ExtraHop 9.5
Trigger-API-Referenz

Weitere Informationen finden Sie unter https://docs.extrahop.com.

Veröffentlicht: 2024-03-20

ExtraHop Networks
Seattle, WA 98101
877-333-9872 (US)
+44 (0)203 7016850 (EMEA)
+65-31585513 (APAC)
www.extrahop.com

Inhaltsübersicht

Überblick

Application Inspection-Trigger bestehen aus benutzerdefiniertem Code, der bei Systemereignissen über die ExtraHop-Trigger-API automatisch ausgeführt wird. Durch das Schreiben von Triggern können Sie benutzerdefinierte Metrikdaten über die Aktivitäten in Ihrem Netzwerk sammeln. Darüber hinaus können Trigger Operationen ausführen an Protokoll Nachrichten (wie HTTP Anfrage), bevor das Paket verworfen wird.

Das ExtraHop-System überwacht, extrahiert und zeichnet einen Kernsatz von Layer 7 auf (L7) Metriken für Geräte im Netzwerk, z. B. Anzahl von Antworten, Fehlerzahlen und Verarbeitungszeiten. Nachdem diese Metriken für ein bestimmtes L7-Protokoll aufgezeichnet wurden, werden die Pakete verworfen, wodurch Ressourcen für die weitere Verarbeitung freigegeben werden.

Mit Triggern können Sie:

- Generieren und speichern Sie benutzerdefinierte Metriken im internen Datenspeicher des ExtraHop-Systems. Während das ExtraHop-System beispielsweise keine Informationen darüber sammelt, welcher Benutzeragent eine HTTP-Anfrage generiert hat, können Sie diese Detailtiefe generieren und erfassen, indem Sie einen Auslöser schreiben und die Daten in den Datenspeicher übertragen. Sie können auch benutzerdefinierte Daten anzeigen, die im Datenspeicher gespeichert sind, indem Sie benutzerdefinierte Metrikseiten erstellen und diese Metriken im Metric Explorer anzeigen und Dashboards.
- Generieren und senden Sie Datensätze für die langfristige Speicherung und den Abruf an eine Recordstore.
- Erstellen Sie eine benutzerdefinierte Anwendung, die Metriken für verschiedene Arten von Netzwerkverkehr sammelt, um Informationen mit stufenübergreifender Wirkung zu erfassen. Zum Beispiel, um einen einheitlichen Überblick über den gesamten Netzwerkverkehr im Zusammenhang mit einer Website zu erhalten — von Webtransaktionen bis hin zu DNS Anfragen und Antworten auf Datenbanktransaktionen — Sie können eine Anwendung erstellen, die all diese websitebezogenen Metriken enthält.
- Generieren Sie benutzerdefinierte Metriken und senden Sie die Informationen an Syslog-Nutzer wie Splunk oder an Datenbanken von Drittanbietern wie MongoDB oder Kafka.
- Initiieren Sie eine PCAP, um einzelne Datenflüsse anhand benutzerdefinierter Kriterien Datensatz. Sie können erfasste Flows herunterladen und mit Tools von Drittanbietern verarbeiten. Ihr ExtraHop-System muss für die PCAP lizenziert sein, um auf diese Funktion zugreifen zu können.

Der Zweck dieses Handbuchs besteht darin, Referenzmaterial für das Schreiben von JavaScript-Codeblöcken bereitzustellen, die ausgeführt werden, wenn die Triggerbedingungen erfüllt sind. Die API-Ressourcen auslösen Dieser Abschnitt enthält eine Liste von Themen, die einen umfassenden Überblick über Trigger-Konzepte und -Verfahren bieten.

API-Ressourcen auslösen

Dieser Abschnitt enthält eine Liste von Themen, die Ihnen helfen, sich mit den Konzepten von Auslöser, der Erstellung eines Auslöser und bewährten Methoden vertraut zu machen.

- Auslöser
- Einen Auslöser erstellen
  - Trigger-Einstellungen konfigurieren
  - Schreiben Sie ein Trigger-Skript
- Triggerleistung überwachen
- Leitfaden mit bewährten Methoden für Trigger
- Häufig gestellte Fragen zu Auslöser
- Exemplarische Vorgehensweise: Erstellen Sie einen Auslöser, um benutzerdefinierte Metriken für HTTP 404-Fehler zu sammeln
- Exemplarische Vorgehensweise: Initiieren Sie präzise Paketerfassungen, um Bedingungen ohne Fenster zu analysieren
- Exemplarische Vorgehensweise: Erstellen Sie einen Auslöser, um Antworten auf NTP-Monlist-Anfragen zu überwachen

Datentypen für benutzerdefinierte Metriken

Mit der ExtraHop Trigger API können Sie benutzerdefinierte Metriken erstellen, die Daten über Ihre Umgebung sammeln, die über das hinausgehen, was die integrierten Protokollmetriken bieten.

Sie können benutzerdefinierte Metriken der folgenden Datentypen erstellen:

Graf

Die Anzahl der Metrik Ereignisse, die in einem bestimmten Zeitraum aufgetreten sind. Um beispielsweise Informationen über die Anzahl der HTTP-Anfragen im Laufe der Zeit Datensatz, wählen Sie eine Zählmetrik der obersten Ebene aus. Sie können auch eine Metrik zur Detailzählung auswählen, um Informationen darüber Datensatz, wie oft Klienten hat mit dem IP-Adressschlüssel und einer Ganzzahl, die die Anzahl der Zugriffe als Wert darstellt, auf einen Server zugegriffen.

Schnappschuss

Eine spezielle Art von Zählmetrik, die, wenn sie im Laufe der Zeit abgefragt wird, den neuesten Wert zurückgibt (z. B. hergestellte TCP-Verbindungen).

deutliche

Die geschätzte Anzahl einzigartiger Elemente, die im Laufe der Zeit beobachtet wurden, z. B. die Anzahl der eindeutigen Ports, die SYN-Pakete empfangen haben, wobei eine hohe Zahl auf ein Port-Scanning hindeuten könnte.

Datensatz

Eine statistische Zusammenfassung der Zeitinformationen, z. B. eine Zusammenfassung mit fünf Zahlen: min, 25. Perzentil, Median, 75. Perzentil, max. Um beispielsweise Informationen über die HTTP-Verarbeitungszeit im Laufe der Zeit Datensatz, wählen Sie eine oberste Ebene Datensatz Metrik.

Probenset

Eine statistische Zusammenfassung von Zeitinformationen wie Mittelwert und Standardabweichung. Um beispielsweise Informationen darüber Datensatz, wie lange der Server für die Verarbeitung der einzelnen URI benötigt hat, wählen Sie ein detailliertes Beispielset mit dem URI-Zeichenkettenschlüssel und einer Ganzzahl aus, die die Verarbeitungszeit als Wert darstellt.

max.

