ExtraHop Trigger API

Version 4.1 HF4 (version 4.1.4.24670)

Publication Date: 1/28/2016

TOC

Global Functions

debug(message:String): void
Writes to the runtime log if debugging is enabled.

getTimestamp(): Number (version 4.0.21305+)
Returns the timestamp from the packet that caused the trigger event to fire, expressed in milliseconds with microseconds as the fractional part after the decimal.

log(message: String): void
Writes to the runtime log regardless of whether debugging is enabled or not.

> **Note**: The limit for runtime log entries is 2048 bytes. To log larger entries, use `rsyslog`.

md5(message: String): String
Returns the MD5 sum of a string.

uuid(): String
Returns a random version 4 Universally Unique Identifier (UUID).

Deprecated

exit(): Void
Deprecated. Use the `return` statement instead.

getTimestampMSec(): Number
Deprecated. User `getTimestamp` instead.

Multiple calls to `debug` and `log` statements in which the message is the same value will display once every 30 seconds.

You can use local JavaScript functions when you write triggers. Elements in ECMAScript 5 are supported. Examples are:

```
function foo(a, b, c) { ... }
```

```
var foo = function(a, b, c) { ... }
```

For triggers that contain multiple events (version 3.9.16835+), use the `event` property to see the event on which the trigger is currently executing. For example:

```
if (event === "HTTP_REQUEST") {
    /* code */
}
else if (event === "HTTP_RESPONSE") {
    /* other code */
}
```

ExtraHop Datatypes

ExtraHop datatypes record custom metrics using the **Network**, **Application**, and **Device** classes.

The ExtraHop system records data using two key metric categories:

- **Top-level metrics**. Time series of simple datatypes
  - **count**: Number (e.g., HTTP requests).
  - **snapshot**: A special type of count metric that, when queried over time, returns the most recent value (e.g., TCP established connections).
  - **dataset**: Statistical summary of timing information (5-number summary: min, 25th-percentile, median, 75th-percentile, max).
  - **sampleset**: Statistical summary of timing information (mean and standard deviation).
  - **max**: A special type of count metric that preserves the maximum.
- **Detail metrics**. Time series of datatypes consisting of key-value pairs, where the key is a string or an IP address and the value is a top-level data type. Detail metrics provide drill-down information for top level metrics.

Examples:

- To record information about the number of HTTP requests over time, use a top-level count metric.
- To record information about HTTP processing time over time, use a top-level sampleset (mean and average) or dataset (5-number summary) metric.
- To record information about the number of times each client IP address accessed the server, use a detail count metric with the IPAddress key and an integer representing the number of accesses as a value.
- To record information about the length of time it took the server to process each URI, use a detail sampleset or dataset metric with the URI string key and an integer representing processing time as a value.
- To record the slowest HTTP statements over time without relying on a **Session** table, use a top-level and a detail max metric.

Events List

This guide is an API reference for ExtraHop® users writing application inspection triggers™. For more information about creating and using triggers in conjunction with custom pages and applications, refer to Getting Started with Application Inspection Triggers on the ExtraHop Support Forum.

Note: Where applicable in this guide, a firmware revision number is included in parentheses to indicate the firmware revision in which that functionality was introduced.

Events	Description
AAA_RESPONSE: Event	Fires on every AAA request processed by the device.
AAA_REQUEST: Event	Fires on every AAA response processed by the device
ACTIVEMQ_MESSAGE: Event	Fires on every Java Message Service (JMS) message processed by the device.
CIFS_REQUEST: Event	Fires on every CIFS request processed by the device.
CIFS_RESPONSE: Event	Fires on every CIFS response processed by the device.
DB_REQUEST: Event	Fires on every database request processed by the device.
DB_RESPONSE: Event	Fires on every database response processed by the device.
DNS_REQUEST: Event	Fires on every DNS request processed by the device.
DNS_RESPONSE: Event	Fires on every DNS response processed by the device.
FIX_REQUEST: Event	Fires on every FIX request processed by the device (version 4.0.18766+).
FIX_RESPONSE: Event	Fires on every FIX response processed by the device (version 4.0.18766+).
FLOW_CLASSIFY: Event	Fires once per flow when the L7 protocol of the flow has been determined (version 3.8.15785+).
FLOW_TICK: Event	Once FLOW_CLASSIFY has fired, FLOW_TICK fires periodically. FLOW_TICK will fire, at a minimum, on every subsequent turn (version 3.9.17024+).
FLOW_TURN: Event	Fires on every TCP or UDP turn (version 3.10.17624+).
FTP_REQUEST: Event	Fires on every FTP request processed by the device (version 4.0.18766+).
FTP_RESPONSE: Event	Fires on every FTP response processed by the device (version 4.0.18766+).
HL7_REQUEST: Event	Fires on every HL7 request processed by the device (version 4.0.18766+).
HL7_RESPONSE: Event	Fires on every HL7 response processed by the device (version 4.0.18766+).
HTTP_REQUEST: Event	Fires on every HTTP request processed by the device.
HTTP_RESPONSE: Event	Fires on every HTTP response processed by the device.
IBMMQ_REQUEST: Event	Fires on every IBMMQ request processed by the device.

Events	Description
IBMMQ_RESPONSE: Event	Fires on every IBMMQ response processed by the device.
ICA_AUTH: Event	Fires when the ICA authentication is complete.
ICA_CLOSE: Event	Fires when the ICA session is torn down.
ICA_OPEN: Event	Fires right after the ICA application first loads.
ICA_TICK: Event	Fires periodically while the user interacts with the ICA application.
ICMP_MESSAGE: Event	Fires on every ICMP message processed by the device (version 4.1.22706+).
LDAP_REQUEST: Event	Fires on every LDAP request processed by the device.
LDAP_RESPONSE: Event	Fires on every LDAP response processed by the device.
LLDP_FRAME: Event	Fires on every LLDP frame processed by the device (version 4.0.19680+).
MEMCACHE_REQUEST: Event	Fires on every Memcache request processed by the device.
MEMCACHE_RESPONSE: Event	Fires on every Memcache response processed by the device.
METRIC_CYCLE_BEGIN: Event	Fires when a metric interval begins (version 4.0.21607+).
METRIC_CYCLE_END: Event	Fires when a metric interval ends (version 4.0.21607+).
MONGODB_REQUEST: Event	Fires on every MongoDB request processed by the device (version 4.0.13997+).
MONGODB_RESPONSE: Event	Fires on every MongoDB response processed by the device (version 4.0.13997+).
NEW_APPLICATION: Event	Fires when an application is first discovered (version 4.0.21607+).
NEW_DEVICE: Event	Fires when a device is first discovered (version 4.0.21607+).
NEW_VLAN: Event	Fires when a VLAN is first discovered (version 4.0.21607+).
NFS_REQUEST: Event	Fires on every NFS request processed by the device.
NFS_RESPONSE: Event	Fires on every NFS response processed by the device.
RTCP_MESSAGE: Event	Fires on every RTCP message processed by the device (version 4.1.22706+).
RTP_CLOSE: Event	Fires when an RTP connection is closed (version 4.1.22706+).
RTP_OPEN: Event	Fires when an RTP connection is opened (version 4.1.22706+).
RTP_TICK: Event	Fires periodically on RTP flows (version 4.1.22706+).
SESSION_EXPIRE: Event	Fires periodically (in approximately 30 second increments) as long as the session table is in use.
SIP_REQUEST: Event	Fires on every SIP request processed by the device (version 4.1.22706+).
SIP_RESPONSE: Event	Fires on every SIP response processed by the device (version 4.1.22706+).
SMPP_REQUEST: Event	Fires on every SMPP request processed by the device.

Events	Description
SMPP_RESPONSE: Event	Fires on every SMPP response processed by the device.
SMTP_REQUEST: Event	Fires on every SMTP request processed by the device (version 4.0.18766+).
SMTP_RESPONSE: Event	Fires on every SMTP response processed by the device (version 4.0.18766+).
SSL_ALERT: Event	Fires when an SSL alert record is exchanged.
SSL_CLOSE: Event	Fires when the SSL connection is closed.
SSL_HEARTBEAT: Event (version 3.10.19872+)	Fires when an SSL heartbeat record is exchanged (version 3.10.19851+).
SSL_OPEN: Event	Fires when the SSL connection is first established.
SSL_PAYLOAD: Event (version 4.1.24320+)	Fires when the decrypted SSL payload matches the criteria configured in the associated trigger. (version 4.1.24320+)
SSL_RECORD: Event (version 3.8.15910+)	Fires when an SSL record is exchanged (version 3.8.15910+).
SSL_RENEGOTIATE: Event (version 4.0.20129+)	Fires on SSL renegotiation (version 4.0.21431+).
TCP_CLOSE: Event	Fires when the TCP connection is shut down.
TCP_OPEN: Event	Fires when the TCP connection is first established.
TCP_PAYLOAD: Event	Fires when the payload matches the criteria configured in the associated trigger (version 3.10.18611+).
TELNET_MESSAGE: Event	Fires on every Telnet command or line of data from the client or server (version 4.1.22706+).
UDP_PAYLOAD: Event	Fires when the payload matches the criteria configured in the associated trigger (version 3.10.18620+).

For triggers that contain multiple events (version 3.9.16835+), use the event property to see the event on which the trigger is currently executing. For example:

```
if (event === "HTTP_REQUEST") {
    /* code */
}
else if (event === "HTTP_RESPONSE") {
    /* other code */
}
```

Classes List

Name	Description
AAA	The AAA class allows retrieval of metrics available during the AAA_REQUEST and AAA_RESPONSE events.
ActiveMQ	The ActiveMQ class allows retrieval of metrics available during the ACTIVEMQ_MESSAGE event.
Application	The Application class allows the creation of new applications and adding custom metrics at the application level. Refer to the table in the Application section for a list of events.
Buffer	A buffer is an object with the characteristics of an array.
CIFS	The CIFS class allows retrieval of metrics available during the CIFS_REQUEST and CIFS_RESPONSE.
Dataset	The Dataset class is used for representing dataset metrics. It provides access to the raw dataset values and provides an interface for computing percentiles.
DB	The DB class allows retrieval of metrics available during the DB_REQUEST and DB_RESPONSE events.
Device	The Device class allows retrieval of device attributes and adding custom metrics at the device level.
Discover	The Discover class provides access to newly discovered VLANs, devices or applications on the NEW_VLAN, NEW_DEVICE, and NEW_APPLICATION events.
DNS	The DNS class allows retrieval of metrics available during the DNS_REQUEST and DNS_RESPONSE events.
FIX	The FIX class allows retrieval of metrics available during the FIX_REQUEST and FIX_RESPONSE events.
FLOW	Flowo refers to a TCP or UDP connection between two endpoints. The Flow class provides access to elements of these conversations, such as endpoint identities and flow age. It also contains a flow store designed to pass objects from request to response on the same flow.
FTP	The FTP class allows retrieval of metrics available during the FTP_REQUEST and FTP_RESPONSE events.
HL7	The HL7 class allows retrieval of metrics available during the HL7_REQUEST and HL7_RESPONSE events.
HTTP	The HTTP class allows retrieval of metrics available during the HTTP_REQUEST and HTTP_RESPONSE events.
IBMMQ	The IBMMQ class allows retrieval of metrics available during the IBMMQ_REQUEST and IBMMQ_RESPONSE events.
ICA	The ICA class allows retrieval of metrics available during the ICA_OPEN, ICA_AUTH, ICA_TICK, and ICA_CLOSE event.
ICMP	The ICMP class allows retrieval of metrics available during the ICMP_MESSAGE event.

Name	Description
IPAddress	The IPAddress class allows setting and retrieval of IP address attributes. IPAddress is the type of IP address properties available on the Flow class.
LDAP	The LDAP class allows retrieval of metrics available during the LDAP_ REQUEST and LDAP_RESPONSE events.
LLDP	The LLDP class allows retrieval of metrics available during LLDP_ FRAME events.
Memcache	The Memcache class allows retrieval of metrics available during the MEMCACHE_REQUEST and MEMCACHE_RESPONSE events.
MetricCycle	The MetricCycle class represents an interval where stats were published. It is valid on the events METRIC_CYCLE_BEGIN, METRIC_RECORD_COMMIT, and METRIC_CYCLE_END.
MetricRecord	The MetricRecord class allows access to the current set of metrics in METRIC_RECORD_COMMIT.
MongoDB	The MongoDB class allows retrieval of metrics available during the MONGODB_REQUEST and MONGODB_RESPONSE events.
Network	The Network class allows adding custom metrics at the global level.
NFS	The NFS class allows retrieval of metrics available during the NFS_ REQUEST and NFS_RESPONSE events.
Remote.HTTP	The Remote.HTTP class allows submission of HTTP REST requests.
Remote.MongoDB	The Remote.MongoDB class allows insertion, removal, and updating or documents in collections in MongoDB.
Remote.Syslog	The Remote.Syslog class allows creation of remote syslog message with specified content.
RemoteSyslog	(Deprecated. Use Remote.Syslog instead.) The RemoteSyslog class allows creation of remote syslog messages with specified content.
RTCP	The RTCP class allows for retrieval of metrics available during the RTCP_ MESSAGE event (version 4.1.22706+).
RTP	The RTP class allows for retrieval of metrics available during the RTP_OPEN, RTP_CLOSE, and RTP_TICK events (version 4.1.22706+).
Sampleset	The Sampleset class is used for representing sampleset metrics.
SDP	The SDP class allows for retrieval of Session Description Protocol (SDP) information during the SIP_REQUEST and SIP_RESPONSE events.
Session	The Session class allows for manipulation of objects in the global session table, a construct designed for passing objects across flows.
SIP	The SIP class allows retrieval of metrics available during the SIP_REQUEST and SIP_RESPONSE events (version 4.1.22706+).
SMPP	The SMPP class allows retrieval of metrics available during the SMPP_ REQUEST and SMPP_RESPONSE events.
SMTP	The SMTP class allows retrieval of metrics available during the SMTP_ REQUEST and SMTP_RESPONSE events.

Name	Description
SSL	The SSL class allows retrieval of metrics available during the SSL_OPEN, SSL_CLOSE, SSL_ALERT, SSL_RECORD, SSL_HEARTBEAT, and SSL_RENEGOTIATE events.
Telnet	The Telnet class allows retrieval of metrics available during the TELNET_MESSAGE event.
Topnset	The Topnset class is used for representing topnset metrics. A topnset metric contains a list of entries with keys and values. Values may be numbers, Datasets, Samplesets, or even other Topnsets.
Topnset Keys	Topnset Keys are objects that represent a property of Topnset.
TroubleGroup	The TroubleGroup class provides an interface for creating custom trouble groups.
Turn	Turn is a top-level object available in the FLOW_TURN event.
VLAN	The VLAN class represents a VLAN on the network. It has an `id` property representing the VLAN ID.
XML	The XML class allows XML parsing of data.

AAA

The AAA class allows the retrieval of metrics available during the `AAA_REQUEST` and `AAA_RESPONSE` events.

*Go to **Classes List**.*

Events

AAA_REQUEST: Event
Fires on every AAA request processed by the device.

*Go to **Events List**.*

AAA_RESPONSE: Event
Fires on every AAA response processed by the device.

*Go to **Events List**.*

Properties

authenticator: String (version 3.8.16027+)
The value of the authenticator field (RADIUS only).

avps: Array

avpLength: Number
The size of the AVP, expressed in bytes. This value includes the AVP header data, as well as the value.

id: Integer
The numeric ID of the attribute.

isGrouped: Boolean
Returns true if this is a grouped AVP (Diameter only).

name: String
A string name for the given AVP.

vendor: String
The vendor name for vendor AVPs (Diameter only).

value: String, Array, or Number
For simple AVPs, a string or numeric value. For grouped AVPs (Diameter only), an array of objects.

error: Boolean (`AAA_RESPONSE` only)
Returns true if there was an error. To retrieve the error details in Diameter, check `AAA.statusCode`. To retrieve the error details in RADIUS, check the AVP with code `18` (Reply-Message).

isDiameter: Boolean
Returns true if the request or response is Diameter.

isRadius: Boolean
Returns true if the request or response is RADIUS.

isRspAborted: Boolean (`AAA_RESPONSE` only)
Returns true if `AAA_RESPONSE` is aborted.

method: Number
Corresponds to the command code in either RADIUS or Diameter.

Diameter Command Codes

Command Name	Abbr.	Code
AA-Request	AAR	265
AA-Answer	AAA	265
Diameter-EAP-Request	DER	268
Diameter-EAP-Answer	DEA	268
Abort-Session-Request	ASR	274
Abort-Session-Answer	ASA	274
Accounting-Request	ACR	271
Credit-Control-Request	CCR	272
Credit-Control-Answer	CCA	272
Capabilities-Exchange-Request	CER	257
Capabilities-Exchange-Answer	CEA	257
Device-Watchdog-Request	DWR	280
Device-Watchdog-Answer	DWA	280
Disconnect-Peer-Request	DPR	282
Disconnect-Peer-Answer	DPA	282
Re-Auth-Request	RAR	258
Re-Auth-Answer	RAA	258
Session-Termination-Request	STR	275
Session-Termination-Answer	STA	275
User-Authorization-Request	UAR	300
User-Authorization-Answer	UAA	300
Server-Assignment-Request	SAR	301
Server-Assignment-Answer	SAA	301
Location-Info-Request	LIR	302
Location-Info-Answer	LIA	302
Multimedia-Auth-Request	MAR	303
Multimedia-Auth-Answer	MAA	303
Registration-Termination-Request	RTR	304
Registration-Termination-Answer	RTA	304
Push-Profile-Request	PPR	305
Push-Profile-Answer	PPA	305
User-Data-Request	UDR	306

Command Name	Abbr.	Code
User-Data-Answer	UDA	306
Profile-Update-Request	PUR	307
Profile-Update-Answer	PUA	307
Subscribe-Notifications-Request	SNR	308
Subscribe-Notifications-Answer	SNA	308
Push-Notification-Request	PNR	309
Push-Notification-Answer	PNA	309
Bootstrapping-Info-Request	BIR	310
Bootstrapping-Info-Answer	BIA	310
Message-Process-Request	MPR	311
Message-Process-Answer	MPA	311
Update-Location-Request	ULR	316
Update-Location-Answer	ULA	316
Authentication-Information-Request	AIR	318
Authentication-Information-Answer	AIA	318
Notify-Request	NR	323
Notify-Answer	NA	323

RADIUS Command Codes

Command Name	Code
Access-Request	1
Access-Accept	2
Access-Reject	3
Accounting-Request	4
Accounting-Response	5
Access-Challenge	11
Status-Server (experimental)	12
Status-Client (experimental)	13
Reserved	255

reqBytes: Number
The number of application-level request bytes.

reqL2Bytes: Number
The number of request L2 bytes.

reqPkts: Number
The number of request packets.

reqRTO: Number (`AAA_REQUEST` only)
The number of request RTOs.

roundTripTime: Number
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number
The number of application-level response bytes.

rspL2Bytes: Number
The number of response L2 bytes.

rspPkts: Number
The number of response packets.

rspRTO: Number (`AAA_RESPONSE` only)
The number of response RTOs.

statusCode: String (`AAA_RESPONSE` Diameter only)
A string representation of the AVP identifier `268` (Result-Code).

tprocess: Number (`AAA_RESPONSE` only)
The server processing time, expressed in milliseconds.

txId: Number
A value that corresponds to the hop-by-hop identifier in Diameter and `msg-id` in RADIUS.

ActiveMQ

The ActiveMQ class allows the retrieval of metrics available during the `ACTIVEMQ_MESSAGE` event. This is an implementation of the Java Messaging Service (JMS).

*Go to **Classes List**.*

Events

ACTIVEMQ_MESSAGE: Event
Fires on every JMS message processed by the device.

*Go to **Events List**.*

Properties

correlationId: String
The JMSCorrelationID field of the message.

expiration: Number
The JMSExpiration field of the message.

msg: **Buffer**
The message body. For TEXT_MESSAGE format messages, this returns the body of the message as a UTF-8 string. For all other message formats, this returns the raw bytes.

msgFormat: String
The message format. Possible values are:

- BYTES_MESSAGE
- MAP_MESSAGE
- MESSAGE
- OBJECT_MESSAGE
- STREAM_MESSAGE
- TEXT_MESSAGE
- BLOG_MESSAGE

msgId: String
The JMSMessageID field of the message.

totalMsgLength: Number
The length of the message, expressed in bytes.

persistent: Boolean
Returns true if the JMSDeliveryMode is PERSISTENT.

priority: Number
The JMSPriority field of the message.

- 0 is the lowest priority.
- 9 is the highest priority.
- 0-4 are gradations of normal priority.
- 5-9 are gradations of expedited priority.

properties: Object
Zero or more properties attached to the message. The keys are arbitrary strings and the values may be booleans, numbers, or strings.

queue: String
The JMSDestination field of the message.

receiverIsBroker: Boolean
Returns true if the flow-level receiver of the message is a broker.

receiverBytes: Number
The number of application-level bytes from the receiver.

receiverIsBroker: Boolean
Returns true if the flow-level receiver of the message is a broker.

receiverL2Bytes: Number
The number of L2 bytes from the receiver.

receiverPkts: Number
The number of packets from the receiver.

receiverRTO: Number
The number of RTOs from the receiver.

redeliveryCount: Number
The number of redeliveries.

replyTo: String
The JMSReplyTo field of the message, converted to a string.

roundTripTime: Number
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

senderBytes: Number
The number of application-level bytes from the sender.

senderIsBroker: Boolean
Returns true if the flow-level sender of the message is a broker.

senderL2Bytes: Number
The number of L2 bytes from the sender.

senderPkts: Number
The number of packets from the sender.

senderRTO: Number
The number of RTOs from the sender.

timeStamp: Number
The time when the message was handed off to a provider to be sent, expressed in GMT. This is the JMSTimestamp field of the message.

totalMsgLength: Number
The length of the message, expressed in bytes.

Application

The Application class allows you to create new applications and add custom metrics at the application level. Applications are user-defined, arbitrary groups of traffic. Applications are defined through triggers only; they cannot be defined in the UI. Application names must use ASCII characters only. Refer to *Getting Started with Application Inspection Triggers* for more information about applications.

*Go to **Classes List**.*

Methods

commit (): void

Commits metrics on an application. Applications are automatically created the first time they are referenced. For instance, to create an application named "myApp", use the following syntax:

```
Application("myApp").commit();
```

The application key (e.g., "myApp") must be a string and not an integer. For example, "1020" is treated the same as 1020. Strings that cannot be used include `null` and `Object Object`.

The following table shows which event(s) to use to create applications.

Application Component	Event
AAA	**AAA_REQUEST: Event** *and* **AAA_RESPONSE: Event**
DB	**DB_RESPONSE: Event**
DNS	**DNS_REQUEST: Event** *and* **DNS_RESPONSE: Event**
FIX	**FIX_REQUEST: Event** *and* **FIX_RESPONSE: Event**
FTP	**FTP_RESPONSE: Event**
HTTP	**HTTP_RESPONSE: Event**
IBMMQ	**IBMMQ_REQUEST: Event** *and* **IBMMQ_RESPONSE: Event**
ICA	**ICA_TICK: Event** *and* **ICA_CLOSE: Event**
L4	**TCP_OPEN: Event** *or* **FLOW_CLASSIFY: Event**
LDAP	**LDAP_REQUEST: Event** *and* **LDAP_RESPONSE: Event**
Memcache	**MEMCACHE_REQUEST: Event** *and* **MEMCACHE_RESPONSE: Event**
NAS	**CIFS_RESPONSE: Event** *or* **NFS_RESPONSE: Event**
RTP	**RTP_TICK: Event**
RTCP	**RTCP_MESSAGE: Event**
SIP	**SIP_REQUEST: Event** *and* **SIP_RESPONSE: Event**
SSL	**SSL_RECORD: Event (version 3.8.15910+)** *and* **SSL_CLOSE: Event**
SMTP	**SMTP_RESPONSE: Event**

NOTE: In `TCP_OPEN` and `FLOW_CLASSIFY`, `Application.commit` is not valid. Instead use `Flow.setApplication("Report Pools")` to create and assign an L4 application to the flow. `Flow.setApplication()` can be used from any device event.

Use the following methods to record custom metrics associated with applications. Refer to the **ExtraHop Datatypes** section for an overview of the data types.

- **metricAddCount**(*metric_name*:String, *count*:Number): void
- **metricAddDataset**(*metric_name*:String, *val*:Number, [*freq*:Number]): void
- **metricAddDetailCount**(*metric_name*:String, *key*:String | IPAddress, *count*:Number): void
- **metricAddDetailSnap**(*metric_name*:String, *key*:String | IPAddress, *count*:Number): void
- **metricAddDetailDataset**(*metric_name*:String, *key*:String | IPAddress, *val*:Number, [*freq*:Number]): void
- **metricAddDetailMax**(*metric_name*:String, *key*:String | IPAddress, *val*:Number) void
- **metricAddDetailSampleset**(*metric_name*:String, *key*:String | IPAddress, *val*:Number): void
- **metricAddMax**(*metric_name*:String, *val*:Number): void
- **metricAddSampleset**(*metric_name*:String, *val*:Number): void
- **metricAddSnap**(*metric_name*:String, *count*:Number): void

The above methods cannot not be called on Application directly, but must be called on specific Application instances. For example:

```
Application("myApp").metricAddCount("requests", 1);
Application("myApp").commit();
```

Notes:

- When NaN is passed to a `metricAdd*` function, it is silently discarded.
- All count parameters for `metricAdd*` functions only accept a non-zero, positive integer between 1 and 2^64.

Properties

id: String (version 4.0.21092+)
The application ID.

See Also

- **Example: Trigger-Based Application Definition**.

Buffer

A buffer is an object with the characteristics of an array. Each element in the array is a number between 0 and 255, representing one byte. It has a length property (the number of items in an array) and a square bracket operator. It contains the `toString()`, `slice()`, `unpack()`, and `decode()` methods.

Encrypted payload is not decrypted for TCP and UDP payload analysis.

`UDP_PAYLOAD` requires a matching string but `TCP_PAYLOAD` does not. If you do not specify a matching string for `TCP_PAYLOAD`, the trigger fires one time after the first *n* bytes of payload.

*Go to **Classes List**.*

Methods

Buffer.decode(*type*:String): String
Interprets the contents of the Buffer and returns a string with one of the following options:

- utf-8
- ucs2
- hex

Buffer.toString(): String
Converts the Buffer to a string.

