To request...	Use this event version number...	And this event code
intrusion events	1 — 4.8.x and earlier 2 — 4.9 - 4.10.x 3 — 5.0 - 5.1 4 — 5.1.1.x 5 — 5.2.x 6 — 5.3 7 — 5.3.1 8 — 5.4.x 9— 6.x 10— 7.0+	12
metadata	1 — 3.2 - 4.5.x 2 — 4.6.0.x 3 — 4.6.1 - 4.6.x 4 — 4.7+	21
correlation and compliance allow list events	1 — 3.2 and earlier 2 — 4.0 - 4.4.x 3 — 4.5 - 4.6.1 4 — 4.7 - 4.8.0.1 5 — 4.8.0.2 - 4.9.1.x 6 — 4.10.0 - 4.10.x 7 — 5.0 - 5.0.2 8 — 5.1 - 5.3.x 9 — 5.4+	31

Table 2-22 Event Types and Versions for Extended Request (continued)

To request...	Use this event version number...	And this event code
discovery events	1 — 3.2 and earlier 2 — 3.0 - 3.4.x 3 — 3.5 - 4.6.x 4 — 4.7 - 4.8.x 5 — 4.9.0.x 6 — 4.9.1 - 4.9.x.x 7 — 4.10.0 - 4.10.x 8 — 5.0.x 9 — 5.1.x 10 — 5.2 - 5.3 11 — 5.3.1+	61
connection events	1 — 4.0 - 4.1 3 — 4.5 - 4.6.1 4 — 4.7 - 4.9.0.x 5 — 4.9.1 - 4.10.x 6 — 5.0.x 7 — 5.1.0.x 8 — 5.1.1.x 9 — 5.2.x 10 — 5.3 11 — 5.3.1 12 — 5.4 13 — 5.4.0.1-5.4.0.2 14 — 6.0.x 15 — 6.1.x 16 — 7.0.x 17 — 7.1+	71
user events	1 — 4.7 - 4.10.x 2 — 5.0.x 3 — 5.1-5.1.x 4 — 5.2 5— 6.0 6— 6.1 7— 6.2+	91
malware events	1 — 5.1.0.x 2 — 5.1.1.x 3 — 5.2.x 4 — 5.3 5 — 5.3.1 6 — 5.4.x 7— 6.x 8— 7.0+	101

Table 2-22 Event Types and Versions for Extended Request (continued)

To request...	Use this event version number...	And this event code
file events	1 — 5.1.1 - 5.1.x 2 — 5.2.x 3 — 5.3 4 — 5.3.1 5 — 5.4.x 6— 6.x 7— 7.0+	111
impact correlation events	1 — 5.2.x and earlier 2 — 5.3+	131
terminating event type in a list	0	0

Sample Extended Request Messages

Streaming Information Message

In the sample below, the server advertises two services, the first type 6667 (eStreamer) and the second type 5000. In Streaming Information messages from the server, the flags field and initial timestamp fields are zero, and the message specifies no event types.

Table 2-23

Header Version:	1	/*always 1*/
Message Type:	2051	/*streaming info msg*/
Message Length	32	/*bytes of msg content*/
Service[1].Type	6667	/*eStreamer service ID*/
Service[1].Length	8	
Service[1].Flags	0	/*no flags from server*/
Service[1].Initial Timestamp	0	/*always 0*/
Service[2].Type	5000	/*service-2 ID*/
Service[2].Length	8	
Service[2].Flags	0	/*no flags from server*/
Service[2].Initial Timestamp	0	/*always 0*/
Header Version:	1	/*always 1*/
Message Type:	2051	/*streaming info msg*/

Streaming Request Message

Below is a Streaming Request message where the client requests service type 6667 (eStreamer) and specifies two event types: version 6 of connection events (event type 71) and version 4 of metadata (event type 21).