Eine spezielle Art von Zählmetrik, die das Maximum beibehält. Um beispielsweise die langsamsten HTTP-Anweisungen im Laufe der Zeit Datensatz, ohne sich auf eine Sitzungstabelle verlassen zu müssen, wählen Sie eine Metrik der obersten Ebene und eine Detail-Max-Metrik aus.

Benutzerdefinierte Metriken werden für die folgenden Quelltypen unterstützt:

- Application
- Device
- Network
- FlowInterface
- FlowNetwork

Weitere Informationen zu den Unterschieden zwischen Top-Level- und Detail-Metriken finden Sie in Häufig gestellte Fragen zu Kennzahlen.

Globale Funktionen

Globale Funktionen können für jedes Ereignis aufgerufen werden.

`cache(key:` ***Schnur*** `, valueFn: () =>` ***Irgendein*** `):` ***Irgendein***

Speichert die angegebenen Parameter in einer Tabelle, um eine effiziente Suche und Rückgabe großer Datensätze zu ermöglichen.

`key:` ***Schnur***

Ein Bezeichner, der den Speicherort des zwischengespeicherten Werts angibt. Ein Schlüssel muss innerhalb eines Auslöser eindeutig sein.

`valueFn: () =>` ***Irgendein***

Eine Funktion mit Nullargumenten, die einen Wert ungleich Null zurückgibt.

Im folgenden Beispiel ist der `cache` Die Methode wird mit großen Datenmengen aufgerufen, die fest in das Triggerskript codiert sind:

```
let storeLookup = cache("storesByNumber", () => ({
    1 : "Newark",
    2 : "Paul",
    3 : "Newark",
    4 : "St Paul"// 620 lines omitted
}));

var storeCity;
var query = HTTP.parseQuery(HTTP.query);

if (query.storeCode) {
   storeCity = storeLookup[parseInt(query.storeCode)];
}
```

Im folgenden Beispiel wird eine Liste bekannter Benutzeragenten in einem JBoss-Trigger normalisiert, bevor sie mit dem beobachteten Benutzeragenten verglichen wird. Der Auslöser konvertiert die Liste in Kleinbuchstaben, entfernt überflüssige Leerzeichen und speichert dann die Einträge im Cache.

```
function jbossUserAgents() {
    return [
        // Add your own user agents here, followed by a comma
        "Gecko-like (Edge 14.0; Windows 10; Silverlight or similar)",
        "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5)
 AppleWebKit/537.36
         (KHTML, like Gecko) Chrome/51.0.2704.79 Safari/537.36",
        "Mozilla/5.0 (Android)"
    ].map(ua => ua.trim().toLowerCase());
}

var badUserAgents = cache("badUserAgents", jbossUserAgents);
```

`commitDetection(type:` ***Schnur*** `, options:` ***Objekt*** `)`

Generiert eine Erkennung auf dem ExtraHop-System.

`type:` ***Schnur***

Ein benutzerdefinierter Typ für die Definition, z. B. `brute_force_attack`. Du kannst Erkennungen abstimmen um mehrere Erkennungen desselben Typs auszublenden. Die Zeichenfolge darf nur Buchstaben, Zahlen und Unterstriche enthalten.

`options:` ***Objekt***

Ein Objekt, das die folgenden Eigenschaften für die Erkennung angibt:

`title:` ***Schnur***

Ein benutzerdefinierter Titel, der die Erkennung identifiziert.

`description:` ***Schnur***

Eine Beschreibung der Erkennung.

`riskScore:` ***Zahl*** | ***null***

Eine optionale Zahl zwischen 1 und 99, die den Risikoscore der Erkennung darstellt.

`participants:` ***Reihe von Objekten***

Ein optionales Array von Teilnehmerobjekten, die der Erkennung zugeordnet sind. Ein Teilnehmerobjekt muss die folgenden Eigenschaften enthalten:

`object:` ***Objekt***

Das Gerät, die Anwendung oder das IP-Adressobjekt, das dem Teilnehmer zugeordnet ist.

`role:` ***Schnur***

Die Rolle des Teilnehmer bei der Erkennung. Die folgenden Werte sind gültig:

- `offender`
- `victim`

`identityKey:` ***Schnur*** | ***null***

Eine eindeutige Kennung, die fortlaufende Erkennungen ermöglicht. Wenn mehrere Erkennungen mit demselben Identitätsschlüssel und Erkennungstyp innerhalb des von der `identityTtl` Eigentum, die Erkennungen werden zu einer einzigen laufenden Erkennung zusammengefasst.

Hinweis: Wenn das ExtraHop-System eine große Anzahl von Erkennungen mit eindeutigen Identitätsschlüsseln generiert, kann das System möglicherweise einige laufende Erkennungen nicht konsolidieren. Das System generiert jedoch nicht mehr als 250 individuelle Erkennungen für einen Auslöser an einem Tag.

`identityTtl:` ***Schnur***

Die Zeitspanne, nach der eine Erkennung generiert wurde, in der doppelte Erkennungen zu einer fortlaufenden Erkennung zusammengefasst werden.

Wenn nach der Generierung einer Erkennung innerhalb des angegebenen Zeitraums eine weitere Erkennung mit demselben Identitätsschlüssel und Erkennungstyp generiert wird, werden die beiden Erkennungen zu einer einzigen laufenden Erkennung zusammengefasst. Jedes Mal, wenn eine Erkennung zu einer laufenden Erkennung konsolidiert wird, wird der Zeitraum zurückgesetzt, und die Erkennung endet erst, wenn der Zeitraum abgelaufen ist. Zum Beispiel, wenn `identityTtl` ist eingestellt auf `day`, und vier doppelte Erkennungen werden jeweils im Abstand von 12 Stunden generiert, die laufende Erkennung erstreckt sich über drei Tage. Die folgenden Zeiträume sind gültig:

- `hour`
- `day`
- `week`

Der Standardzeitraum ist `hour`.

`commitRecord(id:` ***Schnur*** `, record:` ***Objekt*** `):` ***Leere***

Sendet ein benutzerdefiniertes Datensatzobjekt an den konfigurierten Recordstore.

`id:` ***Schnur***

Die ID des Datensatztyps, der erstellt werden soll. Die ID darf nicht mit einer Tilde (~) beginnen.

`record:` ***Objekt***

Ein Objekt, das eine Liste von Eigenschafts- und Wertepaaren enthält, die als benutzerdefinierter Datensatz an den konfigurierten Recordstore gesendet werden sollen.