Buffer.slice(*start*: Number, [*end*: Number]): Buffer
Returns the specified bytes in a Buffer as a new Buffer. Bytes are selected starting at the given **start** argument and ending at (but not including) the **end** argument.

start: Number
Integer that specifies where to start the selection. Use negative numbers to select from the end of a Buffer. This is zero-based.

end: Number
Optional integer that specifies where to end the selection. If omitted, all elements from the start position and to the end of the Buffer will be selected. Use negative numbers to select from the end of a Buffer. This is zero-based.

Buffer.unpack(*format*:String, [*offset*:Number]): Array
Processes binary or fixed-width data from any Buffer object, such as one returned by `HTTP.payload` or `TCP.payload`, according to the given format string and, optionally, at the specified offset. The result is a JavaScript array containing unpacked fields, even if it contains exactly one item.

Notes:

- `Buffer.unpack` uses big-endian, standard alignment, by default.
- The format does not have to consume the entire buffer.
- Null bytes are not included in unpacked strings. For example: `buf.unpack('4s')[0] -> 'foo'`.
- The z format character represents variable-length, null-terminated strings. If the last field is z, the string is produced whether or not the null character is present.
- An exception is throw when all of the fields cannot be unpacked because the buffer does not contain enough data.

Format Strings

Format	C Type	JavaScript Type	Standard Size
x	pad type	no value	
A	struct in6_addr	IPAddress	16
a	struct in_addr	IPAddress	4
b	signed char	string of length 1	1
B	unsigned char	number	1
?	_Bool	boolean	1
h	short	number	2
H	unsigned short	number	2
i	int	number	4
I	unsigned int	number	4
l	long	number	4
L	unsigned long	number	4
q	long long	number	8
Q	unsigned long long	number	8
f	number	number	4
d	double	number	4
s	char[]	string	
z	char[]	string	

The following is an example of a `UDP_PAYLOAD` trigger that parses NTP with `Buffer.unpack`:

```
var buf = Flow.server.payload,
    flags,
    values,
    fmt,
    offset = 0,
    ntpData = {},
    proto = Flow.l7proto;
if ((proto !== 'NTP') || (buf === null)) {
    return;
}

// Parse individual flag values from flags byte
function parseFlags(flags) {
    return {
        'LI': flags >> 6,
        'VN': (flags & 0x3f) >> 3,
        'mode': flags & 0x7
    };
```

```
}

// Convert from NTP short format
function ntpShort(n) {
    return n / 65536.0;
}


// Convert integral part of NTP timestamp format to Date
function ntpTimestamp(n) {
    // NTP dates start at 1900, subtract the difference
    // and convert to milliseconds.
    var ms = (n - 0x83aa7e80) * 1000;
    return new Date(ms);
}

// First part of NTP header:
fmt = ('B' + // Flags (LI, VN, mode)
       'B' + // Stratum
       'b' + // Polling interval (signed)
       'b' + // Precision (signed)
       'I' + // Root delay
       'I'); // Root dispersion

values = buf.unpack(fmt);

offset += values.bytes;

flags = parseFlags(values[0]);
if (flags.VN !== 4) {
    // Expecting NTPv4.
    return;
}

ntpData.flags = flags;
ntpData.stratum = values[1];
ntpData.poll = values[2];
ntpData.precision = values[3];
ntpData.rootDelay = ntpShort(values[4]);
ntpData.rootDispersion = ntpShort(values[5]);
// The next field, the reference ID, depends upon the stratum field.
switch (ntpData.stratum)
{
case 0:
case 1:
    // Identifier string (4 bytes), and 4 NTP timestamps in two parts
    fmt = '4s8I';
    break;
default:
    // Unsigned int (based on IP), and 4 NTP timestamps in two parts
```

```
    fmt = 'I8I';
    break;
}
// Passing in offset allows us to continue parsing where we left off.
values = buf.unpack(fmt, offset);
ntpData.referenceId = values[0];

// We only use the integral parts of the timestamp here.
ntpData.referenceTimestamp = ntpTimestamp(values[1]);
ntpData.originTimestamp = ntpTimestamp(values[3]);
ntpData.receiveTimestamp = ntpTimestamp(values[5]);
ntpData.transmitTimestamp = ntpTimestamp(values[7]);

debug('NTP data:' + JSON.stringify(ntpData, null, 4));
```

Properties

Buffer.length: Number
The number of bytes in the Buffer.

Flow.server.payload: Buffer
Returns a Buffer containing the server payload. If no payload exists, null is returned.

Flow.client.payload: Buffer
Returns a Buffer containing the client payload. If no payload exists, null is returned.

CIFS

The CIFS class allows the retrieval of metrics available during the `CIFS_REQUEST` and `CIFS_RESPONSE` events.

*Go to **Classes List**.*

Events

CIFS_REQUEST: Event
Fires on every CIFS request processed by the device.

*Go to **Events List**.*

CIFS_RESPONSE: Event
Fires on every CIFS response processed by the device.

*Go to **Events List**.*

Properties

accessTime: Number (`CIFS_RESPONSE` only)
The time it took for the server to access a file on disk, expressed in milliseconds. For CIFS, this is the time from the first READ command in a CIFS flow until the first byte of the payload of its response.

error: String (`CIFS_RESPONSE` only)
The detailed error message recorded by the ExtraHop system.

isCommandFileInfo: Boolean
Returns true if the command is a file info command.

isCommandLock: Boolean
Returns true if the command is a locking command.

isCommandRead: Boolean
Returns true if the command is a read.

isCommandWrite: Boolean
Returns true if the command is a write.

method: String
The CIFS method (appears under Methods in the UI).

reqBytes: Number (`CIFS_RESPONSE` only)
The number of L4 request bytes.

reqL2Bytes: Number (`CIFS_RESPONSE` only)
The number of L2 request bytes.

reqPkts: Number (`CIFS_RESPONSE` only)
The number of request packets.

reqRTO: Number (`CIFS_RESPONSE` only)
The number of request RTOs.

reqSize: Number
The size of the request, expressed in bytes.

resource: String
The share, path, and filename, concatenated together.

roundTripTime: Number (`CIFS_RESPONSE` only)

The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number (`CIFS_RESPONSE` only)
The number of L4 response bytes.

rspL2Bytes: Number (`CIFS_RESPONSE` only)
The number of L2 response bytes.

rspPkts: Number (`CIFS_RESPONSE` only)
The number of response packets.

rspRTO: Number (`CIFS_RESPONSE` only)
The number of response RTOs.

rspSize: Number (`CIFS_RESPONSE` only)
The size of the response, expressed in bytes.

share: String
The name of the share to which the user is connected.

statusCode: Number (`CIFS_RESPONSE` only)
The numeric status code of the response (SMB2 only).

user: String
The user name, if available. In some cases, such as when the login event was not visible or the access was anonymous, the user name is not available.

See Also

- **Example: CIFS Trigger**

Dataset

The dataset class is used to represent dataset metrics. It provides access to the raw dataset values and provides an interface for computing percentiles.

*Go to **Classes List**.*

Instance Methods

percentile(...): Array or Number

Accepts a list of percentiles (either as an array or as multiple arguments) to compute and returns the computed percentile values for the dataset. If passed a single numeric argument, a number is returned. Otherwise an array is returned. The arguments must be in ascending order with no duplicates. Floating point values are allowed (e.g., 99.99).

Instance Properties

entries: Array

An array of objects with frequency and value attributes. This is analogous to a frequency table where there is a set of values and the number of times each value was observed.

DB

The DB class allows the retrieval of metrics available during the `DB_REQUEST` and `DB_RESPONSE` events.

*Go to **Classes List**.*

Events

DB_REQUEST: Event
Fires on every database request processed by the device.

*Go to **Events List**.*

DB_RESPONSE: Event
Fires on every database response processed by the device.

*Go to **Events List**.*

Properties

correlationId: Number (version 4.1.4.24670)
Returns the correlation ID for DB2 applications. Returns NULL for non-DB2 applications.

database: String
The database instance. In some cases, such as when login events are encrpyted, the database name is not available.

error: String (`DB_RESPONSE` only)
The detailed error message recorded by the ExtraHop system.

errors: Array of strings (`DB_RESPONSE` only)
An array of error messages recorded by the ExtraHop system.

isReqAborted: Boolean
Returns true if the connection is closed before the DB request is complete.

isRspAborted: Boolean (`DB_RESPONSE` only)
Returns true if the connection is closed before the DB response is complete.

method: String
Database method (appears under **Methods** in the user interface).

params: Array (`DB_REQUEST` only)
List of RPC parameters (only available for Microsoft SQL and DB2 databases). This is an array of objects and each object has the following properties:

name: String
The optional name of the supplied RPC parameter and the value is the value of the parameter.

reqSize: Number
The size of the request record at L7, expressed in bytes.

value: String | Number
If the value is not a text, integer, or a time/date field, it will be canonicalized into hex/ASCII form.

procedure: String
The stored procedure name (appears under **Methods** in the user interface).

reqBytes: Number (`DB_RESPONSE` only)
The number of L4 request bytes.

reqL2Bytes: Number (`DB_RESPONSE` only)
The number of L2 request bytes.

reqPkts: Number (`DB_RESPONSE` only)
The number of request packets.

reqRTO: Number (`DB_RESPONSE` only)
The number of request RTOs.

reqSize: Number
The size of the request record at L7, expressed in bytes.

reqTimeToLastByte: Number
The time from the first byte of the request until the last byte of the request, expressed in milliseconds. Returns NaN on malformed and aborted requests or expired flows.

roundTripTime: Number (`DB_RESPONSE` only)
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number (`DB_RESPONSE` only)
The number of L4 response bytes.

rspL2Bytes: Number (`DB_RESPONSE` only)
The number of L2 response bytes.

rspPkts: Number (`DB_RESPONSE` only)
The number of response packets.

rspRTO: Number (`DB_RESPONSE` only)
The number of response RTOs.

rspSize: Number (`DB_RESPONSE` only)
The size of the response record at L7, expressed in bytes.

rspTimetoFirstByte: Number (`DB_RESPONSE` only)
The time from the first byte of the request until the first byte of the response, expressed in milliseconds. Returns NaN on malformed and aborted responses or expired flows.

rspTimeToLastByte: Number (`DB_RESPONSE` only)
The time from the first byte of the request until the last byte of the response, expressed in milliseconds. Returns NaN on malformed and aborted responses or expired flows.

statement: String (`DB_REQUEST` only)
The full SQL statement (may not be available for all DB methods).

table: String (Sybase IQ database only)
The name of the database table specified in the current statement. This field is empty if there is no table name in the request.

tprocess: Number (`DB_RESPONSE` only)
The server processing time, expressed in milliseconds (equivalent to `rspTimeToFirstByte - reqTimeToLastByte`). Returns NaN on malformed and aborted responses or expired flows.

user: String
The user name, if available. In some cases, such as when login events are encrypted, the user name is not available.

See Also

- **Example: Database Trigger**
- **Example: Trigger-Based Application Definition**

Device

The Device class allows retrieval of device attributes and adding custom metrics at the device level.

*Go to **Classes List**.*

Instance Methods

The following method is present only on instances of the Device class:

Device (id: String)

Constructor for the Device object that accepts one parameter, a unique 16-character string ID. If supplied with an ID from an existing device, the constructor creates a copy of that object with all the properties. Committing metrics on this object with the metricAdd* functions will persist them in the datastore. For example:

```
myDevice = new Device(Flow.server.device.id);
debug("myDevice MAC:  " + myDevice.hwaddr);
```

Use the following methods to record custom metrics associated with devices. Refer to the ExtraHop Datatypes section for an overview of the data types.

- **metricAddCount**(metric_name:String, count:Number): void
- **metricAddDataset**(metric_name:String, val:Number, [freq:Number]): void
- **metricAddDetailCount**(metric_name:String, key:String|IPAddress, count:Number): void
- **metricAddDetailDataset**(metric_name:String, key:String|IPAddress, val:Number, [freq:Number]): void
- **metricAddDetailMax**(metric_name:String, key:String|IPAddress, val:Number): void
- **metricAddDetailSampleset**(metric_name:String, key:String|IPAddress, val:Number): void
- **metricAddDetailSnap**(metric_name:String, key:String|IPAddress, count:Number): void
- **metricAddMax**(metric_name:String, val:Number): void
- **metricAddSampleset**(metric_name:String, val:Number): void
- **metricAddSnap**(metric_name:String, count:Number): void

Notes:

- Calling `Device.metricAdd*` functions records metrics for both devices on the flow, regardless of how the triggers are assigned.
- Calling `Flow.client.device.metricAdd*` functions records metrics only for the client device, regardless of whether the trigger is assigned to the client or the server.
- Calling `Flow.server.device.metricAdd*` functions records metrics only for the server device, regardless of whether the trigger is assigned to the client or the server.
- Adding a metric to the Network object makes it available to network-level custom pages.
- Adding a metric to the Device object makes it available to custom pages on that device.
- The `metricAddMax` and `metricAddDetailMax` functions commit metrics that preserve a maximum. For instance, use the `metricAddMax` function to record maximum values of database server processing times overtime.
- If the information is unavailable or not applicable, the value of a property will be null where the type is normally a String and NaN where the type is normally a Number.
- When NaN is passed to a `metricAdd*` function, it is silently discarded.
- All count parameters for `metricAdd*` functions only accept a non-zero, positive integer between 1 and 2^64.

Instance Properties

The following properties allow retrieval of device attributes and are present only on instances of the Device class.

cpdName: String
The CDP name associated with the device, if present.

dhcpName: String
The DHCP name associated with the device, if present.

discoverTime: Number
The last time the capture process discovered the device (not the original discover time), expressed in milliseconds since the epoch (January 1, 1970). Previously discovered devices may be rediscovered by the capture process if they go idle and later become active again, or if the capture process is restarted.

To take trigger action only on the initial discovery of a device, see the **NEW_DEVICE: Event** trigger event.

dnsNames: Array
The DNS names associated with the device, if present.

hasTrigger: Boolean
Returns true if the currently executing trigger is configured on the Device object on which the `hasTrigger` property is called. For all trigger events with an associated **FLOW** object, at least one of the Device objects in the flow will have its `hasTrigger` property set to true.

hwaddr: String
The MAC address of the device, if present.

id: String
The 16-character unique ID of the device, as shown in the ExtraHop UI in the **Device**>> **Device ID** field.

ipaddrs: Array
An array of **IPAddress**objects representing the device's known IP addresses. This will always be an array of one IP Address for L3 devices.

isGateway: Boolean
Returns true if the device is a gateway.

isL3: Boolean (version 4.0.21090+)
Returns true if the device is an L3 device.

netbiosName: String
The NetBIOS name associated with the device, if present.

vlanId: Number (version 4.0.21091+)
The VLAN ID for the device.

See Also

- **Example: CIFS Trigger**
- **Example: Customer ID Header**
- **Example: Database Trigger**
- **Example: Device Discovery Notification**
- **Example: HTTP Header Object**
- **Example: Memcache Hits and Misses**

- **Example: Memcache Key Parsing**
- **Example: Parse Custom POS Messages with Universal Payload Analysis**
- **Example: Use the Metric Cycle Store**

Discover

The Discover class provides access to newly discovered VLANs, devices, or applications on the `NEW_VLAN`, `NEW_DEVICE`, and `NEW_APPLICATION` events.

Events

NEW_APPLICATION: Event
Fires when an application is first discovered.

NEW_DEVICE: Event
Fires when a device is first discovered.

NEW_VLAN: Event
Fires when a VLAN is first discovered.

Properties

application: **Application**(`NEW_APPLICATION` only)
A newly discovered application.

device: **Device**(`NEW_DEVICE` only)
A newly discovered device.

vlan: **VLAN**(`NEW_VLAN` only)
A newly discovered VLAN.

See Also

- **Example: Device Discovery Notification**

DNS

The DNS class allows retrieval of metrics available during the DNS_REQUEST and DNS_RESPONSE events.

*Go to **Classes List**.*

Events

DNS_REQUEST: Event
Fires on every DNS request processed by the device.

*Go to **Events List**.*

DNS_RESPONSE: Event
Fires on every DNS response processed by the device.

*Go to **Events List**.*

Properties

answers: Array (DNS_RESPONSE only)
An array of objects corresponding to answer resource records. The objects have the following properties:

data: String
The value of data depends on the type and will be null for unsupported record types. Supported record types include:

- A
- AAAA
- NS
- PTR
- CNAME
- MX
- SRV
- SOA
- TXT

name: String
Record name.

ttl: Number
Time-to-live.

type: String
DNS record type.

error: String (DNS_RESPONSE only)
Detailed error message recorded by the ExtraHop system.

isAuthoritative: Boolean (DNS_RESPONSE only)
Returns true if the authoritative answer is set in the response.

isReqTimeout: Boolean (DNS_REQUEST only)
Returns true if the request timed out.

isRspTruncated: Boolean (DNS_RESPONSE only)
Returns true if the response is truncated.

opcode: String
DNS opcode.

DNS Opcodes

OpCode	Name
0	Query
1	IQuery (Inverse Query - Obsolete)
2	Status
3	Unassigned
4	Notify
5	Update
6-15	Unassigned

qname: String
Corresponds to the hostname queried.

qtype: String
The DNS request record type.

reqBytes: Number (`DNS_REQUEST` only)
The number of application-level request bytes.

reqL2Bytes: Number (`DNS_REQUEST` only)
The number of request L2 bytes.

reqPkts: Number (`DNS_REQUEST` only)
The number of request packets.

rspBytes: Number (`DNS_RESPONSE` only)
The number of response bytes.

rspL2Bytes: Number (`DNS_RESPONSE` only)
The number of response L2 bytes.

rspPkts: Number (`DNS_RESPONSE` only)
The number of application-level response bytes.

tprocess: Number (`DNS_RESPONSE` only)
The server processing time, expressed in milliseconds. Returns NaN on malformed and aborted responses or expired flows.

FIX

The FIX class allows retrieval of metrics available during the `FIX_REQUEST` and `FIX_RESPONSE` events.

*Go to **Classes List**.*

Events

FIX_REQUEST: Event
Fires on every FIX request processed by the device.

*Go to **Events List**.*

FIX_RESPONSE: Event
Fires on every FIX response processed by the device.

*Go to **Events List**.*

Note: `FIX_RESPONSE` is matched with request based on order id. There is no one-to-one correlation between request and response. There could be requests without a response and sometimes data is pushed to the client. That limits request data availability on response event, however the session table could be used to solve any complex scenarios like submission order id, etc.

Properties

fields: Array
A list of FIX fields. Since they are text-based, the key-value protocol fields are exposed as an array of objects with name and value properties containing strings. For example:

```
8=FIX.4.2<SOH>9=233<SOH>35=G<SOH>34=206657...
```

translates to:

```
{"BeginString": "FIX.4.2", "BodyLength": "233", "MsgType": "G", "MsgSeqNum":
"206657"}
```

Key string representation is translated, if possible. With extensions, a numeric representation is used. For example, it is not possible to determine 9178=0 (as seen in actual captures). The key is instead translated to "9178". Fields are extracted after message length and version fields are extracted all the way to the checksum (last field). The checksum is not extracted.

For another example, the trigger `debug(JSON.stringify(FIX.fields));` shows the following fields:

```
[
{"name":"MsgType","value":"0"},
{"name":"MsgSeqNum","value":"2"},
{"name":"SenderCompID","value":"AA"},
{"name":"SendingTime","value":"20140904-03:49:58.600"},
{"name":"TargetCompID","value":"GG"}
]
```

To debug and print all FIX fields, enable debugging on the trigger and use the following code:

```
var fields = '';
for (var i = 0; i < FIX.fields.length; i++) {
  fields += '"' + FIX.fields[i].name + '" : "' + FIX.fields[i].value +
'"\n';
}
debug(fields);
```

The following output prints to the trigger's Runtime Log:

```
"MsgType" : "5"
"MsgSeqNum" : "3"
"SenderCompID" : "GRAPE"
"SendingTime" : "20140905-00:10:23.814"
"TargetCompID" : "APPLE"
```

msgType: String
The value of the MessageCompID key.

reqBytes: Number
The umber of application-level request bytes.

reqL2Bytes: Number
The number of request L2 bytes.

reqPkts: Number
The number of request packets.

reqRTO: Number
The number of request RTOs.

rspBytes: Number
The number of application-level response bytes.

rspL2Bytes: Number
The number of response L2 bytes.

rspPkts: Number
The number of response packets.

rspRTO: Number
The number of response RTOs.

sender: String
The value of the SenderCompID key.

target: String
The value of the TargetCompID key.

version: String
The protocol version.

FLOW

Flow refers to a TCP or UDP connection between two endpoints. The Flow class provides access to elements of these conversations, such as endpoint identities and flow age. It also contains a flow store designed to pass objects from request to response on the same flow.

Using a combination of L7 payload analysis, observation of TCP handshakes, and port number-based heuristics, the ExtraHop system identifies the L7 protocol and the client and server roles for the endpoints in a flow.

*Go to **Classes List**.*

NOTE: Deprecated Flow properties are listed at the end of this section.

FLOW_CLASSIFY: Event

Fires once per flow when the L7 protocol of the flow has been determined.

For TCP flows, the `FLOW_CLASSIFY` event fires after the `TCP_OPEN` event.

For certain L7 protocols, the nature of a flow changes over its lifetime (for example, tunneling over HTTP or switching from SMTP to SMTP-TLS). In those cases, `FLOW_CLASSIFY` will fire again after the change.

For flows associated with ExtraHop-monitored protocols (such as HTTP), L7 trigger events (such as `HTTP_REQUEST` and `HTTP_RESPONSE`) will additionally fire for the flow. The Flow object properties and methods discussed in this section are available to every L7 trigger event associated with the flow.

The `FLOW_CLASSIFY` event is useful for initiating an action on a flow when it can be done, based on the earliest knowledge of a flow (for example, client and server IPs, client and server ports, L7 protocol). Actions commonly taken during this trigger event include starting a packet capture via the `captureStart()` method or associating the flow with an application container using `setApplication()`. It is best practice to wait before taking any action until more information becomes known about the flow via a future L7 trigger event (for example, once it's known that the flow is an HTTP request for a specific URI).

*Go to **Events List**.*

Once the `FLOW_CLASSIFY` trigger event fires, two or more of the following data elements are available, depending on the subsequent trigger event.

Event	Client / Server	Sender / Receiver
AAA_REQUEST	yes	yes
AAA_RESPONSE	yes	yes
ACTIVEMQ_MESSAGE	no	yes
CIFS_REQUEST	yes	yes
CIFS_RESPONSE	yes	yes
DB_REQUEST	yes	yes
DB_RESPONSE	yes	yes
DNS_REQUEST	yes	yes
DNS_RESPONSE	yes	yes

Event	Client / Server	Sender / Receiver
HTTP_REQUEST	yes	yes
HTTP_RESPONSE	yes	yes
IBMMQ_REQUEST	yes	yes
IBMMQ_RESPONSE	yes	yes
ICA_AUTH	yes	no
ICA_CLOSE	yes	no
ICA_OPEN	yes	no
ICA_TICK	yes	no
FIX_REQUEST	yes	yes
FIX_RESPONSE	yes	yes
FLOW_CLASSIFY	yes	no
FLOW_DETACH	yes	no
FLOW_TICK	yes	no
FLOW_TURN	yes	no
FTP_REQUEST	yes	yes
FTP_RESPONSE	yes	yes
HL7_REQUEST	yes	yes
HL7_RESPONSE	yes	yes
ICMP_MESSAGE	no	yes
LDAP_REQUEST	yes	yes
LDAP_RESPONSE	yes	yes
MEMCACHE_REQUEST	yes	yes
MEMCACHE_RESPONSE	yes	yes
MONGODB_REQUEST	yes	yes
MONGODB_RESPONSE	yes	yes
NFS_REQUEST	yes	yes
NFS_RESPONSE	yes	yes
RTCP_MESSAGE	no	yes
RTP_CLOSE	no	yes
RTP_OPEN	no	yes
RTP_TICK	no	yes
SIP_REQUEST	yes	yes
SIP_RESPONSE	yes	yes

Event	Client / Server	Sender / Receiver
SMPP_REQUEST	yes	yes
SMPP_RESPONSE	yes	yes
SMTP_REQUEST	yes	yes
SMTP_RESPONSE	yes	yes
SSL_ALERT	yes	yes
SSL_CLOSE	yes	no
SSL_HEARTBEAT	yes	yes
SSL_OPEN	yes	no
SSL_PAYLOAD	yes	yes
SSL_RECORD	yes	yes
SSL_RENEGOTIATE	yes	no
TCP_CLOSE	yes	no
TCP_DESYNC	yes	no
TCP_OPEN	yes	no
TCP_PAYLOAD	yes	yes
UDP_PAYLOAD	yes	yes
TELNET_MESSAGE	yes	yes

client

device: **Device**
The device object associated with the client. For instance, to access the MAC address of the client, user `Flow.client.device.hwaddr`.

equals: Boolean
Performs an equality test between **Device** and **IPAddress** objects.

ipaddr: **IPAddress**
The IP address object associated with the client.

isAborted: Boolean (version 3.10.18355+)
Returns true if the client has aborted a TCP flow by issuing a TCP reset (RST). This condition may be detected in the `TCP_CLOSE` event and in any impacted L7 events (for example, `HTTP_REQUEST` or `DB_RESPONSE`).

Notes about aborts:

- An L4 abort occurs when a TCP connection is closed with a RST instead of a graceful shutdown.
- An L7 response abort occurs when a connection closes while in the middle of a response. This can be due to a RST, a graceful FIN shutdown, or an expiration.
- An L7 request about occurs when a connection closes in the middle of a request. This can also be due to a RST, a graceful FIN shutdown, or an expiration.

isShutdown: Boolean (version 4.0.21527+)

Returns true if the client initiated the shutdown of the TCP Connection.

port: Number
The port number used by the client in the flow.

receiver

device: **Device**
The device object associated with the receiver. For instance, to access the MAC address of the receiver, user `Flow.receiver.device.hwaddr`.

equals: Boolean
Performs an equality test between **Device** and **IPAddress** objects.

ipaddr: **IPAddress**
The IP address object associated with the receiver.

isAborted: Boolean (version 3.10.18355+)
Returns true if the receiver has aborted a TCP flow by issuing a TCP reset (RST). This condition may be detected in the `TCP_CLOSE` event and in any impacted L7 events (for example, `HTTP_REQUEST` or `DB_RESPONSE`).