Table 2-24

Header Version:	1	/*always 1*/
Message Type:	2049	/*stream request msg*/
Message Length	28	/*payload bytes*/
Service[1].Type	6667	/*eStreamer service ID*/
Service[1].Length	20	
Service[1].Flags	30	/*original flags value*/
Service[1].Initial Timestamp	0	/*original timestamp*/
Service[1].Event[1].Version	6	/*version 6*/
Service[1].Event[1].Type	71	/*connection events*/
Service[1].Event[2].Version	4	/* version 4*/
Service[1].Event[2].Type	21	/*metadata*/
Service[1].Event[3].Version	0	/*terminate event list*/
Service[1].Event[3].Type	0	/*terminate event list*/

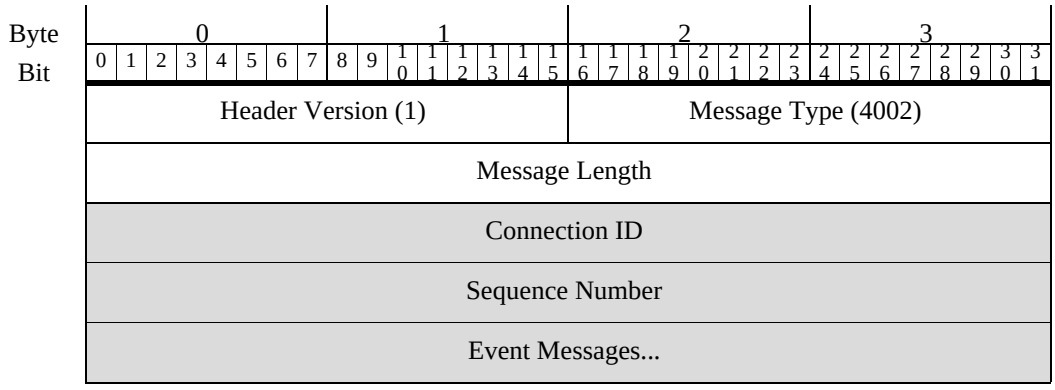
Message Bundle Format

The eStreamer server sends messages in a bundle format when the client submits an extended request.

The client responds with a null message to acknowledge receipt of an entire bundle. The client should not acknowledge receipt of individual messages in a bundle.

Message bundles have a message type of 4002.

The graphic below shows the structure of a message bundle. The shaded fields are specific to the bundle message type. The following table describes the content of the fields and data structures.



The fields of a message bundle message are:

Table 2-25 Message Bundle Message Fields

Field	Data Type	Description
Header Version	uint16	Always `1`.
Message Type	uint16	Always `4002`.
Message Length	uint32	Length of the content of the message after the message header. Does not include the bytes in the bundle's Header Version, Message Type, and Message Length fields. As the client loads a message from the bundle, it can subtract the message's total length (including header) from the length in this field. As long as the remainder is positive, there are more messages to process.
Connection ID	uint32	A unique identifier for the connection with the server.
Sequence Number	uint32	Starts at `1` and increments by one for each bundle sent by the eStreamer server.
Event Messages []	array	The events streamed by the server in the bundle. Each message has a full set of headers, including message version number (1), archive timestamp if requested, and so forth.

Understanding Metadata

The eStreamer server can provide metadata along with requested event records. To receive metadata, you must explicitly request it. See Table 2-6Request Flags, page 2-14 for information on how to request a given version of metadata. The metadata provides context information for codes and numeric identifiers in the event records. For example, an intrusion event contains only the internal identifier of the detecting device, and the metadata provides the device's name.

Depending on the metadata requested and the environment, the amount of metadata sent may vary considerably.

Metadata Transmission

If the request message specifies metadata, eStreamer sends the relevant metadata record before it sends any related event records.

eStreamer keeps track of the metadata it has sent to the client and does not resend the same metadata record. The client should cache each received metadata record. If the client application uses a limited cache size, when the cache becomes full the client should flush the cache and reconnect to the eStreamer service in order to ensure that the client receives all metadata values for the events that are being streamed. eStreamer does not keep a history of metadata transmissions from one session to the next, so when a new session starts and a request message specifies metadata, eStreamer restarts metadata streaming from scratch. When reconnecting, the client can specify the "Initial Timestamp" in the Request Message in order to avoid duplicate or missing events.