Die folgenden Eigenschaften werden Datensätzen automatisch hinzugefügt und in den Objekten, die von den integrierten Datensatzaccessoren zurückgegeben werden, nicht dargestellt, wie `HTTP.record`:

- `ex`
- `flowID`
- `client`
- `clientAddr`
- `clientPort`
- `receiver`
- `receiverAddr`
- `receiverPort`
- `sender`
- `senderAddr`
- `senderPort`
- `server`
- `serverAddr`
- `serverPort`
- `timestamp`
- `vlan`

Zum Beispiel, um auf die zuzugreifen `flowID` Eigenschaft in einem HTTP-Datensatz, die Sie einschließen würden `HTTP.record.Flow.id` in deiner Aussage.

Wichtig: Um unerwartete Daten im Datensatz oder eine Ausnahme beim Aufrufen der Methode zu vermeiden, können die oben aufgeführten Eigenschaftsnamen nicht als Eigenschaftsnamen in benutzerdefinierten Datensätzen angegeben werden.

Darüber hinaus darf ein Eigenschaftsname in benutzerdefinierten Datensätzen keines der folgenden Zeichen enthalten:

`.`
: Zeitraum

`:`
: Dickdarm

`[`
: Eckige Klammer

`]`
: Eckige Klammer

Im folgenden Beispiel sind die beiden Eigenschafts- und Wertepaare, die dem hinzugefügt wurden `record` Variablen werden an einen benutzerdefinierten Datensatz übergeben von `commitRecord` funktion:

```
var record = {
   'field1': myfield1,
   'field2': myfield2
```

```
};
commitRecord('record_type_id', record);
```

Bei den meisten Ereignissen können Sie einen integrierten Datensatz festschreiben, der Standardeigenschaften enthält. Zum Beispiel ein integrierter Datensatz wie der `HTTP.record` Objekt kann die Grundlage für einen benutzerdefinierten Datensatz sein.

Der folgende Beispielcode schreibt einen benutzerdefinierten Datensatz fest , der alle integrierten Metriken aus dem `HTTP.record` Objekt und eine zusätzliche Metrik aus dem `HTTP.headers` Eigentum:

```
var record = Object.assign(
   {'server': HTTP.headers.server},
   HTTP.record
);
commitRecord('custom-http-record', record);
```

Sie können bei den folgenden Ereignissen auf ein integriertes Datensatzobjekt zugreifen:

Klasse	Ereignisse
AAA	AAA_REQUEST AAA_RESPONSE
ActiveMQ	ACTIVEMQ_MESSAGE
AJP	AJP_RESPONSE
CIFS	CIFS_RESPONSE
DB	DB_RESPONSE
DHCP	DHCP_REQUEST DHCP_RESPONSE
DICOM	DICOM_REQUEST DICOM_RESPONSE
DNS	DNS_REQUEST DNS_RESPONSE
FIX	FIX_REQUEST FIX_RESPONSE
Flow	FLOW_RECORD
FTP	FTP_RESPONSE
HL7	HL7_RESPONSE
HTTP	HTTP_RESPONSE
IBMMQ	IBMMQ_REQUEST IBMMQ_RESPONSE
ICA	ICA_OPEN ICA_CLOSE ICA_TICK

Klasse	Ereignisse
ICMP	ICMP_MESSAGE
Kerberos	KERBEROS_REQUEST KERBEROS_RESPONSE
LDAP	LDAP_REQUEST LDAP_RESPONSE
Memcache	MEMCACHE_REQUEST MEMCACHE_RESPONSE
Modbus	MODBUS_RESPONSE
MongoDB	MONGODB_REQUEST MONGODB_RESPONSE
MSMQ	MSMQ_MESSAGE
NetFlow	NETFLOW_RECORD
NFS	NFS_RESPONSE
NTLM	NTLM_MESSAGE
POP3	POP3_RESPONSE
RDP	RDP_OPEN RDP_CLOSE RDP_TICK
Redis	REDIS_REQUEST REDIS_RESPONSE
RTCP	RTCP_MESSAGE
RTP	RTP_TICK
SCCP	SCCP_MESSAGE
SFlow	SFLOW_RECORD
SIP	SIP_REQUEST SIP_RESPONSE
SMPP	SMPP_RESPONSE
SMTP	SMTP_RESPONSE
SSH	SSH_OPEN SSH_CLOSE SSH_TICK
SSL	SSL_ALERT SSL_OPEN SSL_CLOSE

Klasse	Ereignisse
	SSL_HEARTBEAT SSL_RENEGOTIATE
Telnet	TELNET_MESSAGE

`debug(message:` ***Schnur*** `):` ***Leere***

Schreibt an die Debug-Log wenn Debugging aktiviert ist. Die maximale Nachrichtengröße beträgt 2048 Byte. Nachrichten, die länger als 2048 Byte sind, werden gekürzt.

`getTimestamp():` ***Zahl***

Gibt den Zeitstempel des Paket zurück, das das Triggerereignis ausgelöst hat, ausgedrückt in Millisekunden mit Mikrosekunden als Bruchteil nach dem Dezimaltrennzeichen.

`log(message:` ***Schnur*** `):` ***Leere***

Schreibt in das Debug-Log, unabhängig davon, ob das Debugging aktiviert ist.

Mehrere Aufrufe von Debug- und Log-Anweisungen, in denen die Nachricht denselben Wert hat, werden alle 30 Sekunden angezeigt.

Das Limit für Debug-Logeinträge liegt bei 2048 Byte. Um größere Einträge zu protokollieren, siehe Remote.Syslog .

`md5(message:` ***Schnur*** | ***Puffer*** `):` ***Schnur***

Hasht die UTF-8-Darstellung der angegebenen Nachricht Puffer Objekt oder Zeichenfolge und gibt die MD5-Summe der Zeichenfolge zurück.

`sha1(message:` ***Schnur*** | ***Puffer*** `):` ***Schnur***

Hasht die UTF-8-Darstellung der angegebenen Nachricht Puffer Objekt oder Zeichenfolge und gibt die SHA-1-Summe der Zeichenfolge zurück.

`sha256(message:` ***Schnur*** | ***Puffer*** `):` ***Schnur***

Hasht die UTF-8-Darstellung der angegebenen Nachricht Puffer Objekt oder Zeichenfolge und gibt die SHA-256-Summe der Zeichenfolge zurück.

`sha512(message:` ***Schnur*** | ***Puffer*** `):` ***Schnur***

Hasht die UTF-8-Darstellung der angegebenen Nachricht Puffer Objekt oder Zeichenfolge und gibt die SHA-512-Summe der Zeichenfolge zurück.

`uuid():` ***Schnur***

Gibt einen zufälligen Universally Unique Identifier (UUID) der Version 4 zurück.