Notes about aborts:

- An L4 abort occurs when a TCP connection is closed with a RST instead of a graceful shutdown.
- An L7 response abort occurs when a connection closes while in the middle of a response. This can be due to a RST, a graceful FIN shutdown, or an expiration.
- An L7 request about occurs when a connection closes in the middle of a request. This can also be due to a RST, a graceful FIN shutdown, or an expiration.

isShutdown: Boolean (version 4.0.21527+)
Returns true if the receiver initiated the shutdown of the TCP Connection.

port: Number
The port number used by the receiver in the flow.

sender

device: **Device**
The device object associated with the sender. For instance, to access the MAC address of the sender, use `Flow.sender.device.hwaddr`.

equals: Boolean
Performs an equality test between **Device**and **IPAddress** objects.

ipaddr: **IPAddress**
The IP address object associated with the sender.

isAborted: Boolean (version 3.10.18355+)
Returns true if the sender has aborted a TCP flow by issuing a TCP reset (RST). This condition may be detected in the `TCP_CLOSE` event and in any impacted L7 events (for example, `HTTP_REQUEST` or `DB_RESPONSE`).

Notes about aborts:

- An L4 abort occurs when a TCP connection is closed with a RST instead of a graceful shutdown.
- An L7 response abort occurs when a connection closes while in the middle of a response. This can be due to a RST, a graceful FIN shutdown, or an expiration.
- An L7 request about occurs when a connection closes in the middle of a request. This can also be due to a RST, a graceful FIN shutdown, or an expiration.

isShutdown: Boolean (version 4.0.21527+)
Returns true if the sender initiated the shutdown of the TCP connection.

port: Number
The port number used by the sender in the flow

server

device: **Device**
The device object associated with the server. For instance, to access the MAC address of the server, use `Flow.server.device.hwaddr`.

equals: Boolean
Performs an equality test between **Device**and **IPAddress** objects.

ipaddr: **IPAddress**
The IP address object associated with the server.

isAborted: Boolean (version 3.10.18355+)
Returns true if the server has aborted a TCP flow by issuing a TCP reset (RST). This condition may be detected in the `TCP_CLOSE` event and in any impacted L7 events (for example, `HTTP_REQUEST` or `DB_RESPONSE`).

Notes about aborts:

- An L4 abort occurs when a TCP connection is closed with a RST instead of a graceful shutdown.
- An L7 response abort occurs when a connection closes while in the middle of a response. This can be due to a RST, a graceful FIN shutdown, or an expiration.
- An L7 request about occurs when a connection closes in the middle of a request. This can also be due to a RST, a graceful FIN shutdown, or an expiration.

isShutdown: Boolean (version 4.0.21527+)
Returns true if the server initiated the shutdown of the TCP connection.

port: Number
The port number used by the server in the flow.

By default, the ExtraHop system uses loosely-initiated protocol classification, so it will try to classify flows even after the connection was initiated. Loose initiation can be turned off for ports that do not always carry the protocol traffic (e.g., the wildcard port 0). For such flows, `device1`, `port1`, and `ipaddr1`represent the device with the numerically lower IP address and `device2`, `port2`, and `ipaddr2` represent the device with the numerically higher IP address.

device1: **Device**

The device object associated with the device with a numerically lower IP address. For instance, to access the MAC address of the server, use `Flow.device1.hwaddr`.

equals: Boolean
Performs an equality test between device objects.

device2: **Device**
The device object associated with the device with a numerically higher IP address. For instance, to access the MAC address of the server, use `Flow.device2.hwaddr`.

equals: Boolean
Performs an equality test between device objects.

ipaddr1: **IPAddress**
The IP address object associated with the device with the numerically lower IP address.

equals: Boolean
Performs an equality test between IPAddress objects.

ipaddr2: **IPAddress**
The IP address object associated with the device with the numerically higher IP address.

equals: Boolean
Performs an equality test between IPAddress objects.

port1: Number
The port number used by the device with the numerically lower IP address.

port2: Number
The port number used by the device with the numerically higher IP address.

The following properties and methods apply to both types of flows.

age: Number
The time elapsed since the flow was initiated, expressed in seconds.

captureStart(**name**:String, [**options**:Object]: String (version 3.8.16174+)
Initiates a Precision Packet Capture for the flow and returns a unique identifier of the packet capture (a decimal number as a string). Returns null if the packet capture fails to start.

name: String
The name of the packet capture file.

- The maximum length is 256 characters
- The system creates a separate capture for each flow.
- Capture files with the same name are differentiated by timestamps.

options: Object
Omit any of the options to indicate unlimited size for that option. "Lookback buffer" refers to packets captured before the call to `captureStart()`. All options apply to the entire flow except the "lookback" options which apply only to the part of the flow before the trigger event that started the packet capture.

maxBytes: Number
The total maximum number of bytes.

maxBytesLookback: Number
The maximum number of bytes from the lookback buffer.

MaxDurationMSec: Number

The maximum duration of the packet capture, expressed in milliseconds. For global packet captures, the default is 1000 milliseconds.

maxPackets: Number
The total maximum number of packets.

maxPacketsLookback: Number
The maximum number of packets from the lookback buffer.

The following is an example of Flow.captureStart():

```
// EVENT: HTTP_REQUEST
// capture facebook HTTP traffic flows
if (HTTP.uri.indexOf("www.facebook.com") !== -1) {
   var name = "facebook-" + HTTP.uri;
   //packet capture options: capture 20 packets, up to 10 from the lookback
buffer
   var opts = {
       maxPackets: 20,
       maxPacketsLookback: 10
   };
   Flow.captureStart(name, opts);
}
```

Notes about capture functionality:

- The `Flow.captureStart()` function call requires a license with Triggered Packet Capture enabled.
- The packet capture trigger must have packet capture enabled. When configuring the trigger in the Web UI, select the Packet Capture checkbox under Advanced Options.
- Captured files are available in the Admin UI.
- Once the packet capture drive is full, no new captures will be recorded until the user deletes the files manually.
- Maximum file name string length is 256 characters. If the name exceeds 256 characters, it will be truncated and a warning message will be visible in the debug log, but the trigger will continue to execute.
- The capture file size is the lesser of `maxPackets` and `maxBytes`.
- Size of capture allocated to data in the lookback buffer is the lesser of `maxPacketsLookback` and `maxBytesLookback`.
- Each passed max* parameter will capture up to the next packet boundary.
- If the packet capture was already started on the current flow, the `Flow.captureStart()` calls result in a warning visible in the debug log, but the trigger will continue to execute.
- There is a maximum of 128 simultaneous packet captures in the system. If that limit is reached, subsequent calls to `Flow.captureStart()` will generate a warning visible in the debug log, but the trigger will continue to execute.
- Packet captures do not include the opening TCP SYN/ACK handshake.
- For decrypted flows, `Flow.capureStart()` will still write out the raw packet buffer with encrypted data.

captureStop(): Boolean (version 3.8.16174+)
Stops a packet capture that is in progress on the current flow.

getApplication(): String
Gets the currently associated application.

ipproto: String
The IP protocol associated with the flow, such as TCP or UDP.

ipver: String (version 3.8.16428+)
The IP version associated with the flow, such as IPv4 or IPv6.

isExpired: Boolean
Returns true if the flow expired at the time of the event.

l7proto:String
The L7 protocol associated with the flow. For known protocols, it is a string representing the protocol name (e.g., HTTP, DB, Memcache, or any L7 section heading in this document). For lesser-known protocols, it is a string with the format `ipproto:port` (e.g., `tcp:13724` or `udp:11258`). Not valid during the `TCP_OPEN` event.

setApplication(name: String**, [turnTiming:**Boolean**])**: void (version 3.8.16108+)
Associates L2-L4 metrics for a flow with the L4 component of the application specified by `name`. The `turnTiming` flag is set to `false` by default. If set to `true`, the ExtraHop system collects additional turn timing metrics for the flow. If this flag is omitted, no turn timing metrics are recorded for the application on the associated flow.

`Flow.setApplication(name)` is commonly used in a `FLOW_CLASSIFY` event to associate flows with applications where there is no corresponding L7 trigger event on which to call `Application(name).commit()` (e.g., proprietary protocols for which there is no ExtraHop analysis module). For flows that have L7 trigger events that support the `Application(name).commit()` method, that method collects a larger set of protocol metrics and is generally recommended.

A flow is associated with at most one application at a given instant. It is possible, however, to have a set of triggers call `Flow.setApplication(name)` at different instants and with different `name` arguments over the course of a given flow. This configuration results in metrics for a flow being dispersed across multiple applications and is generally not recommended.

Turn timing analysis analyzes L4 behavior in order to infer L7 processing times when the monitored protocol follows a client-request, server-response pattern and in which the client sends the first message. "Banner" protocols (where the server sends the first message) and protocols where data flows in both directions concurrently are not recommended for turn timing analysis.

store: Object
The flow store is designed to pass objects from request to response on the same flow. The `store` object is an instance of an empty JavaScript object. Objects can be attached to the store as properties by defining the property key and property value. For example:

```
Flow.store.myobject = "myvalue";
```

For events that occur on the same flow, you can use Flow.store instead of the session table to share information. For example:

```
/* request */
Flow.store.userAgent = HTTP.userAgent;
```

```
/* response */
var userAgent = Flow.store.userAgent;
```

Important: Flow store values persist across all requests and responses carried on that flow. When working with the flow store, it is a best practice to set the flow store variable to null when its value should not be conveyed to the next request or response. This practice has the added benefit of conserving flow store memory.

Most flow store triggers should have a structure similar to the following example:

```
if (event === 'DB_REQUEST') {
                if (DB.statement) {
                Flow.store.stmt = DB.statement;
} else {
                Flow.store.stmt = null;
}
}
else if (event === 'DB_RESPONSE') {
       var stmt = Flow.store.stmt;
       Flow.store.stmt = null;
       if (stmt) {
                // Do something with 'stmt';
                // e.g., commit a metric
       }
}
```

vlan: Number
The VLAN number associated with the flow. If no VLAN tag is present, this value is set to 0.

FLOW_TICK: Event

Once `FLOW_CLASSIFY` has fired, the `FLOW_TICK` event will fire on every subsequent turn. A turn represents a full cycle of a client transferring a response. For turns larger than 128 packets, `FLOW_TICK` will fire every 128 packets. You can configure this threshold in the Admin UI.

`FLOW_TICK` provides a means to periodically check for certain conditions on the flow, such as zero windows and Nagle delays, and then take an action, such as initiating a packet capture or sending a syslog message.

In addition to the properties and methods given for the `FLOW_CLASSIFY` event, the following properties become available during `FLOW_TICK` events and during the related `FLOW_TURN` event. These properties are not available during the `FLOW_CLASSIFY` event.

*Go to **Events List**.*

Flow.bytes1: Number
The number of L4 payload bytes transmitted by the device with the numerically lower IP address.

Flow.bytes2: Number
The number of L4 payload bytes transmitted by the device with the numerically higher IP address.

Flow.dscp1: Number

The last Differentiated Services Code Point (DSCP) value transmitted by the device with the numerically lower IP address.

Flow.dscp2: Number
The last Differentiated Services Code Point (DSCP) value transmitted by the device with the numerically higher IP address.

Flow.l2Bytes1: Number
The number of L2 bytes, including the ethernet headers, transmitted by the device with the numerically lower IP address.

Flow.l2Bytes2: Number
The number of L2 bytes, including the ethernet headers, transmitted by the device with the numerically higher IP address.

Flow.client.customDevices: Array
A list of custom devices that are acting as the clients in the flow.

Flow.client.dscp: Number
The last Differentiated Services Code Point (DSCP) value transmitted by the client in the flow.

Flow.client.dscpBytes: Array (version 4.1.4.24670)
An array of counts of the number of L2 bytes for a given Differentiated Services Code Point (DSCP) value transmitted by the client in the flow. A zero is returned for each entry which had no bytes of that DSCP value since the last FLOW_TICK.

Flow.client.dscpPkts: Array (version 4.1.4.24670)
An array of counts of the number of L2 packets for a given Differentiated Services Code Point (DSCP) value transmitted by the client in the flow. A zero is returned for each entry which had no packets of that DSCP value since the last FLOW_TICK.

Flow.client.l2Bytes: Number
The number of L2 bytes, including the ethernet headers, transmitted by the client in the flow.

Flow.client.nagleDelay: Number
The number of Nagle delays associated with the client in the flow.

Flow.client.payload: **Buffer**
The payload associated with the client in the flow.

Flow.client.pkts: Number
The number of packets transmitted by the client in the flow.

Flow.client.rcvWndThrottle: Number
The number of receive window throttles associated with the client in the flow.

Flow.client.rto: Number
The number of "RTO Out" when the device is acting as the client in the flow.

Flow.client.zeroWnd: Number
The number of zero windows associated with the client in the flow.

Flow.customDevices1: Number
A list of custom devices associated with the device with the numerically lower IP address.

Flow.customDevices2: Number
A list of custom devices associated with the device with the numerically higher IP address.

Flow.dscpBytes1: Array (version 4.1.4.24670)

An array of counts of the number of L2 bytes for a given Differentiated Services Code Point (DSCP) value transmitted by the device with the numerically lower IP address in the flow. A zero is returned for each entry which had no bytes of that DSCP value since the last FLOW_TICK.

Flow.dscpBytes2: Array (version 4.1.4.24670)
An array of counts of the number of L2 bytes for a given Differentiated Services Code Point (DSCP) value transmitted by the device with the numerically higher IP address in the flow. A zero is returned for each entry which had no bytes of that DSCP value since the last FLOW_TICK.

Flow.dscpPkts1: Array (version 4.1.4.24670)
An array of counts of the number of L2 packets for a given Differentiated Services Code Point (DSCP) value transmitted by the device with the numerically lower IP address in the flow. A zero is returned for each entry which had no packets of that DSCP value since the last FLOW_TICK.

Flow.dscpPkts2: Array (version 4.1.4.24670)
An array of counts of the number of L2 packets for a given Differentiated Services Code Point (DSCP) value transmitted by the device with the numerically higher IP address in the flow. A zero is returned for each entry which had no packets of that DSCP value since the last FLOW_TICK.

Flow.nagleDelay1: Number
The number of nagle delays associated with the device with numerically lower IP address.

Flow.nagleDelay2: Number
The number of nagle delays associated with the device with numerically higher IP address.

Flow.payload1: **Buffer**
The payload associated with the device with the numerically lower IP address.

Flow.payload2: **Buffer**
The payload associated with the device with the numerically higher IP address.

Flow.pkts1: Number
The number of packets transmitted by the device with the numerically lower IP address.

Flow.pkts2: Number
The number of packets transmitted by the device with the numerically higher IP address.

Flow.rcvWndThrottle1: Number
The number of times the advertised receive window of the device with the numerically lower IP address limits the throughput of the connection.

Flow.rcvWndThrottle2: Number
The number of times the advertised receive window of the device with the numerically higher IP address limits the throughput of the connection.

Flow.receiver.bytes: Number
The number of bytes transmitted by the receiver in the flow.

Flow.receiver.customDevices: Number
A list of custom devices that are acting as the receivers in the flow.

Flow.receiver.dscp: Number
The last Differentiated Services Code Point (DSCP) value transmitted by the receiver in the flow.

Flow.receiver.dscpBytes: Array (version 4.1.4.24670)
An array of counts of the number of L2 bytes for a given Differentiated Services Code Point (DSCP) value transmitted by the receiver in the flow. A zero is returned for each entry which had no bytes of that DSCP value since the last FLOW_TICK.

Flow.receiver.dscpPkts: Array (version 4.1.4.24670)

An array of counts of the number of L2 packets for a given Differentiated Services Code Point (DSCP) value transmitted by the receiver in the flow. A zero is returned for each entry which had no packets of that DSCP value since the last FLOW_TICK.

Flow.receiver.l2Bytes: Number
The number of L2 bytes, including the ethernet headers, transmitted by the receiver in the flow.

Flow.receiver.nagleDelay: Number
The number of nagle delays associated with the receiver in the flow.

Flow.receiver.payload: **Buffer**
The payload associated with the receiver in the flow. This is always null.

Flow.receiver.pkts: Number
The number of packets transmitted by the receiver in the flow.

Flow.receiver.rcvWndThrottle: Number
The number of receive window throttles associated with the receiver in the flow.

Flow.receiver.rto: Number
The number of "RTO Out" when the device is acting as the receiver in the flow.

Flow.receiver.zeroWnd: Number
The number of zero windows associated with the receiver in the flow.

Flow.roundTripTime: Number
The median round-trip time (RTT) for the duration of the event, expressed in milliseconds. May be NaN if there are no RTT samples.

Flow.rto1: Number
The number of RTOs associated with the device with numerically lower IP address.

Flow.rto2: Number
The number of RTOs associated with the device with numerically higher IP address.

Flow.sender.bytes: Number
The number of L4 payload bytes transmitted by the sender in the flow.

Flow.sender.customDevices: Number
A list of custom devices that are acting as the senders in the flow.

Flow.sender.dscp: Number
The last Differentiated Services Code Point (DSCP) value transmitted by the sender in the flow.

Flow.sender.dscpBytes: Array (version 4.1.4.24670)
An array of counts of the number of L2 bytes for a given Differentiated Services Code Point (DSCP) value transmitted by the sender in the flow. A zero is returned for each entry which had no bytes of that DSCP value since the last FLOW_TICK.

Flow.sender.dscpPkts: Array (version 4.1.4.24670)
An array of counts of the number of L2 packets for a given Differentiated Services Code Point (DSCP) value transmitted by the sender in the flow. A zero is returned for each entry which had no packets of that DSCP value since the last FLOW_TICK.

Flow.sender.l2Bytes: Number
The number of L2 bytes, including the ethernet headers, transmitted by the sender in the flow.

Flow.sender.nagleDelay: Number
The number of nagle delays associated with the sender in the flow.

Flow.sender.payload: **Buffer**
The payload associated with the sender in the flow.

Flow.sender.pkts: Number
The number of packets transmitted by the sender in the flow.

Flow.sender.rcvWndThrottle: Number
The number of receive window throttles associated with the sender in the flow.

Flow.sender.rto: Number
The number of "RTO Out" when the device is acting as the sender in the flow.

Flow.sender.zeroWnd: Number
The number of zero windows associated with the sender in the flow.

Flow.server.bytes: Number
The number of L4 payload bytes transmitted by the server in the flow.

Flow.server.customDevices: Number
A list of custom devices that are acting as the servers in the flow.

Flow.server.dscp: Number
The last Differentiated Services Code Point (DSCP) value transmitted by the server in the flow.

Flow.server.dscpBytes: Array (version 4.1.4.24670)
An array of counts of the number of L2 bytes for a given Differentiated Services Code Point (DSCP) value transmitted by the server in the flow. A zero is returned for each entry which had no bytes of that DSCP value since the last FLOW_TICK.

Flow.server.dscpPkts: Array (version 4.1.4.24670)
An array of counts of the number of L2 packets for a given Differentiated Services Code Point (DSCP) value transmitted by the server in the flow. A zero is returned for each entry which had no packets of that DSCP value since the last FLOW_TICK.

Flow.server.l2Bytes: Number
The number of L2 bytes transmitted by the server in the flow.

Flow.server.nagleDelay: Number
The number of nagle delays associated with the server in the flow.

Flow.server.payload: **Buffer**
The payload associated with the server in the flow.

Flow.server.pkts: Number
The number of packets transmitted by the server in the flow.

Flow.server.rcvWndThrottle: Number
The number of receive window throttles associated with the server in the flow.

Flow.server.rto: Number
The number of "RTO Out" when the device is acting as the server in the flow.

Flow.server.zeroWnd: Number
The number of zero windows associated with the server in the flow.

Flow.zeroWnd1: Number
The number of zero windows associated with the device with numerically lower IP address.

Flow.zeroWnd2: Number
The number of zero windows associated with the device with numerically higher IP address.

The following is an example of FLOW_TICK:

```
log("RTT " + Flow.roundTripTime);
Remote.Syslog.info(
  " eh_event=FLOW_TICK" +
  " ClientIP="+Flow.client.ipaddr+
  " ServerIP="+Flow.server.ipaddr+
  " ServerPort="+Flow.server.port+
  " ServerName="+Flow.server.device.dnsNames[0]+
  " RTT="+Flow.roundTripTime);
```

FLOW_TURN: Event
Fires on every TCP or UDP turn. A turn represents one full cycle of a client transferring request data followed by a server transferring a response.

In addition to the properties of `FLOW_TICK`, the event also exposes a Turn object.

*Go to **Events List**.*

TCP_CLOSE: Event
Fires when the TCP connection is shut down by being closed, expired or aborted.

*Go to **Events List**.*

TCP_OPEN: Event
Fires when the TCP connection is first fully established. Provides a hook for recording metrics using the following properties:

*Go to **Events List**.*

TCP.client.getOption(): Array
Returns an array of all TCP options on the client that have a kind number matching the passed in value.

TCP.client.hasECNEcho: Boolean
Returns true if the ECN flag is set on the client during the three-way handshake.

TCP.client.initSeqNum: Number
The initial sequence number sent from the client during the three-way handshake.

TCP.client.options: Array
An array of objects representing the TCP options of the client with a numerically lower IP address in the initial handshake packets. For more information, refer to **TCP Options** below.

TCP.client.wndSize: Number
The size of the TCP sliding window on the client negotiated during the three-way handshake.

TCP.hasECNEcho1: Boolean
Returns true if the ECN flag is set on the device with a numerically lower IP address during the three-way handshake.

TCP.hasECNEcho2: Boolean
Returns true if the ECN flag is set on the device with a numerically higher IP address during the three-way handshake.

TCP.initSeqNum1: Number
The initial sequence number of the device with a numerically lower IP address sent during the three-way handshake.

TCP.initSeqNum2: Number
The initial sequence number of the device with a numerically higher IP address sent during the three-way handshake.

TCP.options1: Array
An array of options representing the TCP options of the device with a numerically lower IP address in the initial handshake packets. For more information, refer to **TCP Options** below.

TCP.options2: Array
An array of options representing the TCP options of the device with a numerically higher IP address in the initial handshake packets. For more information, refer to **TCP Options** below.

TCP.server.getOption(): Array
Returns an array of all TCP options on the server that have a kind number matching the passed in value.

TCP.server.hasECNEcho: Boolean
Returns true if the ECN flag is set on the server during the three-way handshake.

TCP.server.initSeqNum: Number
The initial sequence sent from the server during the three-way handshake.

TCP.server.options: Array
An array of objects representing the TCP options of the server with a numerically higher IP address in the initial handshake packets. For more information, refer to **TCP Options** below.

TCP.wndSize1: Number
The size of the TCP sliding window of the device with a numerically lower IP address negotiated during the three-way handshake.

TCP.wndSize2: Number
The size of the TCP sliding window of the device with a numerically higher IP address negotiated during the three-way handshake.

TCP Options

All TCP Options objects have the following properties:

kind: Number
The TCP option kind number.

TCP Kind Numbers

Kind No.	Meaning
0	End of Option List
1	No-Operation
2	Maximum Segment Size
3	Window Scale
4	SACK Permitted
5	SACK
6	Echo (obsoleted by option 8)

Kind No.	Meaning
7	Echo Reply (obsoleted by option 8)
8	Timestamps
9	Partial Order Connection Permitted (obsolete)
10	Partial Order Service Profile (obsolete)
11	CC (obsolete)
12	CC.NEW (obsolete)
13	CC.ECHO (obsolete)
14	TCP Alternate Checksum Request (obsolete)
15	TCP Alternate Checksum Data (obsolete)
16	Skeeter
17	Bubba
18	Trailer Checksum Option
19	MD5 Signature Option (obsoleted by option 29)
20	SCPS Capabilities
21	Selective Negative Acknowledgements
22	Record Boundaries
23	Corruption experienced
24	SNAP
25	Unassigned (released 2000-12-18)
26	TCP Compression Filter
27	Quick-Start Response
28	User Timeout Option (also, other known authorized use)
29	TCP Authentication Option (TCP-AO)
30	Multipath TCP (MPTCP)
31	Reserved (known authorized used without proper IANA assignment)
32	Reserved (known authorized used without proper IANA assignment)
33	Reserved (known authorized used without proper IANA assignment)
34	TCP Fast Open Cookie
35-75	Reserved
76	Reserved (known authorized used without proper IANA assignment)
77	Reserved (known authorized used without proper IANA assignment)
78	Reserved (known authorized used without proper IANA assignment)
79-252	Reserved
253	RFC3692-style Experiment 1 (also improperly used for shipping products)

Kind No.	Meaning
254	RFC3692-style Experiment 2 (also improperly used for shipping products)

name: String
The name of the TCP option.

The following list contains the names of common TCP options and their specific properties:

Maximum Segment Size (name 'mss', option kind 2)

value: Number
The maximum segment size.

Window Scale (name 'wscale', kind 3)

value: Number
The window scale factor.

Selective Acknowledgement Permitted (name 'sack-permitted', kind 4)
No additional properties. Its presence indicates that the selective acknowledgment option was included in the SYN.

Timestamp (name 'timestamp', kind 8)

tsval: Number
The TSVal field for the option.

tsecr: Number
The TSecr field for the option.

Quickstart Response (name 'quickstart-rsp', kind 27)

rate-request: Number
The requested rate for transport, expressed in bytes per second.

ttl-diff: Number
The TTLDif.

qs-nonce: Number
The QS Nonce.

Akamai Address (name 'akamai-addr', kind 28)

value: IPAddr
The IP Address of the Akamai server.

User Timeout (name 'user-timeout', kind 28)

value: Number
The user timeout.

Authentication (name 'tcp-ao', kind 29)

keyId property: Number
The key id for the key in use.

rNextKeyId: Number
The key id for the "receive next" key id.

mac: Buffer
The message authentication code.

Multipath (name 'mptcp', kind 30)

value: Buffer

> **Note**: The Akamai Address and User Timeout options are differentiated by the length of the option.

The following is an example using TCP options:

```
if (TCP.client.options != null) {

    var optMSS = TCP.client.getOption(2)

    if (optMSS && (optMSS.value > 1460)) {
        Network.metricAddCount('large_mss', 1);
        Network.metricAddDetailCount('large_mss_by_client_ip',
                                     Flow.client.ipaddr + " " +
optMSS.value, 1);
    }

}
```

TCP_PAYLOAD: Event
Fires when the payload matches the criteria configured in the associated trigger.

*Go to **Events List**.*

UDP_PAYLOAD: Event
Fires when the payload matches the criteria configured in the associated trigger.

*Go to **Events List**.*

Deprecated

Flow.isClientAborted: Boolean (version 3.10.18355+)
Deprecated. Use `Flow.client.isAborted` instead.

Flow.isServerAborted: Boolean (version 3.10.18355+)
Deprecated. Use `Flow.server.isAborted` instead.

Flow.turnInfo: String (version 3.9.17624+)
Deprecated. Use the top-level Turn object with attributes for the turn.

See Also

- **Example: CIFS Trigger**
- **Example: Customer ID Header**
- **Example: Database Trigger**
- **Example: Parse Custom POS Messages with Universal Payload Analysis**
- **Example: Parse Syslog Over TCP with Universal Payload Analysis**
- **Example: SOAP Request**

FTP

(version 4.0.18766+)

The FTP class allows retrieval of metrics available during the `FTP_REQUEST` and `FTP_RESPONSE` events.

*Go to **Classes List**.*

Events

FTP_REQUEST: Event
Fires on every FTP request processed by the device.

*Go to **Events List**.*

FTP_RESPONSE: Event
Fires on every FTP response processed by the device.

*Go to **Events List**.*

Properties

args: String (`FTP_RESPONSE` only)
The arguments to the command.

cwd: String (`FTP_RESPONSE` only)
In the case of a user at /, when the client sends "CWD subdir":

- FTP.cwd will be / when method == "CWD".
- FTP.cwd will be /subdir for subsequent commands (rather than CWD becoming the changed to directory as part of the CWD response trigger).

Includes "..." at the beginning of the path in the event of a resync or the path is truncated. Includes "..." at the end of the path if the path is too long. Path truncates at 4096 characters.

error: string (`FTP_RESPONSE` only)
The detailed error message recorded by the ExtraHop system.

isReqAborted: Boolean
Returns true if the connection is closed before the FTP request was complete.

isRspAborted: Boolean (`FTP_RESPONSE` only)
Returns true if the connection is closed before the FTP response was complete.

method: String
The FTP method.

path: String (`FTP_RESPONSE` only)
The path for FTP commands. Includes "..." at the beginning of the path in the event of a resync or the path is truncated. Includes "..." at the end of the path if the path is too long. Path truncates at 4096 characters.

reqBytes: Number (`FTP_RESPONSE` only)
The number of L4 request bytes.

reqL2Bytes: Number (`FTP_RESPONSE` only)
The number of L2 request bytes.

reqPkts: Number (`FTP_RESPONSE` only)
The number of request packets.

reqRTO: Number (`FTP_RESPONSE` only)
The number of request RTOs.

roundTripTime: Number (`FTP_RESPONSE` only)
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number (`FTP_RESPONSE` only)
The number of L4 response bytes.

rspL2Bytes: Number (`FTP_RESPONSE` only)
The number of L2 response bytes.

rspPkts: Number (`FTP_RESPONSE` only)
The number of response packets.

rspRTO: Number (`FTP_RESPONSE` only)
The number of response RTOs.

statusCode: Number (`FTP_RESPONSE` only)
The FTP status code of the response.

FTP Status Codes

Code	Meaning
110	Restart marker replay.
120	Service ready in nnn minutes.
125	Data connection already open; transfer starting.
150	File status okay; about to open data connection.
202	Command not implemented, superfluous at this site.
211	System status, or system help reply.
212	Directory status.
213	File status.
214	Help message.
215	NAME system type.
220	Service ready for new user.
221	Service closing control connection.
225	Data connection open; no transfer in progress.
226	Closing data connection. Requested file action successful.
227	Entering Passive Mode.
228	Entering Long Passive Mode.
229	Entering Extended Passive Mode.
230	User logged in, proceed. Logged out if appropriate.
231	User logged out; service terminated.
232	Logout command noted, will complete when transfer done
250	Requested file action okay, completed.
257	"PATHNAME" created.

Code	Meaning
331	User name okay, need password.
332	Need account for login.
350	Requested file action pending further information.
421	Service not available, closing control connection.
425	Can't open data connection.
426	Connection closed; transfer aborted.
430	Invalid username or password.
434	Requested host unavailable.
450	Requested file action not taken.
451	Requested action aborted. Local error in processing.
452	Requested action not taken.
501	Syntax error in parameters or arguments.
502	Command not implemented.
503	Bad sequence of commands.
504	Command not implemented for that parameter.
530	Not logged in.
532	Need account for storing files.
550	Requested action not taken. File unavailable.
551	Requested action aborted. Page type unknown.
552	Requested file action aborted. Exceeded storage allocation.
553	Requested action not taken. File name not allowed.
631	Integrity protected reply.
632	Confidentiality and integrity protected reply.
633	Confidentiality protected reply.
10054	Connection reset by peer.
10060	Cannot connect to remote server.
10061	Cannot connect to remote server. The connection is active refused.
10066	Directory not empty.
10068	Too many users, server is full.

tprocess: Number (`FTP_RESPONSE` only)

The server processing time, expressed in milliseconds (equivalent to `rspTimeToFirstPayload` - `reqTimeToLastByte`). NaN on malformed and aborted responses or expired flows.

user: String

The user name, if available. In some cases, such as when login events are encrypted, the user name is not available.

HL7

(version 4.0.18766+)

The HL7 class allows retrieval of metrics available during the `HL7_REQUEST` and `HL7_RESPONSE` events.

*Go to **Classes List**.*

Events

HL7_REQUEST: Event
Fires on every HL7 request processed by the device.

*Go to **Events List**.*

HL7_RESPONSE: Event
Fires on every HL7 response processed by the device.

*Go to **Events List**.*

Properties

ackCode: String (`HL7_RESPONSE` only)
The two character acknowledgment code.

ackId: String (`HL7_RESPONSE` only)
The identifier for the message being acknowledged.

msgId: String
The unique identifier for this message.

msgType: String
The entire message type field, including the `msgId` subfield.

roundTripTime: Number (`HL7_RESPONSE` only)
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

segments: Array
An array of objects where each object is of type `(name: XYZ, fields: array of strings)`.

subfieldDelimiter: String
Supports non-standard field delimiters.

tprocess: Number (`HL7_RESPONSE` only)
The server processing time, expressed in milliseconds.

version: String
The version advertised in the MSH segment.

Note: The amount of buffered data is limited by the following capture option:
`("message_length_max": number)`

HTTP

The HTTP class allows retrieval of metrics available during the `HTTP_REQUEST` and `HTTP_RESPONSE` events.

*Go to **Classes List**.*

Events

HTTP_REQUEST: Event
Fires on every HTTP request processed by the device.

*Go to **Events List**.*

HTTP_RESPONSE: Event
Fires on every HTTP response processed by the device.

*Go to **Events List**.*

Methods

findHeaders(name:String): Array
Allows access to HTTP header values. The result is an array of header objects (with **name** and **value** properties) where the names match the prefix of the string passed to `findHeaders`. Refer to **Example: HTTP Header Object** for more information.

parseQuery(String): Object (version 3.10.17757+)
Function that accepts a query string and returns an object with names and values corresponding to those in the query string. Example:

```
var query = HTTP.parseQuery(HTTP.query);
debug("user id: " + query.userid);
```

Properties

age: Number
For `HTTP_REQUEST` events, the time from the first byte of the request until the last seen byte of the request. For `HTTP_RESPONSE` events, the time from the first byte of the request until the last seen byte of the response. The time is expressed in milliseconds. Returns a valid value on malformed and aborted requests. Returns NaN on expired requests and responses, or if the timing is invalid.

contentType: String
The value in the HTTP content-type header.

cookies: Array (version 3.10.17757+)
An array of objects representing cookies, containing properties corresponding to the content of each cookie. For example:

```
var cookies = HTTP.cookies,
    cookie,
    i;
for (i = 0; i < cookies.length; i++) {
    cookie = cookies[i];
    if (cookie.domain) {
        debug("domain: " + cookie.domain);
    }
}
```

headers: Object

An array-like object that allows access to HTTP header names and values. Access a specific header using one of these methods:

string property:

The name of the header, accessible in a dictionary-like fashion. For example:

```
var headers = HTTP.headers;
session = headers["X-Session-Id"];
accept = headers.accept;
```

numeric property:

Corresponds to the order in which the headers appear on the wire. The returned object has a name and a value property. Numeric properties are useful for iterating over all the headers and disambiguating headers with duplicate names. For example:

```
for (i = 0; i < headers.length; i++) {
    hdr = headers[i];
    debug("headers[" + i + "].name: " + hdr.name);
    debug("headers[" + i + "].value: " + hdr.value);
}
```

Note: Saving HTTP.headers to the Flow store does not save all of the individual header values. It is a best practice to save the individual header values to the Flow store. Refer to Appendix-A for details.

host: String

The value in the HTTP host header.

isDesync: Boolean

Returns true if the protocol parser became desynchronized due to missing packets.

isPipelined: Boolean

Returns true if the request is pipelined.

isReqAborted: Boolean

Returns true if the connection is closed before the HTTP request was complete.

isRspAborted: Boolean (`HTTP_RESPONSE` only)

Returns true if the connection is closed before the HTTP response was complete.

isRspChunked: Boolean (`HTTP_RESPONSE` only)

Returns true if the response is chunked.

isRspCompressed: Boolean

Returns true if the response is compressed.

method: String

The HTTP method such as POST and GET.

origin: IPAddress | String

The value in X-Forwarded-For or true-client-ip header.

path: String

The path portion of the URI: `/path/`.

payload: **Buffer** (version 4.0.21449+)
The *n* first bytes of HTTP request or response payload (data past the headers), where *n* is the number specified in the trigger. When configuring the trigger in the Web UI, select the `HTTP_RESPONSE` or `HTTP_REQUEST` event, click **Show advanced options**, and enter the number of payload bytes to buffer. If the payload was compressed, the decompressed content is returned.

Example of how to use HTTP payload analysis:

```
/* Extract the user name based on a pattern "user=*&" from payload of a
login URI that
has "auth/login" as a URI substring. */

if (HTTP.payload && /auth\/login/i.test(HTTP.uri)) {
    var user = /user=(.*)\&/i.exec(HTTP.payload);
    if (user != null) {
        Flow.store.user = user[1];
    }
}
```

Note: If two HTTP payload buffering triggers are assigned to the same device, the higher value is used and the value of HTTP.payload will be the same for both triggers.

query: String (`HTTP_REQUEST` only)
The query string portion of the URI: `query=string`. This typically follows the URL and is separated from it by a question mark. Multiple query strings are separated by an ampersand (&) or semi-colon (;) delimiter.

referer: String
The value in the HTTP referrer header.

reqBytes: Number (`HTTP_RESPONSE` only)
The number of L4 request bytes.

reqL2Bytes: Number (`HTTP_RESPONSE` only)
The number of request L2 bytes.

reqPkts: Number (`HTTP_RESPONSE` only)
The number of request packets.

reqRTO: Number (`HTTP_RESPONSE` only)
The number of request RTOs.

reqSize: Number
The size of the request payload, expressed in bytes. Does not count the HTTP header and only counts the number of bytes in the body of the request.

reqTimeToLastByte: Number
Time from the first byte of the request until the last byte of the request, expressed in milliseconds. Return NaN on expired requests and responses.

roundTripTime: Number (`HTTP_RESPONSE` only)
The median TCP round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number (`HTTP_RESPONSE` only)
The number of L4 response bytes.

rspL2Bytes: Number (`HTTP_RESPONSE` only)
The number of response L2 bytes.

rspPkts: Number (`HTTP_RESPONSE` only)
The number of response packets.

rspRTO: Number (`HTTP_RESPONSE` only)
The number of response RTOs.

rspSize: Number (`HTTP_RESPONSE` only)
The size of the response payload, expressed in bytes. Does not count the HTTP header and only counts the number of bytes in the body of the request.

rspTimeToFirstHeader: Number (`HTTP_RESPONSE` only)
The time from the first byte of the request until the status line that precedes the response headers, expressed in milliseconds. Returns NaN on malformed and aborted responses or expired flows.

rspTimeToFirstPayload: Number (`HTTP_RESPONSE` only)
The time from the first byte of the request until the first payload byte of the response, expressed in milliseconds. Returns zero value when the response does not contain payload. Returns NaN on malformed and aborted responses or expired flows.

rspTimeToLastByte: Number (`HTTP_RESPONSE` only)
The time from the first byte of the request until the last byte of the response, expressed in milliseconds. Returns NaN on malformed and aborted responses or expired flows.

rspVersion: String (`HTTP_RESPONSE` only)
The HTTP version.

statusCode: Number (`HTTP_RESPONSE` only)
The HTTP status code of the response.

> **Note**: A status code of 0 is returned when there is not a valid `HTTP_RESPONSE` returned.

title: String
The value in the title element of the HTM content, if present.

thinkTime: Number (version 4.1.22976+)
The time elapsed between the server having transferred the response to the client and the client transferring a new request to the server, expressed in milliseconds. Will return NaN if there is no valid measurement.

tprocess: Number (`HTTP_RESPONSE` only)
The server processing time, expressed in milliseconds (equivalent to `rspTimeToFirstPayload - reqTimeToLastByte`). Returns NaN on malformed and aborted responses or expired flows.

uri: String
The URI without a query string: `f.q.d.n/path/`.

userAgent: String (`HTTP_REQUEST` only)
The value in the HTTP user-agent header.

Deprecated

payloadText: String

Deprecated. Use `payload` instead.

See Also

- **Example: Customer ID Header**
- **Example: SOAP Request**
- **Example: HTTP Header Object**
- **Example: Session Table**
- **Example: Trigger-Based Application Definition**

IBMMQ

The IBMMQ class allows retrieval of metrics available during the `IBMMQ_REQUEST` and `IBMMQ_RESPONSE` events.

*Go to **Classes List**.*

Note: The IBMMQ protocol supports EBCDIC encoding.

Events

IBMMQ_REQUEST: Event
Fires on every IBMMQ request processed by the device.

*Go to **Events List**.*

IBMMQ_RESPONSE: Event
Fires on every IBMMQ response processed by the device.

*Go to **Events List**.*

Properties

channel: String
The communication channel name.

correlationId: String (version 3.8.15928+)
The IBMMQ correlation ID.

error: String
The error string corresponding to the error code on the wire.

messageId: String (version 3.8.15928+)
The IBMMQ message ID.

method: String
The wire protocol request/response method name.

Note: Wireshark users may notice some method names used by ExtraHop differ from those used by Wireshark. These are:

ASYNC_MSG_V7	ASYNC_MESSAGE
MQCLOSEv7	SOCKET_ACTION
MQGETv7	REQUEST_MSGS
MQGETv7_REPLY	NOTIFICATION

msgFormat: String
The message format.

msgSize: Number
The size of the IBMMQ message, expressed in bytes.

objectHandle: String
The handle of the object placed or retrieved from the queue.

payload: **Buffer**(version 4.0.21257+)

For MQPUT, MQPUT1, MQGET_REPLY, ASYNC_MSG_V7, and MESSAGE_DATA messages, the payload is set to an instance of the **Buffer** class and could be converted to string using toString() or formatted using unpack commands. Large queue messages (those greater than approximately 32K) may be broken into more than one segment and, in those cases, a trigger will fire for each segment but only the first segment will have a non-null payload.

For all other messages, it is null.

pcfError: String
The error string corresponding to the error code on the wire for the PCF channel.

pcfMethod: String
The wire protocol request/response method name for the PCF channel.

pcfWarning: String
The warning string corresponding to the warning string on the wire for the PCF channel.

queue: String
The local queue name or null if there was no MQOPEN, MQOPEN_REPLY, MQSP1(Open), or MQSP1_REPLY seen.

queueMgr: String
The local queue manager or null if there was no INITIAL_DATA seen at the start of the connection.

reqBytes: Number
The number of application-level request bytes.

reqL2Bytes: Number
The number of request L2 bytes.

reqPkts: Number
The number of request packets.

reqRTO: Number
The number of request RTOs.

resolvedQueue: String
The resolved queue name from MQGET_REPLY, MQPUT_REPLY, or MQPUT1_REPLY. For a remote queue, this will be different from IBMMQ.queue.

resolvedQueueMgr: String
The resolved queue manager from MQGET_REPLY, MQPUT_REPLY, or MQPUT1_REPLY. For a remote queue, this will be different from IBMMQ.queueMgr.

rfh: Array of Strings (version 4.1.4.24670)
The strings found in the optional RFH header. If there is no RFH header or no strings, the array will be empty.

roundTripTime: Number
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number
The number of application-level response bytes.

rspL2Bytes: Number
The number of response L2 bytes.

rspPkts: Number
The number of request packets.

rspRTO: Number

The number of response RTOs.

warning: String
The warning string corresponding to the warning string on the wire.

ICA

The ICA class allows retrieval of metrics available during the `ICA_OPEN`, `ICA_AUTH`, `ICA_TICK`, and `ICA_CLOSE` events.

*Go to **Classes List**.*

Events

ICA_AUTH: Event
Fires when the ICA authentication is complete.

*Go to **Events List**.*

ICA_CLOSE: Event
Fires when the ICA session is torn down.

*Go to **Events List**.*

ICA_OPEN: Event
Fires right after the ICA application first loads.

*Go to **Events List**.*

ICA_TICK: Event
Fires periodically while the user interacts with the ICA application.

Once `ICA_LOAD` has fired at least once, the `ICA_TICK` event will be fired when either **networkLatency** or **clientLatency** is reported.

networkLatency is reported when a specific ICA packet from the client contains latency information. The latency in that packet is reported as **networkLatency**.

clientLatency is reported when a packet from the client on the EUEM channel reports the result of a single ICA round-trip measurement. The value of that measurement is reported as **clientLatency**.

> **Note**: **clientLatency** is only available via the API. It is not supported in the UI.

*Go to **Events List**.*

Properties

application: String
The name of the application that is being launched.

authDomain: String (version 3.10.20084+)
The Windows authentication domain to which the user belongs.

channels: Array (`ICA_TICK` only)
An array of objects containing information about virtual channels seen since the last tick event. Each object has the following properties:

name: String
The name of the virtual channel.

description: String
The friendly description of the channel name.

clientBytes: Integer
The number of bytes sent by the client for that channel.

serverBytes: Integer
The number of bytes sent by the server for the channel.

client: String
The name of the client machine. This is a name that is advertised by the ICA client and is usually the hostname of the client machine.

clientBytes: Number (`ICA_TICK` and `ICA_CLOSE` only)
The number of application level client bytes.

clientCGPMsgCount: Number (`ICA_TICK` only)
The number of client CGP messages since the last tick event.

clientLatency: Number (`ICA_TICK` only)
The current client-advertised latency in milliseconds.

clientL2Bytes: Number (`ICA_TICK` and `ICA_CLOSE` only)
The number of L2 client bytes.

clientMsgCount: Number (`ICA_TICK` only)
The number of client messages since the last tick event.

clientPkts: Number (`ICA_TICK` and `ICA_CLOSE` only)
The number of client packets.

clientRTO: Number (`ICA_TICK` and `ICA_CLOSE` only)
The number of client RTOs.

clientType: String
The user-agent equivalent to ICA. This is the type of the ICA client.

frameCutDuration: Number (`ICA_TICK` only)
Frame cut duration, as reported by EUEM beacon.

frameSendDuration: Number (`ICA_TICK` only)
The frame send duration, as reported by EUEM beacon.

host: String (version 3.10.20084+)
The host name of the Citrix server.

isAborted: Boolean (`ICA_CLOSE` only)
Returns true if the application failed to launch successfully.

isCleanShutdown: Boolean (`ICA_CLOSE` only)
Returns true if the application shut down cleanly.

isEncrypted: Boolean
Returns true if the application is encrypted using RC5 encryption.

isSharedSession: Boolean
Returns true if the application is launched over an existing connection.

launchParams: String (version 3.9.16980+)
Returns a string that represents the parameters.

loadTime: Number
The load time of the given application, expressed in milliseconds.

> **Note**: The load time is recorded only for the initial application load. The ExtraHop system does not measure load time for applications launched over existing sessions and instead

reports the initial load time on subsequent application loads. Use `ICA.isSharedSession` to distinguish between initial and subsequent application loads.

loginTime: Number (`ICA_OPEN`, `ICA_TICK`, and `ICA_CLOSE` only)
The user login time, expressed in milliseconds.

Note: The login time is recorded only for the initial application load. The ExtraHop system does not measure login time for applications launched over existing sessions and instead reports the initial login time on subsequent application loads. Use `ICA.isSharedSession` to distinguish between initial and subsequent application loads.

networkLatency: Number (`ICA_TICK` only)
The network latency as reported by EUEM beacon.

roundTripTime: Number (`ICA_TICK` and `ICA_CLOSE` only)
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

serverBytes: Number (`ICA_TICK` and `ICA_CLOSE` only)
The number of application-level server bytes.

serverCGPMsgCount: Number (`ICA_TICK` only)
The number of CGP server messages since the last tick event.

serverL2Bytes: Number (`ICA_TICK` and `ICA_CLOSE` only)
The number of L2 server bytes.

serverMsgCount: Number (`ICA_TICK` only)
The number of server messages since the last tick event.

serverPkts: Number (`ICA_TICK` and `ICA_CLOSE` only)
The number of server packets.

serverRTO: Number (`ICA_TICK` and `ICA_CLOSE` only)
The number of server RTOs.

user: String
The name of the user, if available.

Deprecated

authTicket: String (version 3.7.14867+)
Use **user** instead.

ICMP

The ICMP class allows retrieval of metrics available during the `ICMP_MESSAGE` event.

*Go to **Classes List**.*

Events

ICMP_MESSAGE: Event
Fires on every ICMP message processed by the device.

*Go to **Events List**.*

Properties

gwAddr: **IPAddress**
For a Redirect message, the address of the gateway to which traffic for the network specified in the internet destination network field of the original datagram's data should be sent. Returns null for all other messages.

Message	ICMPv4 Type	ICMPv6 Type
Redirect Message	5	n/a

hopLimit: Number
The ICMP packet time to live or hop count.

isError: Boolean
Returns true for messages types in the following table.

Message	ICMPv4 Type	ICMPv6 Type
Destination Unreachable	3	1
Redirect	5	n/a
Source Quench	4	n/a
Time Exceeded	11	3
Parameter Problem	12	4
Packet Too Big	n/a	2

isQuery: Boolean
Returns true for messages types in the following table.

Message	ICMPv4 Type	ICMPv6 Type
Echo Request	8	128
Information Request	15	n/a
Timestamp request	13	n/a
Address Mask Request	17	n/a
Router Discovery	10	151
Multicast Listener Query	n/a	130
Router Solicitation (NDP)	n/a	133
Neighbor Solicitation	n/a	135

Message	ICMPv4 Type	ICMPv6 Type
ICMP Node Information Query	n/a	139
Inverse Neighbor Discovery Solicitation	n/a	141
Home Agent Address Discovery Solicitation	n/a	144
Mobile Prefix Solicitation	n/a	146
Certification Path Solicitation	n/a	148

isReply: Boolean

Returns true for message types in the following table.

Message	ICMPv4 Type	ICMPv6 Type
Echo Reply	0	129
Information Reply	16	n/a
Timestamp Reply	14	n/a
Address Mask Reply	18	n/a
Multicast Listener Done	n/a	132
Multicast Listener Report	n/a	131
Router Advertisement (NDP)	n/a	134
Neighbor Advertisement	n/a	136
ICMP Node Information Response	n/a	140
Inverse Neighbor Discovery Advertisement	n/a	142
Home Agent Address Discovery Reply Message	n/a	145
Mobile Prefix Advertisement	n/a	147
Certification Path Advertisement	n/a	149

msg: **Buffer**

A Buffer containing up to `message_length_max` bytes of the ICMP message. The `message_length_max` option is configured in the ICMP profile in the running config.

The following running config example changes the ICMP message_length_max from its default of 4096 bytes to 1234 bytes:

```
"capture": {
    "app_proto": {
       "ICMP": {
            "message_length_max": 1234
        }
    }
}
```

msgCode: Number

The ICMP message code.

msgID: Number

The ICMP message identifier for Echo Request, Echo Reply, Timestamp Request, Timestamp Reply, Information Request, and Information Reply messages. Returns null for all other message types.

ICMP Message ID

Message	ICMPv4 Type	ICMPv6 Type
Echo Request	8	128
Echo Reply	0	129
Timestamp Request	13	n/a
Timestamp Reply	14	n/a
Information Request	15	n/a
Information Reply	16	n/a

msgLength: Number
The length of the ICMP message, expressed in bytes.

msgText: String
The descriptive text for the message (e.g., `echo request` or `port unreachable`).

msgType: Number
The ICMP message type.

ICMPv4

Type	Message
0	Echo Reply
1 and 2	*Reserved*
3	Destination Unreachable
4	Source Quench
5	Redirect Message
6	Alternate Host Address (deprecated)
7	*Reserved*
8	Echo Request
9	Router Advertisement
10	Router Solicitation
11	Time Exceeded
12	Parameter Problem: Bad IP header
13	Timestamp
14	Timestamp Reply
15	Information Request (deprecated)
16	Information Reply (deprecated)
17	Address Mask Request (deprecated)
18	Address Mask Reply (deprecated)

Type	Message
19	*Reserved*
20-29	*Reserved*
30	Traceroute (deprecated)
31	Datagram Conversion Error (deprecated)
32	Mobile Host Redirect (deprecated)
33	Where Are You (deprecated)
34	Here I Am (deprecated)
35	Mobile Registration Request (deprecated)
36	Mobile Registration Reply (deprecated)
37	Domain Name Request (deprecated)
38	Domain Name Reply (deprecated)
39	Simple Key-Management for Internet Protocol (deprecated)
40	Photuris (deprecated)
41	ICMP experimental
42-255	*Reserved*

ICMPv6

Type	Message
1	Destination Unreachable
2	Packet Too Big
3	Time Exceeded
4	Parameter Problem
100	Private Experimentation
101	Private Experimentation
127	Reserved for expansion of ICMPv6 error messages
128	Echo Request
129	Echo Reply
130	Multicast Listener Query
131	Multicast Listener Report
132	Multicast Listener Done
133	Router Solicitation
134	Router Advertisement
135	Neighbor Solicitation
136	Neighbor Advertisement
137	Redirect Message

Type	Message
138	Router Renumbering
139	ICMP Node Information Query
140	ICMP Node Information Response
141	Inverse Neighbor Discovery Solicitation Message
142	Inverse Neighbor Discovery Advertisement Message
143	Multicast Listener Discovery (MLDv2) reports
144	Home Agent Address Discovery Request Message
145	Home Agent Address Discovery Reply Message
146	Mobile Prefix Solicitation
147	Mobile Prefix Advertisement
148	Certification Path Solicitation
149	Certification Path Advertisement
151	Multicast Router Advertisement
152	Multicast Router Solicitation
153	Multicast Router Termination
155	RPL Control Message
200	Private Experimentation
201	Private Experimentation
255	Reserved for expansion of ICMPv6 informational messages

nextHopMTU: Number

For an ICMPv4 Destination Unreachable or an ICMPv6 Packet Too Big message, the maximum transmission unit of the next-hop link. Returns null for all other messages.