Kurse für allgemeine Zwecke

Die Trigger-API-Klassen in diesem Abschnitt bieten Funktionen, die für alle Ereignisse allgemein anwendbar sind.

Klasse	Beschreibung
Application	Ermöglicht das Erstellen neuer Anwendungen und fügt benutzerdefinierte Metriken auf Anwendungsebene hinzu.
Puffer	Ermöglicht den Zugriff auf Pufferinhalte.
Detection	Ermöglicht das Abrufen von Informationen über Erkennungen auf dem ExtraHop-System.
Device	Ermöglicht das Abrufen von Geräteattributen und das Hinzufügen benutzerdefinierter Metriken auf Geräteebene.
Discover	Ermöglicht Ihnen den Zugriff auf neu entdeckte Geräte und Anwendungen.
Flow	Flow bezieht sich auf eine Konversation zwischen zwei Endpunkten über ein Protokoll wie TCP, UDP oder ICMP. Die Flow-Klasse bietet Zugriff auf Elemente dieser Konversationen, wie z. B. Endpunkt-IP-Adressen und Alter des Flows. Die Flow-Klasse enthält auch einen Flow-Speicher, mit dem Objekte im selben Flow von der Anfrage zur Antwort übergeben werden können.
FlowInterface	Ermöglicht das Abrufen von Flow-Schnittstellenattributen und das Hinzufügen benutzerdefinierter Metriken auf Schnittstellenebene.
FlowNetwork	Ermöglicht das Abrufen von Flow-Netzwerkattributen und das Hinzufügen benutzerdefinierter Metriken auf Flow-Netzwerkebene.
GeoIP	Ermöglicht es Ihnen, den ungefähren Standort einer bestimmten IP-Adresse auf Landes- oder Stadtebene abzurufen.
IPAddress	Ermöglicht das Abrufen von IP-Adressattributen.
Network	Ermöglicht das Hinzufügen benutzerdefinierter Metriken auf globaler Ebene.
Session	Ermöglicht den Zugriff auf die Sitzungstabelle, die die Koordination mehrerer unabhängig voneinander ausgeführter Trigger unterstützt.
System	Ermöglicht den Zugriff auf Eigenschaften, die das ExtraHop-System identifizieren, auf dem ein Auslöser ausgeführt wird.

Klasse	Beschreibung
ThreatIntel	Ermöglicht es Ihnen zu sehen, ob eine IP-Adresse, ein Hostname oder eine URI verdächtig ist.
Trigger	Ermöglicht den Zugriff auf Details zu einem laufenden Auslöser.
VLAN	Ermöglicht den Zugriff auf Informationen über ein VLAN im Netzwerk.

`Application`

Das `Application` Mit dieser Klasse können Sie Metriken für mehrere Arten von Netzwerkverkehr erfassen, um Informationen zu erfassen, die sich auf mehrere Ebenen auswirken. Wenn Sie beispielsweise eine einheitliche Ansicht des gesamten mit einer Website verbundenen Netzwerkverkehrs wünschen — von Webtransaktionen über DNS-Anfragen bis hin zu Antworten auf Datenbanktransaktionen —, können Sie einen Auslöser schreiben, um eine benutzerdefinierte Anwendung zu erstellen, die all diese verwandten Metriken enthält. Das `Application` Mit der Klasse können Sie auch benutzerdefinierte Metriken erstellen und die Metrikdaten an Anwendungen übertragen. Anwendungen können nur durch Trigger erstellt und definiert werden.

Instanzmethoden

Die Methoden in diesem Abschnitt können nicht direkt auf dem aufgerufen werden `Application` Klasse. Sie können diese Methoden nur für bestimmte Application-Klasseninstanzen aufrufen. Zum Beispiel ist die folgende Aussage gültig:

```
Application("sampleApp").metricAddCount("responses", 1);
```

Die folgende Aussage ist jedoch ungültig:

```
Application.metricAddCount("responses", 1);
```

`commit(id:` ***Schnur*** `):` ***Leere***

Erstellt eine Anwendung, überträgt die mit dem Ereignis verknüpften integrierten Metriken an die Anwendung und fügt die Anwendung zu allen integrierten oder benutzerdefinierten Datensätzen hinzu, die während des Ereignisses festgeschrieben wurden.

Die Anwendungs-ID muss eine Zeichenfolge sein. Bei integrierten Anwendungsmetriken werden die Metriken nur einmal festgeschrieben, auch wenn `commit()` Die Methode wird mehrmals für dasselbe Ereignis aufgerufen.

Die folgende Anweisung erstellt eine Anwendung mit dem Namen „myApp“ und überträgt integrierte Metriken an die Anwendung:

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

Wenn Sie benutzerdefinierte Metriken an eine Anwendung übertragen möchten, können Sie die Anwendung erstellen, ohne die `commit()` Methode. Wenn die Anwendung beispielsweise noch nicht existiert, erstellt die folgende Anweisung die Anwendung und überträgt die benutzerdefinierte Metrik an die Anwendung:

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

Du kannst das anrufen `Application.commit` Methode nur bei den folgenden Ereignissen:

Metrische Typen	Ereignis
AAA	AAA_REQUEST -und- AAA_RESPONSE
AJP	AJP_RESPONSE
CIFS	CIFS_RESPONSE
DB	DB_RESPONSE
DHCP	DHCP_REQUEST -und- DHCP_RESPONSE
DNS	DNS_REQUEST -und- DNS_RESPONSE
FIX	FIX_REQUEST -und- FIX_RESPONSE
FTP	FTP_RESPONSE
HTTP	HTTP_RESPONSE
IBMMQ	IBMMQ_REQUEST -und- IBMMQ_RESPONSE
ICA	ICA_TICK -und- ICA_CLOSE
Kerberos	KERBEROS_REQUEST -und- KERBEROS_RESPONSE
LDAP	LDAP_REQUEST -und- LDAP_RESPONSE
Memcache	MEMCACHE_REQUEST -und- MEMCACHE_RESPONSE
Modbus	MODBUS_RESPONSE
MongoDB	MONGODB_REQUEST -und- MONGODB_RESPONSE
NAS	CIFS_RESPONSE -und/oder- NFS_RESPONSE
NetFlow	NETFLOW_RECORD Beachten Sie, dass das Commit nicht erfolgt, wenn Enterprise-IDs im NetFlow-Datensatz vorhanden sind.
NFS	NFS_RESPONSE
RDP	RDP_TICK
Redis	REDIS_REQUEST -und- REDIS_RESPONSE
RPC	RPC_REQUEST -und- RPC_RESPONSE
RTP	RTP_TICK
RTCP	RTCP_MESSAGE
SCCP	SCCP_MESSAGE
SIP	SIP_REQUEST -und- SIP_RESPONSE
SFlow	SFLOW_RECORD
SMTP	SMTP_RESPONSE
SSH	SSH_CLOSE -und- SSH_TICK
SSL	SSL_RECORD -und- SSL_CLOSE

Metrische Typen	Ereignis
WebSocket	WEBSOCKET_OPEN, WEBSOCKET_CLOSE, und WEBSOCKET_MESSAGE

`metricAddCount(metric_name:` ***Schnur*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Metrik zählen. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der Metrik für die Zählung der obersten Ebene.