Message	ICMPv4 Type	ICMPv6 Type
Destination Unreachable	3	n/a
Packet Too Big	n/a	2

pointer: Number

For a Parameter Problem message, the octet of the original datagram's header where the error was detected. Returns null for all other messages.

Message	ICMPv4 Type	ICMPv6 Type
Parameter Problem	12	4

seqNum: Number

The ICMP sequence number for Echo Request, Echo Reply, Timestamp Request, Timestamp Reply, Information Request, and Information Reply messages. Null is returned for all other messages.

version: Number

The ICMP version. Can be either 4 or 6.

IPAddress

The IPAddress class allows setting and retrieval of IP address attributes. IPAddress is the type of IP address properties available on the Flow class.

*Go to **Classes List**.*

Methods

IPAddress(**ip**:String | Number, [**mask**:Number])
Constructor for the IPAddress class that takes two parameters:

ip: String
The IP address string in CIDR format.

mask: Number
The subnet mask in a numerical format, representing the number of leftmost '1' bits in the mask (optional).

Instance Methods

mask(**mask**:Number): IPAddress
Sets the subnet mask of the IPAddress object. Takes one parameter:

mask: Number
The subnet mask in a numerical format, representing the number of leftmost '1' bits in the mask (optional).

toString(): String
Converts the IPAddress object to a printable string.

Properties

hostNames: Array of Strings
An array of hostnames associated with the IPAddress.

isBroadcast: Boolean
Returns true if the IP address is a broadcast address.

isLinkLocal: Boolean
Returns true if the IP address is a link local address (169.254.0.0/16).

isMulticast: Boolean
Returns true if the IP address is a multicast address.

isRFC1918: Boolean
Returns true if the IP address belongs to one of the RFC1918 private IP ranges (10.0.0.0/8, 172.16.0.0, 192.168.0.0/16). Always returns false for IPv6 addresses.

isV4: Boolean
Returns true if the IP address is an IPv4 address.

isV6: Boolean
Returns true if the IP address is an IPv6 address.

LDAP

The LDAP class allows retrieval of metrics available during the `LDAP_REQUEST` and `LDAP_RESPONSE` events.

*Go to **Classes List**.*

Events

LDAP_REQUEST: Event
Fires on every LDAP request processed by the device.

*Go to **Events List**.*

LDAP_RESPONSE: Event
Fires on every LDAP response processed by the device.

*Go to **Events List**.*

Properties

bindDN: String (`LDAP_REQUEST` only, version 3.8.16379+)
The bind DN of the LDAP request.

dn: String
The LDAP distinguished name (DN).

error: String (`LDAP_RESPONSE` only)
The LDAP short error string as defined in the protocol (e.g., `noSuchObject`).

Result Code	Result String
1	operationsError
2	protocolError
3	timeLimitExceeded
4	sizeLimitExceeded
7	authMethodNotSupported
8	strongerAuthRequired
11	adminLimitExceeded
12	unavailableCriticalExtension
13	confidentialityRequired
16	noSuchAttribute
17	undefinedAttributeType
18	inappropriateMatching
19	constraintViolation
20	attributeOrValueExists
21	invalidAttributeSyntax
32	NoSuchObject
33	aliasProblem
34	invalidDNSSyntax

Result Code	Result String
36	aliasDeferencingProblem
48	inappropriateAuthentication
49	invalidCredentials
50	insufficientAccessRights
51	busy
52	unavailable
53	unwillingToPerform
54	loopDetect
64	namingViolation
65	objectClassViolation
66	notAllowedOnNonLeaf
67	notAllowedOnRDN
68	entryAlreadyExists
69	objectClassModsProhibited
71	affectsMultipleDSAs
80	other

errorDetail: String (`LDAP_RESPONSE` only)
The LDAP error detail, when available for that error type (e.g., `protocolError : historical procotol version requested, use LDAPv3 instead`).

method: String
The LDAP method.

msgSize: Number
The size of the LDAP message, expressed in bytes.

reqBytes: Number (version 3.9.17182+)
The number of request bytes.

reqL2Bytes: Number (version 3.9.17182+)
The number of request L2 bytes.

reqPkts: Number (version 3.9.17182+)
The number of request packets.

reqRTO: Number (version 3.9.17182+)
The number of request RTOs.

roundTripTime: Number (version 3.9.17182+)
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number (version 3.9.17182+)
The number of response bytes.

rspL2Bytes: Number (version 3.9.17182+)
The number of response L2 bytes.

rspPkts: Number (version 3.9.17182+)
The number of response packets.

rspRTO: Number (version 3.9.17182+)
The number of response RTOs.

saslMechanism: String (version 3.10.19298+)
The string that defines the SASL mechanism to identify and authenticate a user to a server.

searchAttributes: Array (`LDAP_REQUEST` only, version 4.0.19468+)
The attributes to return from objects that match the filter criteria.

searchFilter: String(`LDAP_REQUEST` only, version 4.0.19468+)
The mechanism to allow certain entries in the subtree and exclude others.

searchScope: String (`LDAP_REQUEST` only, version 4.0.19468+)
The depth of a search within the search base.

tprocess: Number (`LDAP_RESPONSE` only)
The server processing time, expressed in milliseconds. Available for the following:

- BindRequest
- SearchRequest
- ModifyRequest
- AddRequest
- DelRequest
- ModifyDNRequest
- CompareRequest
- ExtendedRequest

LLDP

(version 4.0.19680+)

The LLDP class allows retrieval of metrics available during the `LLDP_FRAME` events.

*Go to **Classes List**.*

Events

LLDP_FRAME: Event
Fires on every LLDP frame processed by the device.

*Go to **Events List**.*

Properties

chassisId: **Buffer**
The chassis ID, obtained from the chassisId data field, or type-length-value (TLV).

chassisIdSubtype: Number
The chassis ID subtype, obtained from the chassisID TLV.

destination: String
The destination MAC address.

optTLVs: Array
An array containing the optional TLVs. Each TLV is an object with the following properties:

customSubtype: Number
The subtype of an organizationally specific TLV.

isCustom: Boolean
Returns true if the object is an organizationally specific TLV.

oui: Integer
The organizationally unique identifier for organizationally specific TLVs.

type: Number
The type of TLV

value: String
The value of the TLV.

portId: **Buffer**
The port ID, obtained from the portId TLV.

portIdSubtype: Number
The port ID subtype, obtained from the portId TLV.

source: **Device**
The device sending the LLDP frame.

ttl: Number
The time to live, expressed in seconds. This is the length of time during which the information in this frame is valid, starting with when the information is received.

Memcache

The Memcache class allows retrieval of metrics available during the MEMCACHE_REQUEST and MEMCACHE_RESPONSE events.

*Go to **Classes List**.*

Events

MEMCACHE_REQUEST: Event
Fires on every memcache request processed by the device.

*Go to **Events List**.*

MEMCACHE_RESPONSE: Event
Fires on every memcache response processed by the device.

*Go to **Events List**.*

Properties

accessTime: Number (MEMCACHE_RESPONSE only)
The access time, expressed in milliseconds. Available only if the first key that was requested produced a hit.

error: String (MEMCACHE_RESPONSE only)
The detailed error message recorded by the ExtraHop system.

hits: Array (MEMCACHE_RESPONSE only)
An array of objects with the following properties:

key: String | Null
The Memcache key for which this was a hit, if available.

size: Number
The size of the value returned for the key, expressed in bytes.

isBinaryProtocol: Boolean
Returns true if the request/response corresponds to the binary version of the memcache protocol.

isNoReply: Boolean (MEMCACHE_REQUEST only)
Returns true if the request has the "noreply" keyword and therefore should never receive a response (text protocol only).

isRspImplicit: Boolean (MEMCACHE_RESPONSE only)
Returns true if the response was implied by a subsequent response from the server (binary protocol only).

method: String
The Memcache method (as recorded in ExtraHop metrics).

misses: Array (MEMCACHE_RESPONSE only)
An array of objects with the following property:

key: String | Null
The Memcache key for which this was a miss, if available.

reqBytes: Number
The number of application-level request bytes.

reqKeys: Array
An array containing the Memcache key strings sent with the request.

reqL2Bytes: Number
The number of request L2 bytes.

reqPkts: Number
The number of request packets.

reqRTO: Number (`MEMCACHE_REQUEST` only)
The number of request RTOs.

reqSize: Number
The size of the value in the request (if any), expressed in bytes. Note that methods such as `set` include values, while `get` and `delete` do not.

roundTripTime: Number
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number
The number of application-level response bytes.

rspL2Bytes: Number
The number of response L2 bytes.

rspPkts: Number
The number of response packets.

rspRTO: Number (`MEMCACHE_RESPONSE` only)
The number of response RTOs.

statusCode: String (`MEMCACHE_RESPONSE` only)
The Memcache status code. For the binary protocol, ExtraHop metrics prepend the method to status codes other than NO_ERROR, but the statusCode property does not. Refer to the examples for code that matches the behavior of ExtraHop metrics.

vbucket: Number
The Memcache vbucket, if available (binary protocol only).

See Also

- **Example: Memcache Hits and Misses**
- **Example: Memcache Key Parsing**

MetricCycle

The MetricCycle class represents an interval where stats were published. It is valid on the following events:

- `METRIC_CYCLE_BEGIN`
- `METRIC_CYCLE_END`
- `METRIC_RECORD_COMMIT`

*Go to **Classes List**.*

Events

METRIC_CYCLE_BEGIN: Event
Fires when a metric interval begins.

*Go to **Events List**.*

METRIC_CYCLE_END: Event
Fires when a metric interval ends.

*Go to **Events List**.*

Properties

id: String
A string representing the metric cycle. Possible values are:

- 30sec
- 5min
- 1hr
- 24hr

interval: Object
An object containing **from** and **until** properties, expressed in milliseconds since the epoch.

store: Object
An object that retains information across all the `METRIC_RECORD_COMMIT` events that occur during a metric cycle, that is, from the `METRIC_CYCLE_BEGIN` event to the `METRIC_CYCLE_END` event. This object is analogous to Flow.store in capture. MetricCycle.store is shared among triggers for `METRIC_*` events. It is cleared at the end of a metric cycle.

See Also

- **Example: Use the Metric Cycle Store**

MetricRecord

The MetricRecord class allows access to the current set of metrics in `METRIC_RECORD_COMMIT`.

*Go to **Classes List**.*

Events

METRIC_RECORD_COMMIT: Event
Fires when a metric record is committed to the datastore. Provides access to the object, metric type, cycle, metric fields (for example, for `extrahop.device.http.server`, properties include `req`, `rsp`, `rsp_error`, `status_code`, etc.).

*Go to **Events List**.*

Properties

fields: Object
An object containing metric values. The properties are the field names and the values can be numbers, **Topnset**, **Dataset** or **Sampleset**.

id: String
The metric type. For example, `extrahop.device.http.server`.

object: Object
The object the metric applies to. For device, application, or VLAN metrics, this property will contain a Device, Application, or VLAN instance, respectively. For capture metrics (e.g., `extrahop.capture.net`), the property will contain the global Network class.

Time: Number
The time that the metric record will be published with.

See Also

- **Example: Create a Custom Trouble Group**
- **Example: Topnset Key Matching**
- **Example: Use the Metric Cycle Store**

MongoDB

(version 4.0.13997+)

The MongoDB class allows retrieval of metrics available during the `MONGODB_REQUEST` and `MONGODB_RESPONSE` events.

*Go to **Classes List**.*

Events

MONGODB_REQUEST: Event
Fires on every MongoDB request processed by the device.

*Go to **Events List**.*

MONGODB_RESPONSE: Event
Fires on every MongoDB response processed by the device.

*Go to **Events List**.*

Properties

collection: String
The name of the database collection specified in the current request.

database: String
The MongoDB database instance. In some cases, such as when login events are encrypted, the database name is not available.

error: String (`MONGODB_RESPONSE` only)
The detailed error message recorded by the ExtraHop system.

isReqAborted: Boolean
Returns true if the connection is closed before the MongoDB request was complete.

isReqTruncated: Boolean
Returns true if the request document(s) size is greater than the maximum payload document size.

isRspAborted: Boolean (`MONGODB_RESPONSE` only)
Returns true if the connection is closed before the MongoDB response was complete.

method: String
The MongoDB database method (appears under **Methods** in the user interface).

opcode: String
The MongoDB operational code on the wire protocol, which may differ from the MongoDB method used.

reqBytes: Number
The number of application-level request bytes.

reqL2Bytes: Number
The number of request L2 bytes.

reqPkts: Number
The number of request packets.

reqRTO: Number
The number of request RTOs.

reqSize: Number
The size of the request record at L7, expressed in bytes.

reqTimeToLastByte: Number
The time from the first byte of the request until the last byte of the request, expressed in milliseconds.

request: Array
An array of JS objects parsed from MongoDB request payload documents. Total document size is limited to 4K.

If BSON documents are truncated, **isReqTruncated** flag is set. Truncated values are represented as follows:

- Primitive string values like code, code with scope, and binary data are partially extracted.
- Objects and Arrays are partially extracted.
- All other primitive values like Numbers, Dates, RegExp, etc., are substituted with null.

If no documents are included in the request, an empty array is returned.

roundTripTime: Number
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number
The number of application-level response bytes.

rspL2Bytes: Number
The number of response L2 bytes.

rspPkts: Number
The number of response packets.

rspRTO: Number
The number of response RTOs.

rspSize: Number (`MONGODB_RESPONSE` only)
The size of the response record at L7, expressed in bytes.

rspTimeToFirstByte: Number (`MONGODB_RESPONSE` only)
The time from the first byte of the request until the first byte of the response, expressed in milliseconds.

rspTimeToLastByte: Number (`MONGODB_RESPONSE` only)
The time from the first byte of the request until the last by of the response, expressed in milliseconds.

tprocess: Number (`MONGODB_RESPONSE` only)
The time to process the request, expressed in milliseconds (equivalent to `rspTimeToFirstByte - reqTimeToLastByte`).

user: String
The user name, if available. In some cases, such as when login events are encrypted, the user name is not available.

Network

The network class allows adding custom metrics at the global level.

*Go to **Classes List**.*

Use the following functions to record custom metrics associated with networks. Refer to the ExtraHop Datatypes section for an overview of the data types.

- **metricAddCount**(metric_name:String, count:Number): void
- **metricAddDataset**(metric_name:String, val:Number, [freq:Number]): void
- **metricAddDetailCount**(metric_name:String, key:String|IPAddress, count:Number): void
- **metricAddDetailDataset**(metric_name:String, key:String|IPAddress, count:Number): void
- **metricAddDetailMax**(metric_name:String, key:String|IPAddress, count:Number): void
- **metricAddDetailSampleset**(metric_name:String, key:String|IPAddress, count:Number): void
- **metricAddDetailSnap**(metric_name:String, key:String|IPAddress, count:Number): void
- **metricAddMax**(metric_name:String, val:Number): void
- **metricAddSampleset**(metric_name:String, val:Number): void
- **metricAddSnap**(metric_name:String, count:Number): void

Notes:

- Freq is the number of occurrences of the value being passed in. If the value is not passed in, the value of freq is 1. The freq argument is useful in cases when you want to simultaneously record multiply occurrences of particular values in a dataset.
- When NaN is passed to a `metricAdd*` function, it is silently discarded.
- All count parameters for `metricAdd*` functions only accept a non-zero, positive integer between 1 and 2^64.

See Also

- **Example: Database Trigger**
- **Example: Parse Syslog Over TCP with Universal Payload Analysis**
- **Example: Session Table**
- **Example: SOAP Request**

NFS

The NFS class allows retrieval of metrics available during the `NFS_REQUEST` and `NFS_RESPONSE` events.

*Go to **Classes List**.*

Events

NFS_REQUEST: Event
Fires on every NFS request processed by the device.

*Go to **Events List**.*

NFS_RESPONSE: Event
Fires on every NFS response processed by the device.

*Go to **Events List**.*

Properties

accessTime: Number (`NFS_RESPONSE` only)
The time it took for the server to access a file on disk, expressed in milliseconds. For NFS, it is the time from every non-pipelined READ and WRITE command in an NFS flow until the payload containing the response is recorded by the ExtraHop system.

authMethod: String (version 4.0.21474+)
The method for authenticating users.

error: String (`NFS_RESPONSE` only)
The detailed error message recorded by the ExtraHop system.

fileHandle: **Buffer**
The file handle returned by the server on LOOKUP, CREATE, SYMLINK, MKNOD, LINK, or READDIRPLUS operations.

isCommandFileInfo: Boolean (version 4.0.21474+)
Returns true if the command is a file info.

isCommandRead: Boolean
Returns true if the command is a read.

isCommandWrite: Boolean
Returns true if the command is a write.

method: String
The NFS method (appears under **Methods** in the UI).

offset: Number (`NFS_REQUEST` only, version 4.0.21300+)
The file offset associated with READ and WRITE NFS commands.

reqBytes: Number (`NFS_RESPONSE` only)
The number of L4 request bytes.

reqL2Bytes: Number (`NFS_RESPONSE` only)
The number of L2 request bytes.

reqPkts: Number (`NFS_RESPONSE` only)
The number of request packets.

reqRTO: Number (`NFS_REQUEST` only)
The number of request RTOs.

reqSize: Number
The size of the request record at L7, expressed in bytes.

resource: String
The path and filename, concatenated together.

roundTripTime: Number (`NFS_RESPONSE` only)
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number (`NFS_RESPONSE` only)
The number of L4 response bytes.

rspL2Bytes: Number (`NFS_RESPONSE` only)
The number of L2 response bytes.

rspPkts: Number (`NFS_RESPONSE` only)
The number of response packets.

rspRTO: Number (`NFS_RESPONSE` only)
Number of response RTOs.

rspSize: Number (`NFS_RESPONSE` only)
The size of the response record at L7, expressed in bytes.

statusCode: String (version 4.0.21474+)
The NFS status code of the request or response.

user: String
The Linux user ID on the system. Uses the format `uid:`*`xxxx`*`@ip_address`.

version: Number
The NFS version.

Remote.HTTP

(version 4.1.23251+)

The `Remote.HTTP` class allows submission of HTTP requests to an HTTP Open Data Stream Configuration previously configured in the ExtraHop Admin UI. This includes the ability to use HTTP REST APIs. Consult with your ExtraHop appliance administrator for the possible values to use for HTTP Data Stream Configuration names.

*Go to **Classes List**.*

Methods

request

Submits an HTTP REST request to an HTTP Data Stream Configuration previously configured in the ExtraHop Admin UI.

Syntax:

```
Remote.HTTP("name").request("method", {path: "path", [headers: headers],
[payload: "payload"]})
```

```
Remote.HTTP.request("method", {path: "path", [headers: headers],
[payload: "payload"]})
```

Parameters:

method: String
String specifying the HTTP method to be used.

- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
- OPTIONS
- CONNECT
- PATCH

options: Object
The options object has the following properties:

path: String
The string specifying the request path.

headers: Object
The optional object specifying the request headers.

It is possible to compress the outgoing HTTP requests by using the Content-Encoding header.

```
'Content-Encoding': 'gzip'
```

The following values are supported for this compression header:

- gzip
- deflate

payload: String | **Buffer**
The optional string or **Buffer** specifying the request payload.

name: String
The name of the HTTP Data Stream Configuration previously configured in the ExtraHop Admin UI. If no name is specified, the request will go to the first (default) Data Stream Configuration.

Return Value:
Returns **true** if the request is queued, otherwise returns **false**.

Helper Methods

The following helper methods allow you to more easily make use of the most common HTTP methods.

- **Remote.HTTP.delete**
- **Remote.HTTP.get**
- **Remote.HTTP.patch**
- **Remote.HTTP.post**
- **Remote.HTTP.put**

Syntax:

```
Remote.HTTP("name").delete({path: "path", [headers: headers],
[payload: "payload"]})
```

```
Remote.HTTP.delete({path: "path", [headers: headers], [payload: "payload"]})
```

```
Remote.HTTP("name").get({path: "path", [headers: headers],
[payload: "payload"]})
```

```
Remote.HTTP.get({path: "path", [headers: headers], [payload: "payload"]})
```

```
Remote.HTTP("name").patch({path: "path", [headers: headers],
[payload: "payload"]})
```

```
Remote.HTTP.patch({path: "path", [headers: headers], [payload: "payload"]})
```

```
Remote.HTTP("name").post({path: "path", [headers: headers],
[payload: "payload"]})
```

```
Remote.HTTP.post({path: "path", [headers: headers], [payload: "payload"]})
```

```
Remote.HTTP("name").put({path: "path", [headers: headers],
[payload: "payload"]})
```

```
Remote.HTTP.put({path: "path", [headers: headers], [payload: "payload"]})
```

Parameters:

All of these helper methods take the following parameters:

options: Object
The options object has the following properties:

path: String
The string specifying the request path.

headers: Object
The optional object specifying the request headers.

payload: String
The optional string specifying the request payload.

name: String
The name of the HTTP Data Stream Configuration previously configured in the ExtraHop Admin UI. If no name is specified, the request will go to the first (default) Data Stream Configuration.

Return Value:
Returns **true** if the request is queued, otherwise returns **false**.

Examples:

HTTP GET

The following example will issue an HTTP GET request to the HTTP configuration `my_destination` and a path that is the URI, including query string variables, that you want the request to be sent to.

```
Remote.HTTP("my_destination").get( { path: "/?foo=bar&baz=stuff" } );
```

HTTP POST

The following example will issue an HTTP POST request to the HTTP configuration `my_destination`, the path that is the URI you want the request to be sent to and a payload. The payload can be data similar to what an HTTP client would send, a JSON blob, XML, or whatever else you want to send.

```
Remote.HTTP("my_destination").post( { path: "/", payload: "stuff I want to
send" } );
```

Custom HTTP Headers

The following example defines a Javascript object with keys to represent the header names and their corresponding values and provide that in a call as the value for the `headers` key.

```
var my_json = { foo: "stuff", bar: 42, baz: false };
var headers = { "Content-Type": "application/json" };
Remote.HTTP("my_destination").post( { path: "/", headers: headers, payload:
JSON.stringify(my_json) });
```

See Also:

- **Example: Send Data to ElasticSearch with Remote.HTTP**
- **Example: Send Information to Azure Table Service with Remote.HTTP**

Remote.MongoDB

(version 4.0.18429+)

The `Remote.MongoDB` class allows insertion, removal, and updating of documents in collections in MongoDB.

*Go to **Classes List**.*

Methods

insert

Inserts a document or array of documents into a collection, and handles both add and modify operations.

Syntax:

```
Remote.MongoDB.insert("collection", "document");
```

```
Remote.MongoDB("name").insert("collection", "document");
```

Parameters:

collection: String
The name of a group of MongoDB documents.

document: String
The JSON-formatted document to insert into the collection.

name: String (version 4.1.24352+)
The name of the host as it appears in the ExtraHop Open Data Streams UI. If no host is specified, the default host will be used.

Return Value:

Returns **true** if the request is queued, otherwise returns **false**.

Examples:

```
Remote.MongoDB.insert('sessions.sess_www',
    {
    'session_id': "100",
    'path': "/index.html",
    'host': "www.extrahop.com",
    'status': "500",
    'src_ip': "10.10.1.120",
    'dst_ip': "10.10.1.100"
    }
);
```

```
var x = Remote.MongoDB.insert('test.tbc', {'foobar': 1});
if (x) {
```

```
    Network.metricAddCount('perf_trigger_success', 1);
}
else {
    Network.metricAddCount('perf_trigger_error', 1);
}
```

Refer to http://docs.mongodb.org/manual/reference/method/db.collection.insert/#db.collection.insert for more information.

remove

Removes documents from a collection.

Syntax:

```
Remote.MongoDB.remove("collection", "document", [justOnce]);
```

```
Remote.MongoDB("name").remove("collection", "document", [justOnce]);
```

Parameters:

collection: String
The name of a group of MongoDB documents.

document: String
The JSON-formatted document to remove from the collection.

justOnce: Boolean
An optional boolean parameter used to limit the removal to just one document. Set to "true" to limit the deletion. The default value is false.

name: String (version 4.1.24352+)
The name of the host as it appears in the ExtraHop Open Data Streams UI. If no host is specified, the default host will be used.

Return Value:

Returns **true** if the request is queued, otherwise returns **false**.

Example:

```
var x = Remote.MongoDB.remove('test.tbc', { qty: 100000}, false);
if (x) {
    Network.metricAddCount('perf_trigger_success', 1);
}
else {
    Network.metricAddCount('perf_trigger_error', 1);
}
```

Refer to
http://docs.mongodb.org/manual/reference/method/db.collection.remove/#db.collection.remove
for more information.

Update

Modifies an existing document or documents in a collection.

Syntax:

```
Remote.MongoDB.update({"collection", "document", "update"}, [{[upsert:true],
[multi:true]}]);
```

```
Remote.MongoDB("name").update({"collection", "document", "update"},
[{[upsert:true], [multi:true]}]);
```

Parameters:

collection: String
The name of a group of MongoDB documents.

document: String
The JSON-formatted document that specifies which documents to update or insert, if upsert option is set to true.

update: String
The JSON-formatted document that specifies how to update the specified documents.

name: String (version 4.1.24352+)
The name of the host as it appears in the ExtraHop Open Data Streams UI. If no host is specified, the default host will be used.

options:
Optional flags that indicate the following additional update options:

upsert: Boolean
An optional boolean parameter to create a new document when no document matches the query data. Set to "true" to create a new document. The default value is false.

multi: Boolean
An optional boolean parameter to update multiple documents that match the query data. Set to "true" to update multiple documents. The default value is false, which updates only the first document returned.

Return Value:

Returns **true** if the request is queued, otherwise returns **false**.

Example:

```
var x = Remote.MongoDB.update('test.tbc', {_id: 1}, {$set: {'foobar':2}},
{upsert:true, multi:false} );
```

```
if (x) {
    Network.metricAddCount('perf_trigger_success', 1);
}
else {
    Network.metricAddCount('perf_trigger_error', 1);
}
```

Refer to http://docs.mongodb.org/manual/reference/method/db.collection.update/#db.collection.update for more information.

See Also:

- **Example: Parse Syslog Over TCP with Universal Payload Analysis**

Remote.Syslog

(version 4.0.18429+)

The `Remote.Syslog` class allows creation of remote syslog messages with specified content.

*Go to **Classes List***

Each of these methods sends a message to the configured remote syslog server with a severity corresponding to the method name using the "user" facility. You can specify the specific host using the 'name' field and using the name as it appears in the ExtraHop Open Data Streams UI. If no host is specified, the default host will be used.

- **emerg**(message: String): void
- **alert**(message: String): void
- **crit**(message: String): void
- **error**(message: String): void
- **warn**(message: String): void
- **notice**(message: String): void
- **info**(message: String): void
- **debug**(message: String): void

For instance, to send an rsyslog message to the default host for every HTTP response that includes the URI, request and response sizes, and server processing time, add the following trigger on the `HTTP_RESPONSE` event:

```
Remote.Syslog.info("eh_event=web uri=" + HTTP.uri + " req_size=" + HTTP.reqSize + "
rsp_size=" + HTTP.rspSize + " tprocess=" + HTTP.tprocess);
```

To send an rsyslog message to the host '`name`' for every HTTP response that includes the URI, request and response sizes, and server processing time, add the following trigger on the `HTTP_RESPONSE` event (version 4.1.24352+):

```
Remote.Syslog("name").info("eh_event=web uri=" + HTTP.uri + " req_size=" +
HTTP.reqSize + " rsp_size=" + HTTP.rspSize + " tprocess=" + HTTP.tprocess);
```

If submitting an rsyslog message succeeds, the APIs will return true. In the case of either success or failure, the trigger will continue to execute as a failure to submit an rsyslog message is a "soft" failure. Incorrect usage of the APIs, i.e. calling them with the wrong number or type of arguments, will still result in trigger execution stopping.

Message size

By default, the message sent to the remote server is limited to 1024 bytes, including the message header and trailer (if necessary). The message header always includes the priority and timestamp, which together are up to 30 bytes.

To increase the default message size, go to the Admin UI, click **Running Config**, and then click **Edit**. Go to the "capture" section, and under "rsyslog", add "message_length_max". The "message_length_max" setting applies only to the message passed to the Remote.Syslog APIs, the message header does not count against the max. Sample configuration:

```
"remote": {
    "rsyslog": {
    "host": "splunkium",
    "port": 54322,
    "ipproto": "tcp",
    "message_length_max": 4000
    }
}
```

Timestamp

The timestamp format for rsyslog messages is expressed in UTC in versions before 4.1.24484 and after 4.1.24504. For example:

```
2015-07-08T23:42:35.075Z
```

For versions 4.1.24484 through 4.1.24504, , the timestamp format for rsyslog messages is expressed in local time with a offset. For example:

```
2015-07-08T13:47:32.724-10:00
```

See Also

- **Example: Device Discovery Notification**
- **Example: Parse Syslog Over TCP with Universal Payload Analysis**
- **Example: Topnset Key Matching**

RemoteSyslog

Note: Deprecated. Use **Remote.Syslog** on page 100 instead.

*Go to **Classes List**.*

Each of these methods send a message to the configured remote syslog server with a severity corresponding to the method name using the "user" facility.

- **emerg**(message: String): void
- **alert**(message: String): void
- **crit**(message: String): void
- **error**(message: String): void
- **warn**(message: String): void
- **notice**(message: String): void
- **info**(message: String): void
- **debug**(message: String): void

For instance, to send an rsyslog message for every HTTP response that includes the URI, request and response sizes, and server processing time, add the following trigger on the `HTTP_RESPONSE` event:

```
RemoteSyslog.info("eh_event=web uri=" + HTTP.uri + " req_size=" + HTTP.reqSize + "
rsp_size=" + HTTP.rspSize + " tprocess=" + HTTP.tprocess);
```

If submitting an rsyslog message succeeds, the APIs will return true. In the case of either success or failure, the trigger will continue to execute as a failure to submit an rsyslog message is a "soft" failure. Incorrect usage of the APIs, i.e. calling them with the wrong number or type of arguments, will still result in trigger execution stopping.

By default, the message sent to the remote server is limited to 1024 bytes, including the message header and trailer (if necessary). The message header always includes the priority and timestamp, which together are up to 30 bytes.

To increase the default message size, go to the **Admin UI**, click **Running Config**, and then click **Edit**. Go to the "remote" section, and under "rsyslog", add "message_length_max". The "message_length_max" setting applies only to the message passed to the RemoteSyslog APIs, the message header does not count against the max. Sample configuration:

```
"remote": {
    "rsyslog": {
        "host": "splunkium",
        "port": 54322,
        "ipproto": "tcp",
        "message_length_max": 4000
    }
}
```

RTCP

(version 4.1.22706+)

The RTCP class allows for retrieval of metrics available during the `RTCP_MESSAGE` event.

*Go to **Classes List**.*

Events

RTCP_MESSAGE: Event
Fires on every RTCP UDP packet processed by the device.

*Go to **Events List**.*

Properties

callId: String
The Call ID for associating with a SIP flow.

packets: Array
An array of RTCP packet objects where each object represents a packet and contains a **packetType** field. Each object has different fields based on the message type, as described below.

packetType: String
The type of packet. If the packet type is not recognizable, then the packetType will be "Unknown *n*" where *n* is the RTP control packet type value.

Value	Type	Name
194	SMPTETC	SMPTE time-code mapping
195	IJ	Extended inter-arrival jitter report
200	SR	sender report
201	RR	receiver report
202	SDES	source description
203	BYE	goodbye
204	APP	application-defined
205	RTPFB	Generic RTP Feedback
206	PSFB	Payload-specific
207	XR	extended report
208	AVB	AVB RTCP packet
209	RSI	Receiver Summary Information
210	TOKEN	Port Mapping
211	IDMS	IDMS Settings

APP packet objects have the following fields:

name: String
The name chosen by the person defining the set of APP packets to be unique. Interpreted as four case-sensitive ASCII characters.

ssrc: Number
The SSRC of the sender.

value: **Buffer**
The optional application-dependent data.

BYE packet objects have the following fields:

packetType: Number
Contains the number 203 to identify this as an RTCP BYE packet.

SR packet objects have the following fields:

ntpTimestamp: Number
The NTP timestamp, converted to milliseconds since the epoch (January 1, 1970).

reportBlocks: Array
An array of report objects which contain:

fractionLost: Number
The 8-bit number indicating the number of packets lost divided by the number of packets expected.

jitter: Number
An estimate of the statistical variance of the RTP data packet interarrival time, expressed in milliseconds.

lastSR: Number
The middle 32 bits of the ntp_Timestamp received as part of the most recent RTCP sender report (SR) packet from the source SSRC. If no SR has been received yet, this field is set to zero.

lastSRDelay: Number
The delay between receiving the last SR packet from the source SSRC and sending this reception block, expressed in units of 1/65536 seconds. If no SR packet has been received yet, this field is set to zero.

packetsLost: Number
The total number of RTP data packets from the source SSRC that have been lost since the beginning of reception.

seqNum: Number
The highest sequence number received from the source SSRC.

ssrc: Number
The SSRC of the sender:

rtpTimestamp: Number
The RTP timestamp, converted to milliseconds since the epoch (January 1, 1970).

senderOctets: Number
The sender octet count.

senderPkts: Number
The sender packet count.

RR packet objects have the following fields:

reportBlocks: Array
An array of report objects which contain:

fractionLost: Number

The 8-bit number indicating the number of packets last divided by the number of packets expected.

jitter: Number
An estimate of the statistical variance of the RTP data packet interarrival, expressed in milliseconds.

lastSR: Number
The middle 32 bits of the ntp_Timestamp received as part of the most recent RTCP sender report (SR) packet from the source SSRC. If no SR has been received yet, this field is set to zero.

lastSRDelay: Number
The delay between receiving the last SR packet from the source SSRC and sending this reception report block, expressed in units of 1/65536 seconds. If no SR packet has been received yet, this field is set to zero.

packetsLost: Number
The total number of RTP data packets from the source SSRC that have been lost since the beginning of reception.

seqNum: Number
The highest sequence number received from the source SSRC.

ssrc: Number
The SSRC of the sender.

ssrc: Number
The SSRC of the sender.

SDES packet objects have the following fields:

descriptionBlocks: Array
An array of objects that contain:

type: Number
The SDES type.

SDES Type	Abbrev.	Name
0	END	end of SDES list
1	CNAME	canonical name
2	NAME	user name
3	EMAIL	user's electronic mail address
4	PHONE	user's phone number
5	LOC	geographic user location
6	TOOL	name of application or tool
7	NOTE	notice about the source
8	PRIV	private extensions
9	H323-C ADDR	H.323 callable address
10	APSI	Application Specific Identifier

value: **Buffer**
A buffer containing the text portion of the SDES packet.

ssrc: Number
The SSRC of the sender.

XR packet objects have the following fields:

ssrc: Number
The SSRC of the sender.

xrBlocks: Array
An array of report blocks which contain:

statSummary: Object (type 6 only)
The statSummary object contains the following properties:

beginSeq: Number
The beginning sequence number for the interval.

devJitter: Number
The standard deviation of the relative transit time between each two packet series in the sequence interval.

devTTLOrHL: Number
The standard deviation of TTL or Hop Limit values of data packets in the sequence number range.

dupPackets: Number
The number of duplicate packets in the sequence number interval.

endSeq: Number
The ending sequence number for the interval.

lostPackets: Number
The number of lost packets in the sequence number interval.

maxJitter: Number
The maximum relative transmit time between two packets in the sequence interval, expressed in milliseconds.

maxTTLOrHL: Number
The maximum TTL or Hop Limit value of data packets in the sequence number range.

meanJitter: Number
The mean relative transit time between two packet series in the sequence interval, rounded to the nearest value expressible as an RTP timestamp, expressed in milliseconds.

meanTTLOrHL: Number
The mean TTL or Hop Limit value of data packets in the sequence number range.

minJitter: Number
The minimum relative transmit time between two packets in the sequence interval, expressed in milliseconds.

minTTLOrHL: Number
The minimum TTL or Hop Limit value of data packets in the sequence number range.

ssrc: Number

The SSRC of the sender.

type: Number
The XR block type.

Block Type	Name
1	Loss RTE Report Block
2	Duplicate RLE Report Block
3	Packet Receipt Times Report Block
4	Receiver Reference Time Report Block
5	DLRR Report Block
6	Statistics Summary Report Block
7	VoIP Metrics Report Block
8	RTCP XP
9	Texas Instruments Extended VoIP Quality Block
10	Post-repair Loss RLE Report Block
11	Multicast Aquisition Report Block
12	IBMS Report Block
13	ECN Summary Report
14	Measurement Information Block
15	Packet Delay Variation Metrics Block
16	Delay Metrics Block
17	Burst/Gap Loss Summary Statistics Block
18	Burst/Gap Discard Summary Statistics Block
19	Frame Impairment Statistics Summary
20	Burst/Gap Loss Metrics Block
21	Burst/Gap Discard Metrics Block
22	MPEG2 Transport Stream PSI-Independent Decodability Statistics Metrics Block
23	De-Jitter Buffer Metrics Block
24	Discard Count Metrics Block
25	DRLE (Discard RLE Report)
26	BDR (Bytes Discarded Report)
27	RFISD (RTP Flows Initial Synchronization Delay)
28	RFSO (RTP Flows Synchronization Offset Metrics Block)
29	MOS Metrics Block
30	LCB (Loss Concealment Metrics Block)

Block Type	Name
31	CSB (Concealed Seconds Metrics Block)
32	MPEG2 Transport Stream PSI Decodability Statistics Block

typeSpecific: Number
The contents of this field depend on the block type.

value: **Buffer**
The contents of this field depend on the block type.

voipMetrics: Object (type 7 only)
The voipMetrics object contains the following properties:

burstDensity: Number
The fraction of RTP data packets within burst periods since the beginning of reception that were either lost or discarded.

burstDuration: Number
The mean duration, expressed in milliseconds, of the burst periods that have occurred since the beginning of reception.

discardRate: Number
The fraction of RTP data packets from the source that have been discarded since the beginning of reception, due to late or early arrival, under-run or overflow at the receiving jitter buffer.

endSystemDelay: Number
The most recently estimated end system delay, expressed in milliseconds.

extRFactor: Number
The external R factor quality metric. A value of 127 indicates this parameter is unavailable.

gapDensity: Number
The fraction of RTP data packets within inter-burst gaps since the beginning of reception that were either lost or discarded.

gapDuration: Number
The mean duration of the gap periods that have occurred since the beginning of reception, expressed in milliseconds.

gmin: Number
The gap threshold.

jbAbsMax: Number
The absolute maximum delay, expressed in milliseconds, that the adaptive jitter buffer can reach under worst case conditions.

jbMaximum: Number
The current maximum jitter buffer delay, which corresponds to the earliest arriving packet that would not be discarded, expressed in milliseconds.

jbNominal: Number
The current nominal jitter buffer delay, which corresponds to the nominal jitter buffer delay for packets that arrive exactly on time, expressed in milliseconds.

lossRate: Number
The fraction of RTP data packets from the source lost since the beginning of reception.

mosCQ: Number
The estimated mean opinion score for conversational quality (MOS-CQ). A value of 127 indicates this parameter is unavailable.

mosLQ: Number
The estimated mean opinion score for listening quality (MOS-LQ). A value of 127 indicates this parameter is unavailable.

noiseLevel: Number
The noise level, expressed in decibels.

rerl: Number
The residual echo return loss value, expressed in decibels.

rFactor: Number
The R factor quality metric. A value of 127 indicates this parameter is unavailable.

roundTripDelay: Number
The most recently calculated round-trip time (RTT) between RTP interfaces, expressed in milliseconds.

rxConfig: Number
The receiver configuration byte.

signalLevel: Number
The voice signal relative level, expressed in decibels.

ssrc: Number
The SSRC of the sender.

RTP

(version 4.1.22706)

The RTP class allows for retrieval of metrics available during the `RTP_OPEN`, `RTP_CLOSE` and `RTP_TICK` events.

*Go to **Classes List**.*

Events

RTP_CLOSE: Event
Fires when an RTP connection is closed.

*Go to **Events List**.*

RTP_OPEN: Event
Fires when a new RTP connection is opened.

*Go to **Events List**.*

RTP_TICK: Event
Fires periodically on RTP flows.

*Go to **Events List**.*

Properties

bytes: Number (`RTP_TICK` only)
The number of bytes sent.

callId: String
The call ID for associating with SIP flow.

drops: Number (`RTP_TICK` only)
The number of dropped packets detected.

dups: Number (`RTP_TICK` only)
The number of duplicate packets detected.

jitter: Number (`RTP_TICK` only)
An estimate of the statistical variance of the data packet interarrival time.

l2Bytes: Number (RTP_TICK only)
The number of L2 bytes.

mos: Number (`RTP_TICK` only, version 4.1.23229+)
The estimated mean opinion score for quality.

outOfOrder: Number (`RTP_TICK` only)
The number of out-of-order messaged detected.

payloadType: String (`RTP_TICK` only)
The type of RTP payload.

payloadTypeId	payloadType
0	ITU-T G.711 PCMU Audio
3	GSM 6.10 Audio
4	ITU-T G.723.1 Audio
5	IMA ADPCM 32kbit Audio

payloadTypeId	payloadType
6	IMA ADPCM 64kbit Audio
7	LPC Audio
8	ITU-T G.711 PCMA Audio
9	ITU-T G.722 Audio
10	Linear PCM Stereo Audio
11	Linear PCM Audio
12	QCELP
13	Comfort Noise
14	MPEG Audio
15	ITU-T G.728 Audio
16	IMA ADPCM 44kbit Audio
17	IMA ADPCM 88kbit Audio
18	ITU-T G.729 Audio
25	Sun CellB Video
26	JPEG Video
28	Xerox PARC Network Video
31	ITU-T H.261 Video
32	MPEG Video
33	MPEG-2 Transport Stream
34	ITU-T H.263-1996 Video

payloadTypeId: Number (`RTP_TICK` only)
The numeric value of the payload type. See table under **payloadType**.

pkts: Number (`RTP_TICK` only)
The number of packets sent.

rFactor: Number (`RTP_TICK` only)
The R factor quality metric.

ssrc: Number
The SSRC of sender.

version: Number
The RTP version number.

Sampleset

The Sampleset class is used to represent sampleset metrics.

*Go to **Classes List**.*

Properties

count: Number
The number of samples in the sampleset.

mean: Number
The average value of the samples.

sigma: Number
The standard deviation.

sum: Number
The sum of the samples.

sum2: Number
The sum of the squares of the samples.

SDP

The SDP class allows for retrieval of Session Description Protocol (SDP) information during the `SIP_REQUEST` and `SIP_RESPONSE` events.

*Go to **Classes List**.*

Properties

mediaDescriptions: Array (`SIP_REQUEST` and `SIP_RESPONSE`)
An array of objects containing the following fields:

attributes: Array of Strings
The optional session attributes.

bandwidth: Array of Strings
The optional proposed bandwidth type and bandwidth to be used by the session or media.

connectionInfo: String
The connection data, including network type, address type and connection adddress. May also contain optional sub-fields, depending on the address type.

description: String
The session description which may contain one or more media descriptions. Each media description consists of media, port and transport protocol fields.

encryptionKey: String
The optional encryption method and key for the session.

mediaTitle: String
The title of the media stream.

sessionDescription: Object (`SIP_REQUEST` and `SIP_RESPONSE`)
An object containing the following fields:

attributes: Array of Strings
The optional session attributes.

bandwidth: Array of Strings
The optional proposed bandwidth type and bandwidth to be used by the session or media.

connectionInfo: String
The connection data, including network type, address type and connection address. May also contain optional sub-fields, depending on the address type.

email: String
The optional email address. If present, this can contain multiple email addresses.

encryptionKey: String
The optional encryption method and key for the session.

origin: String
The originator of the session, including username, address of the user's host, a session identifier, and a version number.

phoneNumber: String
The optional phone number. If present, this can contain multiple phone numbers.

sessionInfo: String
The session description.

sessionName: String

The session name.

timezoneAdjustments: String
The adjustment time and offset for a scheduled session.

uri: String
The optional URI intended to provide more information about the session.

version: String
The version number. This should be 0.

timeDescriptions: Array (`SIP_REQUEST` and `SIP_RESPONSE`)
An array of objects containing the following fields:

repeatTime: String
The session repeat time, including interval, active duration, and offsets from start time.

time: String
The start time and stop times for a session.

Session

The Session object is in-memory and global per ExtraHop appliance. It is designed to support coordination across multiple independently executing triggers. The ExtraHop Open Data Context API exposes the Session object via the management network, enabling coordination with external processes. The Session object's global state means any changes by a trigger or external process becomes visible to all other entities on the same ExtraHop appliance using the Session object. Because the Session object is in-memory, changes are not saved when you restart the ExtraHop appliance or the capture process.

> Note: ECM cluster nodes do not share their global state. An ECM does not execute triggers; it only manages them.

*Go to **Classes List**.*

Events

SESSION_EXPIRE: Event

The `SESSION_EXPIRE` event fires periodically (in approximately 30 second increments) as long as the session table is in use. It does not fire once for every expired entry, and it does not fire immediately after an entry has expired.

*Go to **Events List**.*

Methods

add (key:String, value *, [options:Object]): *

Returns the given key in the session table. If the key is present, the corresponding value is returned without modifying the entry. If the key is not present, a new entry is created for the given key and value, and the new value is returned.

getOptions (key:String): Object

Returns the Options object for the entry corresponding to the one passed in through `Session.add` or `Session.replace`.

increment (key:String, [count:Number]): Number | Null

Atomic lookup and increment. The default count value is 1. On success, the new value is returned. If lookup fails, null is returned. If the value is not a number, an exception is thrown.

lookup (key:String): *

Look up the given key in the session table and return the corresponding value. Returns null if the key is not present.

modify (key:String, value:*, [options:Object]): * (version 3.9.17050+)

Modify the entry associated with the given key. If the key is present, update the value and return the previous value. If the key is not present, no new entry is created. If options are provided, the options of the entry are updates and old options are merged with new ones. If the "expire" option is provided, the expiration timer is reset.

remove (key:String): *

Removes the entry for the given key and returns the associated value.

replace (key:String, value:*, [options:Object]): * (versions 3.9.17031-3.9.17555, 3.10.18339+)

Update the entry associated with the given key. If the key is present, update the value and return the previous value. If the key is not present, add the entry and return the previous value (null). If options are provided, the options of the entry are updated and old options are merged with new ones. If the "expire" option is provided, the expiration timer is reset.

Constants

PRIORITY_LOW: Number
Default value is 0.**Example: Session Table**

PRIORITY_NORMAL: Number
Default value is 1

PRIORITY_HIGH: Number
Default value is 2.

Properties

expiredKeys: Array (`SESSION_EXPIRE` only)
An array of objects with the following properties:

age: Integer
The age of the expired object, expressed in milliseconds. Age is the amount of time elapsed between when the object in the session table was added or modified, and the `SESSION_EXPIRE` event.

name: String
The key of the expired object.

value: Number | String | IPAddress | Boolean | Device
The value of the entry in the session table.

Session table **options** are as follows:

expire: Number
The duration after which eviction will occur, expressed in seconds. If null or undefined, the entry will be evicted only when the session table grows too large. This is the default.

notify: Boolean
Indicates whether the key will be available on `SESSION_EXPIRE` events. The default value is false.

priority: String
A constant, `Session.PRIORITY_{LOW, NORMAL, HIGH}`, used to determine which entries to evict if the session table grows too large. The default value is `PRIORITY_NORMAL`.

The `SESSION_EXPIRE` event is not associated with any particular flow, so triggers on `SESSION_EXPIRE` events cannot commit device metrics in the usual way (e.g., `Device.metricAdd*` or `Flow.client.device.metricAdd*`). To commit device metrics on this event, you have to add any needed Device objects to the session table using the `Device()` constructor.

Deprecated

update (key:String, value:*, [options:Object]) * (version 3.9.17031+)
Deprecated. Use `Session.replace` instead.

See Also

- **Example: Session Table**

SIP

(version 4.1.22706+)

The SIP class allows retrieval of metrics available during the `SIP_REQUEST` and `SIP_RESPONSE` events.

*Go to **Classes List**.*

Events

SIP_REQUEST: Event
Fires on every SIP request processed by the device.

*Go to **Events List**.*

SIP_RESPONSE: Event
Fires on every SIP response processed by the device.

*Go to **Events List**.*

Methods

findHeaders(name: String): Array
Allows access to SIP header values. The result is an array of header objects (with **name** and **value** properties) where the names match the prefix of the string passed to **findHeaders**.

Properties

callId: String
The call ID for this message.

from: String
The contents of the From header.

hasSDP: Boolean
Returns true if this event includes SDP information.

headers: Object
An array-like object that allows access to SIP header names and values. Access a specific header using one of these methods:

string property:
The name of the header, accessible in a dictionary-like fashion. For example:

```
var headers = SIP.headers;
session = headers["X-Session-Id"];
accept = headers.accept;
```

numeric property:
The order in which headers appear on the wire. The returned object has a name and a value property. Numeric properties are useful for iterating over all the headers and disambiguating headers with duplicate names. For example:

```
for (i = 0; i < headers.length; i++) {
 hdr = headers[i];
 debug("headers[" + i + "].name: " + hdr.name);
```

```
debug("headers[" + i + "].value: " + hdr.value);
}
```

Note: Saving SIP.headers to the Flow store does not save all of the individual header values. It is best practice to save the individual header values to the Flow store.

method: String
The SIP method.

Method Name	Description
ACK	Confirms the client has received a final response to an INVITE request.
BYE	Terminates a call. Can be sent by either the caller or the callee.
CANCEL	Cancels any pending request
INFO	Sends mid-session information that doesn't change the session state.
INVITE	Invites a client to participate in a call session.
MESSAGE	Transports instant messages using SIP.
NOTIFY	Notify the subscriber of a new event.
OPTIONS	Queries the capabilities of servers.
PRACK	Provisional Acknowledgement.
PUBLISH	Publish an event to the server.
REFER	Ask recipient to issue a SIP request (call transfer).
REGISTER	Registers the address listed in the To header field with a SIP server.
SUBSCRIBE	Subscribes for an event of Notification from the Notifier.
UPDATE	Modifies the state of a session without changing the state of the dialog.

processingTime: Number (`SIP_RESPONSE` only)
The time between the request and the first response, expressed in milliseconds.

reqBytes: Number
The number of L4 request bytes.

reqL2Bytes: Number
The number of L2 request bytes.

reqPkts: Number
The number of request packets.

reqRTO: Number
The number of request RTOs.

reqSize: Number (`SIP_REQUEST` only)
The size of the request payload, expressed in bytes. Does not include headers.

roundTripTime: Number

The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number
The number of L4 response bytes.

rspL2Bytes: Number
The number of L2 response bytes.

rspPkts: Number
The number of response packets.

rspRTO: Number
The number of response RTOs.

rspSize: Number (`SIP_RESPONSE` only)
The size of the response payload, expressed in bytes. Does not include headers.

statusCode: Number (`SIP_RESPONSE` only)
The SIP response status code.