`count:` ***Zahl***

Der Inkrementwert. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. Ein `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgende Eigenschaft enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailCount(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Metrik zählen anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der Metrik für die Detailanzahl.

`key:` ***Schnur*** `|` ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`count:` ***Zahl***

Der Inkrementwert. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. Ein `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgende Eigenschaft enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDataset(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Datensatz-Metrik. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der Datensatzmetrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`freq:` ***Zahl***

Eine Option, die es Ihnen ermöglicht, mehrere Vorkommen bestimmter Werte im Datensatz gleichzeitig aufzuzeichnen, wenn sie auf die Anzahl von Vorkommen gesetzt ist, die durch `val` Parameter. Wenn kein Wert angegeben ist, ist der Standardwert 1.

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailDataset(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Datensatz-Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der Metrik für die Detailanzahl.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`freq:` ***Zahl***

Eine Option, die es Ihnen ermöglicht, mehrere Vorkommen bestimmter Werte im Datensatz gleichzeitig aufzuzeichnen, wenn sie auf die Anzahl von Vorkommen gesetzt ist, die durch `val` Parameter. Wenn kein Wert angegeben ist, ist der Standardwert 1.

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDistinct(metric_name:` ***Schnur*** `, item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse*** `:void`

Erstellt ein benutzerdefiniertes oberste Ebene unterschiedliche Zählmetrik. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der Metrik für die eindeutige Anzahl auf oberster Ebene.

`item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse***

Der Wert, der in das Set aufgenommen werden soll. Der Wert wird in eine Zeichenfolge umgewandelt, bevor er in den Satz aufgenommen wird.

`metricAddDetailDistinct(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse*** `:void`

Erstellt ein benutzerdefiniertes Detail unterschiedliche Zählmetrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der detaillierten Metrik für unterschiedliche Zählungen.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse***

Der Wert, der in das Set aufgenommen werden soll. Der Wert wird in eine Zeichenfolge umgewandelt, bevor er in den Satz aufgenommen wird.

`metricAddMax(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene maximale Metrik. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der maximalen Metrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailMax(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail maximale Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der maximalen Detailmetrik.

`key:` ***Schnur | IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddSampleset(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Stichprobensatz, Metrik. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der Sampleset-Metrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailSampleset(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Stichprobensatz, Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der Detail-Sampleset-Metrik.

`key:` ***Schnur | IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddSnap(metric_name:` ***Schnur*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Snapshot-Metrik. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der Snapshot-Metrik der obersten Ebene.

`count:` ***Zahl***

Der beobachtete Wert, z. B. aktuell hergestellte Verbindungen. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailSnap(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Snapshot-Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Anwendung.

`metric_name:` ***Schnur***

Der Name der Detail-Sampleset-Metrik.

`key:` ***Schnur*** `|` ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`count:` ***Zahl***

Der beobachtete Wert, z. B. aktuell hergestellte Verbindungen. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`toString()`: ***Schnur***

Gibt das Application-Objekt als Zeichenfolge im folgenden Format zurück:

```
[object Application <application_id>]
```

Instanzeigenschaften

`id`: ***Schnur***

Die eindeutige ID der Anwendung, wie sie im ExtraHop-System auf der Seite für diese Anwendung angezeigt wird.

Beispiele für Trigger

- Beispiel: Erstellen Sie einen Anwendungscontainer

Puffer

Das `Buffer` Klasse bietet Zugriff auf Binärdaten.

Ein Puffer ist ein Objekt mit den Eigenschaften eines Arrays. Jedes Element im Array ist eine Zahl zwischen 0 und 255, die ein Byte darstellt. Jedes Pufferobjekt hat eine Längeneigenschaft (die Anzahl der Elemente in einem Array) und einen Operator mit eckigen Klammern.

Verschlüsselte Nutzdaten werden für die TCP- und UDP-Nutzlastanalyse nicht entschlüsselt.

`UDP_PAYLOAD` benötigt aber eine passende Zeichenfolge `TCP_PAYLOAD` tut nicht. Wenn Sie keine passende Zeichenfolge angeben für `TCP_PAYLOAD`, der Auslöser wird einmal nach den ersten N Byte der Nutzlast ausgeführt.

Methoden

`Buffer(string:` ***Schnur*** `| format:` ***Schnur*** `)`

Konstruktor für die Buffer-Klasse, der eine codierte Zeichenfolge in ein Buffer-Objekt dekodiert. Die folgenden Parameter sind erforderlich:

`string:` ***Schnur***

Die codierte Zeichenfolge.

`format:` ***Schnur***

Das Format, mit dem das Zeichenkettenargument codiert ist. Die folgenden Kodierungsformate sind gültig:

- `base64`
- `base64url`

Instanzmethoden

`decode(type:` ***Schnur*** `):` ***Schnur***

Interpretiert den Inhalt des Puffers und gibt eine Zeichenfolge mit einer der folgenden Optionen zurück:

- `utf-8`
- `utf-16`
- `ucs2`
- `hex`

`equals(buffer:` ***Puffer*** `):` ***Boolesch***

Führt einen Gleichheitstest zwischen Buffer-Objekten durch, wobei `buffer` ist das Objekt, mit dem verglichen werden soll.

`slice(start:` ***Zahl*** `, end:` ***Zahl*** `):` ***Puffer***

Gibt die angegebenen Bytes in einem Puffer als neuen Puffer zurück. Bytes werden ab dem angegebenen Startargument ausgewählt und enden am Endargument (aber nicht einschließlich).

`start:` ***Zahl***

Ganzzahl, die angibt, wo die Auswahl beginnen soll. Geben Sie negative Zahlen an, um am Ende eines Puffers auszuwählen. Das ist nullbasiert.

`end:` ***Zahl***

Optionale Ganzzahl, die angibt, wo die Auswahl beendet werden soll. Wenn nicht angegeben, werden alle Elemente von der Startposition bis zum Ende des Puffers ausgewählt. Geben Sie negative Zahlen an, um am Ende eines Puffers auszuwählen. Das ist nullbasiert.