Provisional Responses

Number	Response
100	Trying
180	Ringing
181	Call is Being Forwarded
182	Queued
183	Session In Progress
199	Early Dialog Terminated

Successful Responses

Number	Response
200	OK
202	Accepted
204	No Notification

Redirection Responses

Number	Response
300	Multiple Choice
301	Moved Permanently
302	Moved Temporarily
305	Use Proxy
380	Alternative Service

Client Failure Responses

Number	Response
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Conditional Request Failed
413	Request Entity Too Large
414	Request URI Too Long
415	Unsupported Media Type
416	Unsupported URI Scheme
417	Unknown Resource Priority
420	Bad Extension
421	Extension Required
422	Session Interval Too Small
423	Interval Too Brief
424	Bad Location Information
428	Use Identity Header
429	Provide Referrer Identity
430	Flow Failed
433	Anonymity Disallowed
436	Bad Identity Info
437	Unsupported Certificate
438	Invalid Identity Header
439	First Hop Lacks Outbound Support
470	Consent Needed
480	Temporarily Unavailable
481	Call/Transaction Does Not Exist

Number	Response
482	Loop Detected
483	Too Many Hops
484	Address Incomplete
485	Ambiguous
486	Busy Here
487	Request Terminated
488	Not Acceptable Here
489	Bad Event
491	Request Pending
493	Undecipherable
494	Security Agreement Required

Server Failure Responses

Number	Response
500	Server Internal Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Server Timeout
505	Version Not Supported
513	Message Too Large
580	Precondition Failure

Global Failure Responses

Name	Response
600	Busy Everywhere
603	Decline
604	Does Not Exist Anywhere
606	Not Acceptable

to: String
The contents of the To header.

uri: String
The URI for SIP request or response.

SMPP

The SMPP class allows retrieval of metrics available during the `SMPP_REQUEST` and `SMPP_RESPONSE` events.

> **Note**: The `mdn`, `shortcode`, and `error` properties may be null, depending on availability and relevance.

*Go to **Classes List**.*

Events

SMPP_REQUEST: Event
Fires on every SMPP request processed by the device.

*Go to **Events List**.*

SMPP_RESPONSE: Event
Fires on every SMPP response processed by the device.

*Go to **Events List**.*

Properties

command: String
The SMPP command ID.

destination: String
The destination address as specified in the `SMPP_REQUEST`. Will be null if this is not available for the current command type.

error: String (`SMPP_RESPONSE` only)
The error code corresponding to `command_status`. If the command status is ROK, the value of error will be null.

message: **Buffer** (`SMPP_REQUEST` only, version 4.1.23411+)
The contents of the short_message field on DELIVER_SM and SUBMIT_SM messages. Will be null if unavailable or not applicable.

reqSize: Number
The size of the request, expressed in bytes.

reqTimeToLastByte: Number
The time from the first byte of the request until the last byte of the request, expressed in milliseconds. Returns NaN on malformed and aborted requests or expired flows.

rspSize: Number (`SMPP_RESPONSE` only)
The size of the response, expressed in bytes.

rspTimeToFirstByte: Number (`SMPP_RESPONSE` only)
The time from the first byte of the request until the first byte of the response, expressed in milliseconds. Returns NaN on malformed and aborted responses or expired flows.

rspTimeToLastByte: Number (`SMPP_RESPONSE` only)
The time from the first byte of the request until the last byte of the response, expressed in milliseconds. Returns NaN on malformed and aborted responses or expired flows.

source: String
The source address as specified in the `SMPP_REQUEST`. Will be null if this is not available for the current command type.

tprocess: Number (`SMPP_RESPONSE` only)

The server processing time, expressed in milliseconds. Equivalent to `rspTimeToFirstByte - reqTimeToLastByte`. Returns NaN on malformed and aborted responses or expired flows.

SMTP

(version 4.0.18766+)

The SMTP class allows retrieval of metrics available during the SMTP_REQUEST and SMTP_RESPONSE events.

*Go to **Classes List**.*

Events

SMTP_REQUEST: Event
Fires on every SMTP request processed by the device.

*Go to **Events List**.*

SMTP_RESPONSE: Event
Fires on every SMTP response processed by the device.

*Go to **Events List**.*

Properties

dataSize: Number
The size of the attachment, expressed in bytes.

domain: String
The domain of the address the message is coming from.

error: String (`SMTP_RESPONSE` only, version 4.1.23176+)
The error string corresponding to the status code.

headers: Object
An object that allows access to SMTP header names and values.

isEncrypted: Boolean
Returns true if the application is encrypted using STARTTLS encryption.

isReqAborted: Boolean
Returns true if the connection is closed before the SMTP request is complete.

isRspAborted: Boolean (`SMTP_RESPONSE` only)
Returns true if the connection is closed before the SMTP response is complete.

method: String
The SMTP method.

recipient: String
The address the message should be sent to.

recipientList: Array of Strings
A list of recipient addresses.

reqBytes: Number
The number of L4 request bytes.

reqL2Bytes: Number
The number of request L2 bytes.

reqPkts: Number
The number of request packets.

reqRTO: Number
The number of request RTOs.

reqSize: Number
The size of the request in bytes.

reqTimeToLastByte: Number
The time from the first byte of the request until the last byte of the request, expressed in milliseconds. Not valid on malformed and aborted requests, or expired flows.

roundTripTime: Number
The median TCP round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

rspBytes: Number
The number of L4 response bytes.

rspL2Bytes: Number
The number of response L2 bytes.

rspPkts: Number
The number of response packets.

rspRTO: Number
The number of response RTOs.

rspSize: Number (`SMTP_RESPONSE` only)
The size of the response, expressed in bytes.

rspTimeToFirstByte: Number (`SMTP_RESPONSE` only)
The time from the first byte of the request until the last byte of the request, expressed in milliseconds. Not valid on malformed and aborted requests, or expired flows.

rspTimeToLastByte: Number (`SMTP_RESPONSE` only)
The time from the first byte of the request until the last byte of the response, expressed in milliseconds. Not valid on malformed and aborted requests, or expired flows.

sender: String
The sender of the message.

statusCode: Number (`SMTP_RESPONSE` only)
The SMTP status code of the response.

statusText: String (`SMTP_RESPONSE` only)
The multi-line response string.

tprocess: Number (`SMTP_RESPONSE` only)
The server processing time, expressed in milliseconds. Equivalent to `rspTimeToFirstPayload - reqTimeToLastByte`. Not valid on malformed and aborted responses, or expired flows.

SSL

The SSL class allows retrieval of metrics available during the `SSL_OPEN`, `SSL_CLOSE`, `SSL_ALERT`, `SSL_RECORD`, `SSL_HEARTBEAT`, and `SSL_RENEGOTIATE` events.

*Go to **Classes List**.*

Events

SSL_ALERT: Event
Fires when an SSL alert record is exchanged.

*Go to **Events List**.*

SSL_CLOSE: Event
Fires when the SSL connection is shut down.

*Go to **Events List**.*

SSL_HEARTBEAT: Event (version 3.10.19872+)
Fires when an SSL heartbeat record is exchanged.

*Go to **Events List**.*

SSL_OPEN: Event
Fires when the SSL connection is first established.

*Go to **Events List**.*

SSL_PAYLOAD: Event (version 4.1.24320+)
Fires when the decrypted SSL payload matches the criteria configured in the associated trigger.

Depending on the **FLOW**, the payload can be found in the following:

- flow.client.payload
- flow.payload1
- flow.payload2
- flow.receiver.payload
- flow.sender.payload
- flow.server.payload

*Go to **Events List**.*

SSL_RECORD: Event (version 3.8.15910+)
Fires when an SSL record is exchanged.

*Go to **Events List**.*

SSL_RENEGOTIATE: Event (version 4.0.20129+)
Fires on SSL renegotiation.

*Go to **Events List**.*

Methods

getClientExtensionData(extension_name | extension_id): **Buffer**
Returns the extension data if the extension was passed as part of the hello message from the client and had data. Returns null otherwise.

getServerExtensionData(extension_name | extension_id): **Buffer**

Returns the extension data if the extension was passed as part of the hello message from the server and had data. Returns null otherwise.

hasClientExtension(extension_name | extension_id): boolean
Returns true if the extension was passed as part of the hello message from the client.

hasServerExtension(extension_name | extension_id): boolean
Returns true if the extension was passed as part of the hello message from the server.

Properties

alertCode: Number (`SSL_ALERT` only)
If the session is opaque, the value is `SSL.ALERT_LEVEL_UNKNOWN (255)`. Otherwise, it corresponds to the numeric part of the AlertDescription data structure specified in RFC2246.

alertLevel: Number (`SSL_ALERT` only)
If the session is opaque, the value is `SSL.ALERT_LEVEL_UNKNOWN (255)`. Otherwise, it corresponds to the numeric part of the AlertLevel data structure as specified in RFC2246.

certificate: SSLCert
The SSL certificate object associated with the communication. Each object has the following properties:

fingerprint: String
The string hex representation of the SHA-1 hash of the certificate. This is the same string shown in most browsers' certificate information dialog boxes, but without spaces. For example:

```
"55F30E6D49E19145CF680E8B7E3DC8FC7041DC81"
```

keySize: Number
The certificate key size.

notAfter: Number
The certificate expiration time in UTC.

publicKeyExponent: String
A string hex representation of the public key's exponent. This is the same string shown in most browsers' certificate information dialog boxes, bit without spaces.

publicKeyModulus: String
A string hex representation of the public key's modulus. This is the same string shown in most browser's certificate information dialog boxes, but without spaces. For example:

```
"010001"
```

signatureAlgorithm: String (version 4.1.22459+)
The algorithm used to sign the certificate. Some possible values are:

- From RFC 3279:
  - md2WithRSAEncryption
  - md5WithRSAEncryption
  - sha1WithRSAEncryption
- From RFC 4055:
  - sha224WithRSAEncryption
  - sha256WithRSAEncryption
  - sha384WithRSAEncryption
  - sha512WithRSAEncryption

- From RFC 4491:
  - id-GostR3411-94-with-Gost3410-94
  - id-GostR3411-94-with-Gost3410-2001

subject: String
The certificate subject CN string.

cipherSuite: String
String representing the cryptographic cipher suite negotiated between the server and the client.

clientExtensions: Array (`SSL_OPEN` and `SSL_RENEGOTIATE` only)
An array of extension objects. Each object has the following properties:

id: Number
The ID number of the SSL extension

name: String
The name of the SSL extension, if known. Otherwise "unknown" will be used.

Known SSL Extensions

ID	Name
0	server_name
1	max_fragment_length
2	client_certificate_url
3	trusted_ca_keys
4	truncated_hmac
5	status_request
6	user_mapping
7	client_authz
8	server_authz
9	cert_type
10	elliptic_curves
11	ec_point_formats
12	srp
13	signature_algorithms
14	use_srtp
15	heartbeat
16	application_layer_protocal_negotiation
17	status_request_v2
18	signed_certificate_timestamp
19	client_certificate_type
20	server_certificate_type
35	SessionTicket TLS

ID	Name
65281	renegotiation_info

clientSessionId: String
The client session ID, byte array encoded as a string.

contentType: String (`SSL_RECORD` only)
The content type for the current record.

handshakeTime: Number (`SSL_OPEN` and `SSL_RENEGOTIATE` only)
The amount of time required to negotiate the SSL connection, expressed in milliseconds. This is the amount of time between the client sending ClientHello and the server sending ChangeCipherSpec.

HEARTBEAT_TYPE_REQUEST: Number
The heartbeatType is a request as specified in RFC 6520.

HEARTBEAT_TYPE_RESPONSE: Number
The heartbeatType is a response as specified in RFC 6520.

HEARTBEAT_TYPE_UNKNOWN: Number
A heartbeatType that is unknown because the connection was not decrypted.

heartbeatPayloadLength: Number (`SSL_HEARTBEAT` only)
The payload_length field of the HeatbeatMessage data structure as specified in RFC 6520.

heartbeatType: Number (`SSL_HEARTBEAT` only)
The HeartbeatMessageType field of the HeatbeartMessage data structure as specified in RFC 6520.

host: string (`SSL_OPEN` and `SSL_RENEGOTIATE` only)
The value of the SSL Server Name Indication (SNI), if present.

isAborted: Boolean (`SSL_CLOSE` only)
Returns true if the SSL session is aborted.

isCompressed: Boolean
Returns true if the SSL record is compressed.

isV2ClientHello: Boolean
Returns true if the Hello record corresponds to SSLv2.

privateKeyId: String
Returns null if the ExtraHop applicance is not decrypting the SSL traffic. Returns a string ID associated with the private key if the ExtraHop appliance is decrypting the SSL traffic.

To find the private key ID in the ExtraHop Admin UI, got to the **Configuration** section, click **Capture**, click **SSL Decryption**, and then click a certificate. The pop-up window displays all identifiers for the certificate.

recordLength: Number (`SSL_RECORD`, `SSL_ALERT`, and `SSL_HEARTBEAT` only)
The length field of the TLSPlaintext, TLSCompressed, and TLSCiphertext data structures as specified in RFC 5246.

recordType: Number (`SSL_RECORD`, `SSL_ALERT`, and `SSL_HEARTBEAT` only)
The type of field in the TLSPlaintext, TLSCompressed, and TLSCiphertext data structures as specified in RFC 5246.

reqBytes: Number (`SSL_RECORD` and `SSL_CLOSE` only)
The number of request bytes.

reqL2Bytes: Number (`SSL_RECORD` and `SSL_CLOSE` only)

The number of L2 request bytes.

reqPkts: Number (`SSL_RECORD` and `SSL_CLOSE` only)
The number of request packets.

rspBytes: Number (`SSL_RECORD` and `SSL_CLOSE` only)
The number of response bytes.

rspL2Bytes: Number (`SSL_RECORD` and `SSL_CLOSE` only)
The number of L2 response bytes.

rspPkts: Number (`SSL_RECORD` and `SSL_CLOSE` only)
The number of response packets.

roundTripTime: Number (`SSL_RECORD` and `SSL_CLOSE` only)
The median round-trip time (RTT), expressed in milliseconds. May be NaN if there are no RTT samples.

serverExtensions: Array (`SSL_OPEN` and `SSL_RENEGOTIATE` only)
An array of extension objects. Each object has the following properties:

id: Number
The ID number of the SSL extension

name: String
The name of the SSL extension, if known. Otherwise "unknown" will be used. See `clientExtensions` above for a list of known extension names.

serverSessionId: String
The server session ID, byte array encoded as a string.

version: Number
The SSL protocol version with the RFC hexidecimal version number expressed as a decimal.

Version	Hex	Decimal
SSLv3	0x300	768
TLS 1.0	0x301	769
TLS 1.1	0x302	770
TLS 1.2	0x303	771

Telnet

The Telnet class allows retrieval of metrics available during the `TELNET_MESSAGE` event.

*Go to **Classes List**.*

Events

TELNET_MESSAGE: Event
Fires on a telnet command or line of data from the telnet client or server.

*Go to **Events List**.*

Properties

command: String
The command type. Will be null if the event fired due to a line of data being sent.

The possible values returned are:

- Abort
- Abort Output
- Are You There
- Break
- Data Mark
- DO
- DON'T
- End of File
- End of Record
- Erase Character
- Erase Line
- Go Ahead
- Interrupt Process
- NOP
- SB
- SE
- Suspend
- WILL
- WON'T

line: String
A line of the data sent by the client or server. Terminal escape sequences and special characters are filtered out. Things like cursor movement/line editing are not currently simulated (with the exception of backspace characters).

option: String
The option being negotiated. Will be null if the command is not an option command.

The possible values are:

- 3270-REGIME
- AARD
- ATCP
- AUTHENTICATION
- BM
- CHARSET
- COM-PORT-OPTION
- DET

- ECHO
- ENCRYPT
- END-OF-RECORD
- ENVIRON
- EXPOPL
- EXTEND-ASCII
- FORWARD-X
- GMCP
- KERMIT
- LINEMODE
- LOGOUT
- NAOCRD
- NAOFFD
- NAOHTD
- NAOHTS
- NAOL
- NAOLFD
- NAOP
- NAOVTD
- NAOVTS
- NAWS
- NEW-ENVIRON
- OUTMRK
- PRAGMA-HEARTBEAT
- PRAGMA-LOGON
- RCTE
- RECONNECT
- REMOTE-SERIAL-PORT
- SEND-LOCATION
- SEND-URL
- SSPI-LOGON
- STATUS
- SUPDUP
- SUPDUP-OUTPUT
- SUPPRESS-GO-AHEAD
- TERMINAL-SPEED
- TERMINAL-TYPE
- TIMING-MARK
- TN3270E
- TOGGLE-FLOW-CONTROL
- TRANSMIT-BINARY
- TTYLOC
- TUID
- X-DISPLAY-LOCATION
- X.3-PAD
- XAUTH

optionData:**Buffer**

For option subnegotiations (the "SB" command), the raw, option-specific data sent. Will be null if the command is not "SB".

Topnset

The Topnset class is used to represent topnset metrics. A topnset metric contains a list of entries with keys and values. Values may be numbers, **Dataset**, **Sampleset**, or even other Topnsets. **Topnset Keys** are objects that represent a property of Topnset.

*Go to **Classes List**.*

Methods

findEntries(keyPattern: Object): Array
Returns all entries with matching keys.

findKeys(keyPattern: Object): Array
Returns all keys matching the specified pattern.

lookup(keyPattern: Object): *
Look up an item in the topnset, returning the first that matches the pattern.

Properties

entries: Array
An array of the topnset entries. The array contains *N* objects with key and value properties, with *N* currently being set to 1000.

Topnset Keys

Topnset Keys are objects that represent a property of **Topnset**.

*Go to **Classes List**.*

Properties

type: String

The type of the topnset key. Possibilities include:

- string
- int
- ipaddr
- ether
- device_id

value: *

Varies depending on the type of key.

- For string keys, the value is a string.
- For int keys, the value is a number.
- For ipaddr keys, the value is an object containing:
  - addr
  - proto
  - port
  - device_oid
  - origin
  - custom_devices
- For ether keys, the value is an object containing:
  - ethertype
  - hwaddr

TroubleGroup

In ExtraHop 4.0, it is possible to create your own trouble group. A trouble group is a set of devices for which a potential problem has been identified. This class may be used in conjunction with metric and discovery events (`NEW_DEVICE`, `METRIC_CYCLE_BEGIN`, `METRIC_RECORD_COMMIT`) to identify potential performances, security or configuration problems.

*Go to **Classes List**.*

Methods

addDevice(troubleGroupName: String, device: Device): void
Adds a device to the specified trouble group.

See Also

- **Example: Create a Custom Trouble Group**

Turn

Turn is top-level object available in the `FLOW_TURN` event.

*Go to **Classes List**.*

Properties

clientBytes: Number (version 4.0.20256+)
The size of the request that the client transferred, expressed in bytes.

clientTransferTime: Number (version 4.0.20256+)
The request transfer time, expressed in milliseconds. Corresponds to the **Network In** metric in the ExtraHop Web UI.

processingTime: Number (version 4.0.20256+)
The time elapsed between the client having transferred the request to the server and the server beginning to transfer the response back to the client, expressed in milliseconds. Corresponds to the **Processing Time** metric in the **Turn Timing** sections of the ExtraHop Web UI.

serverBytes: Number (version 4.0.20256+)
The size of the response that the server transferred, expressed in bytes.

serverTransferTime: Number (version 4.0.20256+)
The response transfer time, expressed in milliseconds. Corresponds to the **Network Out** metric in the ExtraHopy Web UI.

thinkTime: Number (version 3.10.19576+)
The time elapsed between the server having transferred the response to the client and the client transferring a new request to the server, expressed in milliseconds. Will return NaN if there is no valid measurement.

Deprecated

reqSize: Number (version 4.0.20256+)
Deprecated. Use `clientBytes`instead.

reqXfer: Number (version 4.0.20256+)
Deprecated. Use `clientTransferTime` instead.

rspSize: Number (version 4.0.20256+)
Deprecated. Use `serverBytes`instead.

rspXfer: Number (version 4.0.20256+)
Deprecated. Use `serverTransferTime` instead.

tprocess: Number (version 4.0.20256+)
Deprecated. Use `processingTime`instead.

VLAN

The VLAN class represents a VLAN on the network.

*Go to **Classes List**.*

Properties

id: Number
The numerical ID of the VLAN.

XML

*Go to **Classes List**.*

Returns parsed XML data. Example:

```
var payload = "<Header><storeid>foo</storeid></Header>"
var xml = new XML(payload);
debug("storeid: " + xml.storeid); // The value is "foo."
```

Examples

The following examples are available:

Example: HTTP Header Object
Shows how to use the HTTP Headers object.

Example: SOAP Request
Tracks SOAP requests via the SOAPAction header and stashes them into the flow store.

Example: Customer ID Header
Records 500 errors by customer ID and URI.

Example: Database Trigger
Records responses and processing time by database query.

Example: CIFS Trigger
Records total read and written bytes as well as bytes written by users not authorized to access a sensitive resource.

Example: Memcache Hits and Misses
Records keys for each hit or miss, and hit access time.

Example: Memcache Key Parsing
Parses the memcache keys to extract detailed breakdowns by module and class name, as well as by ID.

Example: Trigger-Based Application Definition
Creates an ExtraHop application container based on traffic associated with a two-tier application.

Example: Session Table
Records specific transactions to the session table with operating system strings in HTTP.userAgent.

Example: Create a Custom Trouble Group
Creates a custom trouble group.

Example: Topnset Key Matching
Matches Topnset Keys.

Example: Use the Metric Cycle Store
Illustrates the use of the Metric Cycle store.

Example: Device Discovery Notification
Creates remote syslog messages when new devices are discovered.

Example: Parse Custom POS Messages with Universal Payload Analysis
Parses a POS system custom TCP messages, looking for specific values in the 4th to 7th bytes of both response and request messages.

Example: Parse Syslog Over TCP with Universal Payload Analysis
Parses the syslog over TCP and count the syslog activity over time, both network-wide and per device.

Example: Send Data to ElasticSearch with Remote.HTTP
Specify the JSON text suitable for ElasticSearch and send the information using Remote.HTTP

Example: Send Information to Azure Table Service with Remote.HTTP
Sends information to the Microsoft Azure Table Service via Remote.HTTP.

Example: HTTP Header Object

```
/*
 * Trigger: HTTP Headers Example
 * Description: Shows how to use the HTTP Headers object
 * Event: HTTP_RESPONSE
 */
var hdr,
    session,
    accept,
    results,
    headers = HTTP.headers,
    i;

/* Header lookups are case-insensitive properties */
session = headers["X-Session-Id"];

/*
 * session is a string representing the value of the header (or null,
 * if the header was not present). header values are always strings,
 * even for headers that typically have numbers as values.
 */

/* This syntax works too, if the header is a legal property name */
accept = headers.accept;
/*
 * In the event that there are multiple instances of a header,
 * accessing the header in the above manner (as a property)
 * will always return the value for the first appearance of the
 * header.
 */
if (session !== null)
{
    /* Count requests per session ID */
    Device.metricAddCount("req_count", 1);
    Device.metricAddDetailCount("req_count", session, 1);
}

/*
 * Looping over all headers
 *
 * "length" is a special property (case-sensitive) that is not
 * treated as a header lookup, but instead returns the number of
 * headers (as if HTTP.headers were an array). in the unlikely
 * event that there is a header called "Length," it would still be
 * accessible with HTTP.headers["Length"] (or HTTP.headers.Length).
 *
 */
for (i = 0; i < headers.length; i++) {
    hdr = headers[i];
```

```
    debug("headers[" + i + "].name: " + hdr.name);
    debug("headers[" + i + "].value: " + hdr.value);
    Device.metricAddCount("hdr_count", 1);
    /* Count instances of each header */
    Device.metricAddDetailCount("hdr_count", hdr.name, 1);
}

/* Searching for headers by prefix */
results = HTTP.findHeaders("Content-");

/*
 * result is an array (a real javascript array, as opposed to
 * an array-like object) of header objects (with name and value
 * properties) where the names match the prefix of the string passed
 * to findHeaders.
 */
for (i = 0; i < results.length; i++) {
    hdr = results[i];
    debug("results[" + i + "].name: " + hdr.name);
    debug("results[" + i + "].value: " + hdr.value);
}
```

See Also

- **Device**
- **HTTP**

Example: SOAP Request

```
/*
 * Name: SOAP Action
 * Description: Tracks SOAP requests via the SOAPAction header and stashes them
 * into the flow store. The big requirement is that the SOAP implementation
 * actually passes this information. Not all do, so be sure and confirm that
 * part.
 * Events: HTTP_REQUEST, HTTP_RESPONSE
 */
var soapAction,
    headers = HTTP.headers,
    method,
    detailMethod,
    parts;

if (event === "HTTP_REQUEST") {
    soapAction = headers["SOAPAction"]
    if (soapAction != null) {
        Flow.store.soapAction = soapAction;
    }
}
else if (event === "HTTP_RESPONSE") {
    soapAction = Flow.store.soapAction;
    if (soapAction != null) {
        parts = soapAction.split("/");
        if (parts.length > 0) {
            method = soapAction.split("/")[1];
        }
        else {
            method = soapAction;
        }
        detailMethod = method + "_detail";
        Network.metricAddCount(method, 1);
        Network.metricAddDetailCount(method_detail, Flow.client.ipaddr, 1);
        Network.metricAddSampleset("soap_proc", HTTP.tprocess);
        Network.metricAddDetailSampleset("soap_proc_detail", method,
                                         HTTP.tprocess);
    }
}
```

See Also

- **FLOW**
- **HTTP**
- **Network**

Example: Customer ID Header

```
/*
 * Name: Customer ID
 * Description: Record 500 errors by customer ID and URI.
 * Event: HTTP_REQUEST
 */
var custId,
    query,
    uri,
    key;

if (event === "HTTP_REQUEST") {
    custId = HTTP.headers["Cust-ID"];
    /* Only keep the URI if there is a customer id */
    if (custId !== null) {
        Flow.store.custId = custId;

        query = HTTP.query;

        /*
         * Pull the complete URI (URI plus query string) and record it on
         * the Flow store for a subsequent response event.
         *
         * The query string data is only available on the request.
         */
        uri = HTTP.uri;
        if ((uri !== null) && (query !== null)) {
            uri = uri  + "?" + query;
        }

        /* Keep URIs for handling by HTTP_RESPONSE triggers */
        Flow.store.uri = uri;
    }
}
else if (event === "HTTP_RESPONSE")
{
    /*
     * Name: HTTP Cust-ID 500 URI Tracking
     * Comment: Record 500 errors by Customer ID and URI.
     * Event: HTTP_RESPONSE
     */
    custId = Flow.store.custId;

    /* Count total requests by customer ID */
    Device.metricAddCount("custid_rsp_count", 1);
    Device.metricAddDetailCount("custid_rsp_count_detail", custId, 1);

    /* If the status code is 500 or 503, record the URI and customer ID */
    if ((HTTP.statusCode === 500) || (HTTP.statusCode === 503))
```

```
{
        /* combine URI and customer ID to create the detail key */
        key = custId;
        if (Flow.store.uri != null) {
            key += ", " + Flow.store.uri;
        }
        Device.metricAddCount("custid_error_count", 1);
        Device.metricAddDetailCount("custid_error_count_detail", key, 1);
    }
}
```