`toString(format:` ***Schnur*** `):` ***Schnur***

Konvertiert den Puffer in eine Zeichenfolge. Der folgende Parameter ist optional:

`format:` ***Schnur***

Das Format, mit dem die Zeichenfolge codiert werden soll. Wenn keine Kodierung angegeben ist, ist die Zeichenfolge uncodiert. Die folgenden Kodierungsformate sind gültig:

- `base64`
- `base64url`
- `hex`

`unpack(format:` ***Schnur*** `, offset:` ***Zahl*** `):` ***Reihe***

Verarbeitet binäre Daten oder Daten mit fester Breite aus einem beliebigen Pufferobjekt, z. B. eines, das von `HTTP.payload`, `Flow.client.payload`, oder `Flow.sender.payload`, entsprechend der angegebenen Formatzeichenfolge und optional am angegebenen Offset.

Gibt ein JavaScript-Array zurück, das ein oder mehrere entpackte Felder enthält und die absolute Payload-Byteposition +1 des letzten Bytes im entpackten Objekt enthält. Der Bytewert kann in weiteren Aufrufen zum Entpacken eines Puffers als Offset angegeben werden.

Hinweis: Das `buffer.unpack` Die Methode interpretiert Bytes standardmäßig in Big-Endian-Reihenfolge. Um Bytes in Little-Endian-Reihenfolge zu interpretieren, stellen Sie der Formatzeichenfolge ein Kleiner-als-Zeichen voran (<).

- Das Format muss nicht den gesamten Puffer verbrauchen.
- Null-Bytes sind in entpackten Zeichenketten nicht enthalten. Zum Beispiel: `buf.unpack('4s')[0] - > 'example'`.
- Das Z-Formatzeichen steht für nullterminierte Zeichenketten variabler Länge. Wenn das letzte Feld z ist, wird die Zeichenfolge erzeugt, unabhängig davon, ob das Nullzeichen vorhanden ist oder nicht.
- Eine Ausnahme wird ausgelöst, wenn nicht alle Felder entpackt werden können, weil der Puffer nicht genügend Daten enthält.

Die folgende Tabelle zeigt die unterstützten Puffer-String-Formate:

Formatieren	Typ C	JavaScript-Typ	Standardgröße
x	pad type	kein Wert	
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

Formatieren	Typ C	JavaScript-Typ	Standardgröße
H	unsignierter Kurzfilm	number	2
h	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	

Instanzeigenschaften

`length:` ***Zahl***

Die Anzahl der Byte im Puffer.

Beispiele für Trigger

- Beispiel: NTP mit universeller Nutzlastanalyse analysieren
- Beispiel: Syslog über TCP mit universeller Nutzlastanalyse analysieren

`Detection`

Die `Detection` class ermöglicht es Ihnen, Informationen über Erkennungen auf dem ExtraHop-System abzurufen.

Hinweis: Erkennungen durch maschinelles Lernen erfordern eine Verbindung zu ExtraHop Cloud Services.

Ereignisse

`DETECTION_UPDATE`

Wird ausgeführt, wenn eine Erkennung auf dem ExtraHop-System erstellt oder aktualisiert wird.

Wichtig: Dieses Ereignis wird für alle Erkennungen ausgeführt, unabhängig davon, welcher Modulzugriff dem Benutzer gewährt wird, der den Auslöser erstellt. Trigger, die von Benutzern mit NPM-Modulzugriff erstellt wurden, laufen beispielsweise auf `DETECTION_UPDATE` Ereignisse sowohl für Sicherheits- als auch für Leistungserkennungen.

Hinweis: Dieses Ereignis wird nicht ausgeführt, wenn der Status eines Erkennungstickets aktualisiert wird. Wenn Sie beispielsweise einen Erkennungsbeauftragten ändern, wird das DETECTION_UPDATE-Ereignis nicht ausgeführt. Dieses Ereignis wird auch nicht für versteckte Erkennungen ausgeführt.

Hinweis: Sie können Trigger, die nur bei diesem Ereignis ausgeführt werden, nicht bestimmten Geräten oder Gerätegruppen zuweisen. Trigger, die bei diesem Ereignis ausgeführt werden, werden immer dann ausgeführt, wenn dieses Ereignis eintritt.

Eigenschaften

`applianceId:` ***Zahl***

Wenn auf einem aufgerufen Konsole, gibt die ID des angeschlossenen Sensor zurück, an dem die Erkennung stattgefunden hat. Wenn es über einen Sensor aufgerufen wird, kehrt zurück `0`.

`assignee:` ***Schnur***

Der Bevollmächtigte des Tickets, das mit der Erkennung verknüpft ist.

`categories:` ***Reihe von Zeichenketten***

Die Liste der Kategorien, zu denen die Erkennung gehört.

`description:` ***Schnur***

Die Beschreibung der Erkennung.

Hinweis: Es ist oft einfacher, Informationen über eine Erkennung aus dem `Detection.properties` Eigenschaft als das Parsen der `Detection.description` Text. Weitere Informationen finden Sie in der `Detection.properties` Beschreibung.

Die folgende Tabelle zeigt gängige Markdown-Formate, die Sie in die Beschreibung aufnehmen können:

Format	Beschreibung	Beispiel
Überschriften	Platzieren Sie ein Zahlenzeichen (#) vor Ihrem Text, um Überschriften zu formatieren. Die Ebene der Überschrift wird durch die Anzahl der Nummernzeichen bestimmt.	`####Example H4 heading`
Ungeordnete Listen	Platzieren Sie ein einzelnes Sternchen (*) vor Ihrem Text.	`* First example * Second example`
Geordnete Listen	Platzieren Sie eine einzelne Zahl und einen Punkt (1.) vor Ihrem Text.	`1. First example 2. Second example`
Mutig	Platzieren Sie doppelte Sternchen vor und nach Ihrem Text.	`**bold text**`
Kursiv	Platzieren Sie vor und nach Ihrem Text einen Unterstrich.	`_italicized text_`
Hyperlinks	Platzieren Sie den Linktext in Klammern vor der URL in Klammern. Oder geben Sie Ihre URL ein. Links zu externen Websites werden in einem neuen Browser-Tab geöffnet. Links innerhalb des ExtraHop-Systems, wie z. B. Dashboards,	`[Visit our home page](https://www.extrahop.com)` `https://www.extrahop.com`

Format	Beschreibung	Beispiel
	werden im aktuellen Browser-Tab geöffnet.	
Zitate blockieren	Platzieren Sie eine rechtwinklige Klammer und ein Leerzeichen vor Ihrem Text.	`On the ExtraHop website:` `> Access the live demo and review case studies.`
Emojis	Kopieren Sie ein Emoji-Bild und fügen Sie es in das Textfeld ein. Sehen Sie die Unicode-Emoji-Diagramm Website für Bilder. Die Markdown-Syntax unterstützt keine Emoji-Shortcodes.	

`endTime:` ***Zahl***

Die Zeit, in der die Erkennung endete, ausgedrückt in Millisekunden seit der Epoche.

`id:` ***Zahl***

Die eindeutige Kennung für die Erkennung.

`isCustom:` ***Boolescher Wert***

Der Wert ist `true` wenn es sich bei der Erkennung um eine benutzerdefinierte Erkennung handelt, die durch einen Auslöser generiert wird.

`mitreCategories:` ***Reihe von Objekten***

Eine Reihe von Objekten, die die mit der Erkennung verbundenen MITRE-Techniken und -Taktiken enthalten. Jedes Objekt enthält die folgenden Eigenschaften:

`id`

Die ID der MITRE-Technik oder -Taktik.

`name`

Der Name der MITRE-Technik oder -Taktik.

`url`

Die Webadresse der Technik oder Taktik auf der MITRE-Website.

`participants:` ***Reihe von Objekten***

Eine Reihe von Teilnehmerobjekten, die mit der Erkennung verknüpft sind. Ein Teilnehmerobjekt enthält die folgenden Eigenschaften:

`object:` ***Objekt***

Das Gerät-, Anwendungs- oder IP-Adressobjekt, das dem Teilnehmer zugeordnet ist.

`id:` ***Zahl***

Die ID des Teilnehmer.

`role:` ***Schnur***

Die Rolle des Teilnehmer bei der Erkennung. Die folgenden Werte sind gültig:

- `offender`
- `victim`

`properties:` ***Objekt***

Ein Objekt, das die Eigenschaften der Erkennung enthält. Nur integrierte Erkennungstypen enthalten Erkennungseigenschaften. Der Erkennungstyp bestimmt, welche Eigenschaften verfügbar sind.

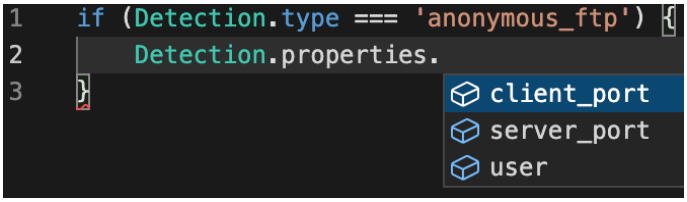
Die Feldnamen des Objekts sind die Namen der Erkennungseigenschaften. Der Erkennungstyp Anonym FTP Auth Enabled umfasst beispielsweise `client_port` Eigenschaft, auf die Sie mit dem folgenden Code zugreifen können:

```
Detection.properties.client_port
```

Um die Namen der Erkennungseigenschaften anzuzeigen, zeigen Sie Erkennungstypen mit dem `GET /detections/formats` Betrieb in der ExtraHop REST API.

Hinweis: Im Trigger-Editor können Sie gültige Erkennungseigenschaften mit der Autocomplete-Funktion anzeigen, wenn Sie Logik einbeziehen, die den Erkennungstyp bestimmt. Wenn der Auslöser beispielsweise den folgenden Code enthält und Sie nach „Eigenschaften“ einen Punkt eingeben, zeigt der Trigger-Editor die gültigen Eigenschaften für die Erkennung Anonymous FTP Auth Enabled an:

```
if (Detection.type === 'anonymous_ftp') {
    Detection.properties
}
```



`resolution:` ***Schnur***

Die Auflösung des Tickets, das mit der Erkennung verknüpft ist. Gültige Werte sind `action_taken` und `no_action_taken`.

`riskScore:` ***Nummer*** | ***null***

Die Risikoscore der Erkennung.

`startTime:` ***Zahl***

Die Zeit, zu der die Erkennung begann, ausgedrückt in Millisekunden seit der Epoche.

`status:` ***Schnur***

Der Status des Tickets, das mit der Erkennung verknüpft ist. Gültige Werte sind `acknowledged`, `new`, `in_progress`, und `closed`.

`ticketId:` ***Schnur***

Die ID des Tickets, das mit der Erkennung verknüpft ist.

`title:` ***Schnur***

Der Titel der Erkennung.

`type:` ***Schnur***

Die Art der Erkennung. Bei benutzerdefinierten Erkennungen wird der benutzerdefinierten Zeichenfolge „custom“ vorangestellt . Wenn Sie beispielsweise angeben `brute_force_attack` in der `commitDetection` Funktion, der Erkennungstyp ist `custom.brute_force_attack`.

`updateTime:` ***Zahl***

Das letzte Mal, dass die Erkennung aktualisiert wurde, ausgedrückt in Millisekunden seit der Epoche.

`Device`

Das `Device` Mit dieser Klasse können Sie Geräteattribute abrufen und benutzerdefinierte Metriken auf Geräteebene hinzufügen.

Methoden

`Device(id:` ***Schnur*** `)`

Konstruktor für das Device-Objekt, das einen Parameter akzeptiert, bei dem es sich um eine eindeutige 16-stellige Zeichenketten-ID handelt.

Wenn eine ID von einem vorhandenen Device-Objekt bereitgestellt wird, erstellt der Konstruktor eine Kopie dieses Objekts mit allen Objekteigenschaften, wie im folgenden Beispiel gezeigt:

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

Metriken , die über eine an ein Geräteobjekt übergeben wurden `metricAdd*` Funktionen werden im Datenspeicher gespeichert

`lookupByIP(addr:` ***IP-Adresse*** `|` ***Schnur*** `, vlan:` ***Zahl*** `):` ***Gerät***

Gibt das L3-Gerät zurück, das der angegebenen IP-Adresse und VLAN-ID entspricht. Retouren `null` wenn kein Treffer gefunden wird.

`addr:` ***IP-Adresse*** `|` ***Schnur***

Die IP-Adresse für das Gerät. Die IP-Adresse kann angegeben werden als IPAddress Objekt oder als Zeichenfolge.

`vlan:` ***Nummer***

Die VLAN-ID für das Gerät. Gibt einen Standardwert von `0` wenn keine VLAN-ID angegeben wird oder wenn der Wert der `devices_across_vlans` Die Einstellungen sind eingestellt auf `true` in der Konfigurationsdatei ausführen.

`lookupByMAC(addr:` ***Schnur*** `, vlan:` ***Zahl*** `):` ***Gerät***

Gibt das L2-Gerät zurück, das der angegebenen MAC-Adresse und VLAN-ID entspricht. Retouren `null` wenn kein Treffer gefunden wird.

`addr:` ***Schnur***

Die MAC-Adresse für das Gerät.