See Also

- **Device**
- **FLOW**
- **HTTP**

Example: Database Trigger

```
/*
 * Name: DB Full Statement
 * Description: Record responses and processing times by database query
 * Events: DB_REQUEST, DB_RESPONSE
 *
 * Note: Assign this trigger to all the devices in a group and then
 * create a custom page for the Capture that displays the Network metric
 * to show the counts across the device group.
 */
var stmt = Flow.store.stmt;

if (event === "DB_REQUEST")
{
    /* Store the statement for the response. */
    Flow.store.stmt = DB.statement || DB.procedure;
    return;
}
if ((typeof stmt === "undefined") || (stmt === null))
{
    /*
     * Restrict statement length to 1024 bytes, since statements larger than
     * this are unlikely to be useful.
     */
    if (stmt.length > 1024) {
        stmt = stmt.substr(0, 1023);
    }

    /* Remove common blank line from front of DB procedures */
    if ((stmt.length > 0) && (stmt[0] === "\n")) {
        stmt = stmt.slice(1);
    }
    /* Record counts by statement */
    Device.metricAddCount("db_rsp_count", 1);
    Device.metricAddDetailCount("db_rsp_count_detail", stmt, 1);
    /* Record processing times by statement */
    Device.metricAddSampleset("db_proc_time", DB.tprocess);
    Device.metricAddDetailSampleset("db_proc_time_detail",
                                    stmt, DB.tprocess);

    /* Record these metrics at the network level as well */
    Network.metricAddCount("db_rsp_count", 1);
    Network.metricAddDetailCount("db_rsp_count_detail", stmt, 1);
    Network.metricAddSampleset("db_proc_time", DB.tprocess);
    Network.metricAddDetailSampleset("db_proc_time_detail",
                                     stmt, DB.tprocess);
}
```

See Also

- **DB**
- **Device**
- **FLOW**
- **Network**

Example: CIFS Trigger

```
/*
 * Name: CIFS Audit Support
 * Description: Records total read and written bytes as well as
 * bytes written by users not authorized to access a sensitive resource.
 * Event: CIFS_RESPONSE
 */
var client = Flow.client.device,
    server = Flow.server.device,
    clientAddress = Flow.client.ipaddr,
    serverAddress = Flow.server.ipaddr,
    file = CIFS.resource,
    user = CIFS.user,
    resource,
    permissions,
    writeBytes,
    readBytes;

/* Resource to monitor */
resource = "\\Clients\\Confidential\\";
/* Users of interest and their permissions */
permissions = {
    "\\\\EXTRAHOP\\tom" : {read: false, write: false},
    "\\\\Anonymous" : {read: true, write: false},
    "\\\\WORKGROUP\\bob" : {read: true, write: true}
};

/* Check if this is an action on our monitored resource */
if ((file !== null) && (file.indexOf(resource) !== -1)) {
    if (CIFS.isCommandWrite) {
        writeBytes = CIFS.reqSize;
        /* Record bytes written */
        Device.metricAddCount("cifs_write_bytes", writeBytes);
        Device.metricAddDetailCount("cifs_write_bytes", user, writeBytes);

        /* Record number of writes */
        Device.metricAddCount("cifs_writes", 1);
        Device.metricAddDetailCount("cifs_writes", user, 1);
        /* Record number of unauthorized writes */
        if (!permissions[user] || !permissions[user].write) {
            Device.metricAddCount("cifs_unauth_writes", 1);
            Device.metricAddDetailCount("cifs_unauth_writes", user, 1);
        }
    }

    if (CIFS.isCommandRead) {
        readBytes = CIFS.reqSize;
        /* Record bytes read */
        Device.metricAddCount("cifs_read_bytes", readBytes);
```

```
        Device.metricAddDetailCount("cifs_read_bytes", user, readBytes);
        /* Record number of reads */
        Device.metricAddCount("cifs_reads", 1);
        Device.metricAddDetailCount("cifs_reads", user, 1);

        /* Record number of unauthorized reads */
        if (!permissions[user] || !permissions[user].read) {
            Device.metricAddCount("cifs_unauth_reads", 1);
            Device.metricAddDetailCount("cifs_unauth_reads", user, 1);
        }
    }
}
```

See Also

- **CIFS**
- **Device**
- **FLOW**

Example: Memcache Hits and Misses

```
/*
* Name: Memcache_hits_misses
* Description: Records keys for each hit or miss, and hit access time.
* Event: MEMCACHE_RESPONSE
*/
var hits = Memcache.hits;
var misses = Memcache.misses;
var accessTime = Memcache.accessTime;
var i;

Device.metricAddCount('memcache_key_hit', hits.length);

for (i = 0; i < hits.length; i++) {
    var hit = hits[i];
    if (hit.key != null) {
        Device.metricAddDetailCount('memcache_key_hit_detail', hit.key, 1);
    }
}

if (!isNaN(accessTime)) {
    Device.metricAddSampleset('memcache_key_hit', accessTime);
    if ((hits.length > 0) && (hits[0].key != null)) {
        Device.metricAddDetailSampleset('memcache_key_hit_detail', hits[0].key,
                                        accessTime);
    }
}

Device.metricAddCount('memcache_key_miss', misses.length);

for (i = 0; i < misses.length; i++) {
    var miss = misses[i];
    if (miss.key != null) {
        Device.metricAddDetailCount('memcache_key_miss_detail', miss.key, 1);
    }
}
```

See Also

- **Device**
- **Memcache**

Example: Memcache Key Parsing

```
/*
 * Name: Memcache_Example_Trigger
 * Description: Parses the memcache keys to extract detailed breakdowns. Keys
 * look like:
 *     "com.extrahop.<module>.<class>_<id>"
 *     (for example: "com.extrahop.widgets.sprocket_12345")
 * and we want breakdowns by module and class name as well as ID.
 * Event: MEMCACHE_RESPONSE
 */

var method = Memcache.method;
var statusCode = Memcache.statusCode;
var reqKeys = Memcache.reqKeys;
var hits = Memcache.hits;
var misses = Memcache.misses;
var error = Memcache.error;
var hit;
var miss;
var key;
var size;
var reqKey;
var i;

/* Record breakdown of hit count and value size by module and class: */
for (i = 0; i < hits.length; i++) {
    hit = hits[i];
    key = hit.key;
    size = hit.size;

    Device.metricAddCount("hit", 1);
    if (key != null) {
        var parts = key.split(".");

        if ((parts.length == 4) && (parts[0] == "com") &&
            (parts[1] == "extrahop")) {
            var module = parts[2];
            var subparts = parts[3].split("_");

            Device.metricAddDetailCount("hit_module", module, 1);
            Device.metricAddDetailSampleset("hit_module_size", module, size);

            if (subparts.length == 2) {
                var hitClass = module + "." + subparts[0];

                Device.metricAddDetailCount("hit_class", hitClass, 1);
                Device.metricAddDetailSampleset("hit_class_size", hitClass,
                                                size);
            }
```

```
        }
    }
}

/*
 * Users have reported slowness accessing sprockets. Record misses by ID to help
 * identify caching issues:
 */
for (i = 0; i < misses.length; i++) {
    miss = misses[i];
    key = miss.key;
    if (key != null) {
        var parts = key.split(".");

        if ((parts.length == 4) && (parts[0] == "com") &&
            (parts[1] == "extrahop") && (parts[2] == "widgets")) {
            var subparts = parts[3].split("_");

            if ((subparts.length == 2) && (subparts[0] == "sprocket")) {
                Device.metricAddDetailCount("sprocket_miss_id", subparts[1], 1);
            }
        }
    }
}

/* Record the key(s) that produced any errors: */
if (error != null && method != null) {
    for (i = 0; i < reqKeys.length; i++) {
        reqKey = reqKeys[i];
        if (reqKey != null) {
            var errDetail = method + " " + reqKey + " / "  + statusCode + ": " +
                            error;
            Device.metricAddDetailCount("error_key", errDetail, 1);
        }
    }
}

/* Record the status code, matching built-in metrics */
if (Memcache.isBinaryProtocol && statusCode != "NO_ERROR") {
    Device.metricAddDetailCount("status_code",
                                method + "/" + statusCode,
                                1);
}
else {
    Device.metricAddDetailCount("status_code", statusCode, 1);
}
```

See Also

- **Device**
- **Memcache**

Example: Trigger-Based Application Definition

```
/*
 * Name: Application Builder - My App
 * Description: Trigger to create an ExtraHop application container based
 *              on traffic associated with a two-tier application
 * Events: HTTP_RESPONSE, DB_RESPONSE
 * Assignments: All HTTP servers that process application HTTP traffic
 *              All DB servers that process application DB traffic
 * Firmware: 3.9+ required, 3.10+ recommended
*/

// Initialize the Application object against which we'll
// commit specific HTTP and DB transactions.  After traffic is
// committed, an application container "My App" will appear in the
// Applications tab in the WebUI.
var myApp = Application("My App");
// These configurable properties describe features that define
// our application traffic.

var myAppHTTPHost = "myapp.internal.example.com";
var myAppDatabaseName = "myappdb";

if (event == "HTTP_RESPONSE") {

    // HTTP transactions can be committed to an Application on
    // HTTP_RESPONSE events.

    // Commit this HTTP transaction only if the HTTP Host header for
    // this response is defined and matches our application's HTTP Host.

    if (HTTP.host && (HTTP.host == myAppHTTPHost)) {

        myApp.commit();

        // Capture custom metrics about user agents that experience
        // HTTP 40x or 50x responses.

        if (HTTP.statusCode && (HTTP.statusCode >= 400))
{

            // Increment the overall count of 40x or 50x responses
            myApp.metricAddCount('myapp_40x_50x', 1);

            // Collect additional detail on referer, if any

            if (HTTP.referer) {
                myApp.metricAddDetailCount('myapp_40x_50x_refer_detail',
                                           HTTP.referer, 1);
            }
```

```
        }

    }

} else if (event == "DB_RESPONSE") {
    // Database transactions can be committed to an Application on
    // DB_RESPONSE events.

    // Commit this database transaction only if the database name for
    // this response matches the name of our application database.
    if (DB.database && (DB.database == myAppDatabaseName)) {
        myApp.commit();
    }

}
```

See Also

- **Application**
- **DB**
- **HTTP**

Example: Session Table

```
/*
 * Name: Session Table
 * Description: Records specific transactions to the session table with operating
   system strings in HTTP.userAgent.
 * Events: HTTP_REQUEST, SESSION_EXPIRE
*/

    /* HTTP_REQUEST */
    if (HTTP.userAgent === null) {
        return;
    }

    /* Look for the OS name */
    var re = /(Windows|Mac|Linux)/;
    var os = HTTP.userAgent.match(re);
    if (os === null) {
        return;
    }
    /* Use matched string as key for session table entry */
    os = os[0];

    var opts =
{
        /* Expire added entries after 30 seconds */
        expire: 30,
        /* Retain entries with normal priority if the session table grows too large
*/
        priority: Session.PRIORITY_NORMAL,
        /* Make expired entries available on SESSION_EXPIRE events */
        notify: true
    };
    /* Ensure an entry for this key is present (an existing entry will not be
replaced) */
    Session.add(os, 0, opts);
    /* Increase the count for this entry */
    var count = Session.increment(os);
    debug(os + ": " + count);
After 30 seconds, the accumulated per-OS counts appear in the Session.expiredKeys
list, accessible in the SESSION_EXPIRE event:

    /* SESSION_EXPIRE */
    var keys = Session.expiredKeys;
    for (var i = 0; i < keys.length; i++) {
        debug("count of " + keys[i].name + ": " + keys[i].value);
        if (keys[i].value > 500) {
            Network.metricAddCount("os-high-request-count", 1);
            Network.metricAddDetailCount("os-high-request-count",
                                         keys[i].name, 1);
```

```
        }
    }
```

See Also

- **HTTP**
- **Network**
- **Session**

Example: Create a Custom Trouble Group

```
/*
 * Name: Create a Custom Trouble Group
 * Event: METRIC_RECORD_COMMIT
 * Advanced options: 30sec cycle, extrahop.device.http_server
*/
var fields = MetricRecord.fields,
   tprocess = fields.tprocess,
   slowWebServerThreshold = 200,
   pctl;

pctl = tprocess.percentile(50);

if (pctl > slowWebServerThreshold) {
   TroubleGroup.addDevice('Slow Web Servers', MetricRecord.object);
}
```

See Also

- **MetricRecord**
- **TroubleGroup**

Example: Topnset Key Matching

```
/*
 * Name: Topnset Key Matching
 * Event: METRIC_RECORD_COMMIT
 * Advanced options: 30sec cycle, extrahop.device.net_detail
*/
var stat = MetricRecord.fields['bytes_out'],
   entry,
   entries,
   key,
   i;

entries = stat.findEntries({addr: /192.168.112.1*/, proto: 17});

debug('matched ' + entries.length + '/' + stat.entries.length + ' entries')};

for (i = 0; i < entries.length; i++) {
   entry = entries[i];
   key = entry.key;
   Remote.Syslog.alert('unexpected outbound UDP traffic from: ' +
                       JSON.stringify(key));
}
```

See Also

- **MetricRecord**
- **Remote.Syslog**

Example: Use the Metric Cycle Store

```
/*
 * Name: Using the Metric Cycle Store
 * Event: METRIC_CYCLE_BEGIN, METRIC_RECORD_COMMIT, METRIC_CYCLE_END
 * Advanced options: 30sec cycle, extrahop.device.http_server, extrahop.device.tcp
*/
var store = MetricCycle.store;

function processMetric() {
   var id = MetricRecord.id,
       deviceId = MetricRecord.object.id,
       fields = MetricRecord.fields;

   if (!store.metrics[deviceId]) {
       store.metrics[deviceId] = {};
   }
   if (id === 'extrahop.device.http_server') {
       store.metrics[deviceId].httpRspAborted= fields['rsp_abort'];
   }
   else if (id === 'extrahop.device.tcp') {
       store.metrics[deviceId].tcpAborted = fields['aborted_out'];
   }
}

function commitSyntheticMetrics() {
   var dev,
       metrics,
       abortPct,
       deviceId;
   for (deviceId in store.metrics) {
       metrics = store.metrics[deviceId];
       abortPct = (metrics.httpRspAborted / metrics.tcpAborted) * 100;
       dev = new Device(deviceId);
       dev.metricAddSnap('http-tcp-abort-pct', abortPct);
   }
}

switch (event) {
case 'METRIC_CYCLE_BEGIN':
   store.metrics = {};
   break;

case 'METRIC_RECORD_COMMIT':
   processMetric();
   break;

case 'METRIC_CYCLE_END':
   commitSyntheticMetrics();
```

```
    break;
}
```

See Also

- **Device**
- **MetricCycle**
- **MetricRecord**

Example: Device Discovery Notification

```
/*
 * Name: Device Discovery Notification
 * Event: NEW_DEVICE
*/
var dev = Discover.device;
Remote.Syslog.info('Discovered device ' + dev.id + ' (hwaddr: ' + dev.hwaddr + ')
');
```

See Also

- **Device**
- **Discover**
- **Remote.Syslog**

Example: Parse Custom POS Messages with Universal Payload Analysis

Pare

```
/* Name: Parse Custom POS Messages with Universal Payload Analysis
 * Description: Parses a POS system custom TCP messages, looking for specific
values
 *              in the 4th to 7th bytes of both response and request messages.
 * Event: TCP_PAYLOAD
 * Version: 4.1.24080
 */

//
// Define variables, store client or server payload into Buffer
//

var buf_client      = Flow.client.payload,
    buf_server      = Flow.server.payload,
    protocol        = Flow.l7proto,

//
// PoS Message Type Structure Definition
//
    pos_message_type = {
        "0100" : "0100_Authorization_Request",
        "0101" : "0101_Authorization_Request_Repeat",
        "0110" : "0110_Authorization_Response",
        "0200" : "0200_Financial_Request",
        "0201" : "0201_Financial_Request_Repeat",
        "0210" : "0210_Financial_Response",
        "0220" : "0220_Financial_Transaction_Advice_Request",
        "0221" : "0221_Financial_Transaction_Advice_Request_Repeat",
        "0230" : "0230_Financial_Transaction_Advice_Response",
```

```
        "0420" : "0420_Reversal_Advice_Request",
        "0421" : "0421_Reversal_Advice_Request_Repeat",
        "0430" : "0430_Reversal_Advice_Response",
        "0600" : "0600_Administration_Request",
        "0601" : "0601_Administration_Request_Repeat",
        "0610" : "0610_Administration_Response",
        "0620" : "0620_Administration_Advice_Request",
        "0621" : "0621_Administration_Advice_Request_Repeat",
        "0630" : "0630_Administration_Advice_Response",
        "0800" : "0800_Administration_Request",
        "0801" : "0801_Administration_Request_Repeat",
        "0810" : "0810_Administration_Response"
    };

//
// Skip parsing if it's other protocol of no interest or there is no payload
//
if (protocol !== 'tcp:4015' || (buf_client === null && buf_server === null)) {
    //debug('Not interested protocol: ' + protocol);
    return;
} else {
    // Store the data into variables for future access since there is some payload
to parse
    var client_ip       = Flow.client.ipaddr,
        server_ip       = Flow.server.ipaddr,
        client_port     = Flow.client.port,
        server_port     = Flow.server.port;
        //client          = new Device(Flow.client.device.id),
        //server          = new Device(Flow.server.device.id);
}

if (buf_client !== null && buf_client.length >= 7) {

    // This is a client payload
    var cli_msg_type = buf_client.slice(3,7).decode('utf-8');
    debug('Client: ' + client_ip + ":" + client_port + " Type: " + pos_message_type
[cli_msg_type]);
    Device.metricAddCount('UPA_Request', 1);
    Device.metricAddDetailCount('UPA_Request_by_Message', pos_message_type[cli_msg_
type], 1);
    Device.metricAddDetailCount('UPA_Request_by_Client', client_ip.toString(), 1);

} else if (buf_server !== null && buf_server.length >= 7) {

    // This is a server payload
    var srv_msg_type = buf_server.slice(3,7).decode('utf-8');
    debug('Server: ' + server_ip + " Client: " + client_ip + ":" + client_port +"
Type: " + pos_message_type[srv_msg_type]);
    Device.metricAddCount('UPA_Response', 1);
    Device.metricAddDetailCount('UPA_Response_by_Message', pos_message_type[srv_
```

```
msg_type], 1);
    Device.metricAddDetailCount('UPA_Response_by_Client', client_ip.toString(), 1);

} else {

    // No buffer captured situation
    //debug('Null or not enough buffer data');
    return;
}
```

See Also:

- **Device**
- **FLOW**

Example: Parse Syslog Over TCP with Universal Payload Analysis

NOTE: The complete solution that includes this trigger code is available on the ExtraHop forums.

```
/* Name: Parse Syslog over TCP via Universal Payload Analysis
 * Description: Parses the syslog over TCP and count the syslog activity
 *              over time, both network-wide and per device.
 * Event: TCP_PAYLOAD, UDP_PAYLOAD
 * Version: 4.0.22123
 *
 */

//
// Variables we need throughout (aka global)
//

var buffer          = Flow.client.payload,
    buffer_size     = Flow.client.payload.length + 1,
    client          = new Device(Flow.client.device.id),
    data_as_json    = { client_ip       : Flow.client.ipaddr.toString(),
                        client_port     : Flow.client.port.toString(),
                        server_ip       : Flow.server.ipaddr.toString(),
                        server_port     : Flow.server.port.toString(),
                        protocol        : 'syslog',
                        protocol_fields : {} },
    protocol        = Flow.l7proto,
    server          = new Device(Flow.server.device.id),
    syslog          = {},
    syslog_facility = {
        "0": "kern",
        "1": "user",
        "2": "mail",
        "3": "daemon",
```

```
        "4": "auth",
        "5": "syslog",
        "6": "lpr",
        "7": "news",
        "8": "uucp",
        "9": "clock_daemon",
        "10": "authpriv",
        "11": "ftp",
        "12": "ntp",
        "13": "log_audit",
        "14": "log_alert",
        "15": "cron",
        "16": "local0",
        "17": "local1",
        "18": "local2",
        "19": "local3",
        "20": "local4",
        "21": "local5",
        "22": "local6",
        "23": "local7",
    },
    syslog_priority = {
        "0": "emerg",
        "1": "alert",
        "2": "crit",
        "3": "err",
        "4": "warn",
        "5": "notice",
        "6": "info",
        "7": "debug",
    };

//
// Exit out early if not classified properly or no payload
//

if ( ( protocol != 'tcp:5141' ) || ( buffer === null ) ) {
    debug('Invalid protocol ' + protocol +
          ' or null buffer (' + buffer.unpack('z').join(' ') + ')');
    return;
}

//
// Get started parsing Syslog
//

var data = buffer.unpack('z');

// Separate the PRIO field from the rest of the message
var msg_part  = data[0].split('>')[1].split(' ');
```

```
var prio_part = data[0].split('>')[0].split('<')[1];

// Decode the PRIO field into Syslog facility and priority
var raw_facility = parseInt(prio_part) >> 3;
var raw_priority = parseInt(prio_part) & 7;

syslog.facility = syslog_facility[raw_facility];
syslog.priority = syslog_priority[raw_priority];

// Timestamp and hostname are technically part of the HEADER field, but
// we are just treating the rest of the message as a <space> delimited
// string (which it is, the syslog protocol is very basic)
syslog.timestamp = msg_part.slice(0,3).join(' ');
syslog.hostname  = msg_part[3];
syslog.message   = msg_part.slice(4).join(' ');

// At the network level, keep counts of who is sending messages by
// both facility and priority
Network.metricAddCount('syslog:priority_' + syslog.priority, 1);
Network.metricAddDetailCount('syslog:priority_' +
                             syslog.priority    + '_detail',
                             Flow.client.ipaddr, 1);
Network.metricAddCount('syslog:facility_' + syslog.facility, 1);
Network.metricAddDetailCount('syslog:facility_' +
                             syslog.facility    + '_detail',
                             Flow.client.ipaddr, 1);

// Devices receiving messages keep a count of who sent those messages
// by facility and priority
server.metricAddCount('syslog:priority_' + syslog.priority, 1);
server.metricAddDetailCount('syslog:priority_' +
                            syslog.priority    + '_detail',
                            Flow.client.ipaddr, 1);
server.metricAddCount('syslog:facility_' + syslog.facility, 1);
server.metricAddDetailCount('syslog:facility_' +
                            syslog.facility    + '_detail',
                            Flow.client.ipaddr, 1);

// Devices sending messages keep a count of who they sent those messages
// to by facility and priority
client.metricAddCount('syslog:priority_' + syslog.priority, 1);
client.metricAddDetailCount('syslog:priority_' +
                            syslog.priority    + '_detail',
                            Flow.server.ipaddr, 1);
client.metricAddCount('syslog:facility_' + syslog.facility, 1);
client.metricAddDetailCount('syslog:facility_' +
                            syslog.facility    + '_detail',
                            Flow.server.ipaddr, 1);

data_as_json.protocol_fields = syslog;
```

```
data_as_json.ts              = new Date();

//try {
//    Remote.MongoDB.insert('payload.syslog', data_as_json);
//}
//catch ( err ) {
//    Remote.Syslog.debug(JSON.stringify(data_as_json));
//}
debug('Syslog data: ' + JSON.stringify(data_as_json, null, 4));"
        }
```

See Also:

- **FLOW**
- **Network**
- **Remote.MongoDB**
- **Remote.Syslog**

Example: Send Data to ElasticSearch with Remote.HTTP

```
/*
 * Name: Send Data to ElasticSearch with Remote.HTTP
 * Description: Specify the JSON text suitable for ElasticSearch and
 *              send the information using Remote.HTTP
 * Version: 4.1.23251+
 */

var date = new Date();
var payload = {
    'ts' : date.toISOString(), // Timestamp recognized by
Elasticsearch
    'eh_event' : 'http',
    'my_path' : HTTP.path};
var obj = {
    'path' : '/extrahop/http', // Add to extrahop index
    'headers' : {},
    'payload' : JSON.stringify(payload)} ;

Remote.HTTP('elasticsearch').request('POST', obj);
```

See Also:

- **Remote.HTTP**

Example: Send Information to Azure Table Service with Remote.HTTP

Note: This example requires the configuration of Open Datastream for HTTP to use Microsoft Azure request signing.

```
/*
 * Name: Send Information to Azure Table Service with Remote.HTTP
 * Description: Sends information to the Microsoft Azure Table Service via
Remote.HTTP.
 * Version: 4.1.23251+
 */

// the name of the HTTP destination defined in the ODS config
var REST_DEST = "my_table_storage";

// the name of the table within Azure Table Storage
var TABLE_NAME = "TestTable";

/* Without this header set to this value, Azure Table Storage is expecting XML to
 * be sent to it, and it's obviously way easier for a trigger to send JSON.
 * The odata stuff allows us to specify the datatype of fields - in this case
 * that TS is a datetime even though it's serialized from a Date to a String
*/

var headers = { "Content-Type": "application/json;odata=minimalmetadata" };

var now = new Date(getTimestamp());
var msg = {
    "RowKey":        now.getTime().toString(), // must be a string
    "PartitionKey":  "foo", // must be a string
    "HTTPMethod":    HTTP.method,
    "DestAddr":      Flow.server.ipaddr,
    "SrcAddr":       Flow.client.ipaddr,
    "SrcPort":       Flow.client.port,
    "DestPort":      Flow.server.port,
    "TS@odata.type": "Edm.DateTime", // metadata to descibe format of TS field
    "TS":            now.toISOString(),
    "ServerTime":    HTTP.tprocess,
    "RspTTLB":       HTTP.rspTimeToLastByte,
    "RspCode":       HTTP.statusCode.toString(),
    "URI":           "http://" + HTTP.host + HTTP.path,
};

//debug(JSON.stringify(msg));
Remote.HTTP(REST_DEST).post( { path: "/" + TABLE_NAME, headers: headers, payload:
JSON.stringify(msg) } );
```

See Also:

- **Remote.HTTP**