`vlan:` ***Zahl***

Die VLAN-ID für das Gerät. Gibt einen Standardwert von `0` wenn keine VLAN-ID angegeben wird oder wenn der Wert der `devices_across_vlans` Die Einstellungen sind eingestellt auf `true` in der Konfigurationsdatei ausführen.

`toString():` ***Schnur***

Gibt das Device-Objekt als Zeichenfolge im folgenden Format zurück:

```
[object Device <discovery_id>]
```

Instanzmethoden

Die in diesem Abschnitt beschriebenen Methoden sind nur auf Instanzen der Device-Klasse vorhanden. Mit den meisten Methoden können Sie benutzerdefinierte Messwerte auf Geräteebene erstellen, wie im folgenden Beispiel gezeigt:

```
Flow.server.device.metricAddCount("slow_rsp", 1);
```

Hinweis: Ein Gerät kann manchmal als Client und manchmal als Server in einem Fluss agieren.

- Rufen Sie eine Methode auf als `Device.metricAdd*` um Daten für beide Geräterollen zu sammeln.
- Rufen Sie eine Methode auf als `Flow.client.device.metricAdd*` um nur Daten für die Client-Rolle zu sammeln, unabhängig davon, ob der Auslöser dem Client oder dem Server zugewiesen ist.

- Rufen Sie eine Methode auf als `Flow.server.device.metricAdd*` um nur Daten für die Serverrolle zu sammeln, unabhängig davon, ob der Auslöser dem Client oder dem Server zugewiesen ist.

`equals(device:` ***Gerät*** `):` ***Boolesch***

Führt einen Gleichheitstest zwischen Device-Objekten durch, wobei `device` ist das Objekt, mit dem verglichen werden soll.

`metricAddCount(metric_name:` ***Schnur*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Metrik zählen. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der Metrik für die Zählung der obersten Ebene.

`count:` ***Zahl***

Der Inkrementwert. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. Ein `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgende Eigenschaft enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailCount(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Metrik zählen anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der Metrik für die Detailzählung.

`key:` ***Schnur | IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`count:` ***Zahl***

Der Inkrementwert. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. Ein `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgende Eigenschaft enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDataset(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Datensatz-Metrik. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der Datensatzmetrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`freq:` ***Zahl***

Eine Option, die es Ihnen ermöglicht, mehrere Vorkommen bestimmter Werte im Datensatz gleichzeitig aufzuzeichnen, wenn sie auf die Anzahl von Vorkommen gesetzt ist, die durch `val` Parameter. Wenn kein Wert angegeben ist, ist der Standardwert 1.

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailDataset(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Datensatz-Metrik anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der Metrik für die Detailzählung.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`freq:` ***Zahl***

Eine Option, die es Ihnen ermöglicht, mehrere Vorkommen bestimmter Werte im Datensatz gleichzeitig aufzuzeichnen, wenn sie auf die Anzahl von Vorkommen gesetzt ist, die durch `val` Parameter. Wenn kein Wert angegeben ist, ist der Standardwert 1.

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDistinct(metric_name:` ***Schnur*** `, item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse*** `:void`

Erstellt ein benutzerdefiniertes oberste Ebene unterschiedliche Zählmetrik. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der Metrik für die eindeutige Anzahl auf oberster Ebene.

`item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse***

Der Wert, der in das Set aufgenommen werden soll. Der Wert wird in eine Zeichenfolge umgewandelt, bevor er in den Satz aufgenommen wird.

`metricAddDetailDistinct(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse*** `:void`

Erstellt ein benutzerdefiniertes Detail unterschiedliche Zählmetrik anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der detaillierten Metrik für unterschiedliche Zählungen.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse***

Der Wert, der in das Set aufgenommen werden soll. Der Wert wird in eine Zeichenfolge umgewandelt, bevor er in den Satz aufgenommen wird.

`metricAddMax(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene maximale Metrik. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der maximalen Metrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailMax(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail maximale Metrik anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der maximalen Detailmetrik.

`key:` ***Schnur*** `|` ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddSampleset(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Stichprobensatz, Metrik. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der Sampleset-Metrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailSampleset(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Stichprobensatz, Metrik anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der Detail-Sampleset-Metrik.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddSnap(metric_name:` ***Schnur*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Snapshot-Metrik. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der Snapshot-Metrik der obersten Ebene.

`count:` ***Zahl***

Der beobachtete Wert, z. B. aktuell hergestellte Verbindungen. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailSnap(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Snapshot-Metrik anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrikdaten an das angegebene Gerät.

`metric_name:` ***Schnur***

Der Name der Detail-Sampleset-Metrik.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`count:` ***Zahl***

Der beobachtete Wert, z. B. aktuell hergestellte Verbindungen. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

Instanzeigenschaften

Die folgenden Eigenschaften ermöglichen das Abrufen von Geräteattributen und sind nur auf Instanzen der Device-Klasse vorhanden.

`cdpName:` ***Schnur***

Der dem Gerät zugeordnete CDP-Name, falls vorhanden.

`dhcpName:` ***Schnur***

Das DHCP Name, der dem Gerät zugeordnet ist, falls vorhanden.

`discoverTime:` ***Zahl***

Der Zeitpunkt, an dem das Gerät durch den Erfassungsvorgang zum letzten Mal entdeckt wurde (nicht die ursprüngliche Entdeckungszeit), ausgedrückt in Millisekunden seit der Epoche (1. Januar 1970). Zuvor erkannte Geräte können vom Erfassungsprozess wiedererkannt werden, wenn sie inaktiv sind und später wieder aktiv werden, oder wenn der Erfassungsvorgang neu gestartet wird.

Informationen darüber, wie ein Auslöser nur bei der ersten Erkennung eines Gerät ausgeführt werden soll, finden Sie unter `NEW_DEVICE` besprochene Ereignis in der Discover Klasse.

`dnsNames:` ***Reihe***

Ein String-Array, das die mit dem Gerät verknüpften DNS-Namen auflistet, falls vorhanden.

`hasTrigger:` ***Boolesch***

Der Wert ist `true` wenn ein Auslöser, der dem Device-Objekt zugewiesen ist, gerade läuft.

Wenn der Auslöser bei einem Ereignis ausgeführt wird, das mit einem verknüpft ist Flow Objekt, das `hasTrigger` Immobilienwert ist `true` auf mindestens einem der Device-Objekte im Fluss.

Das `hasTrigger` Diese Eigenschaft ist nützlich, um Geräterollen zu unterscheiden. Wenn beispielsweise einer Gruppe von Proxyservern ein Auslöser zugewiesen ist, können Sie leicht feststellen, ob ein Gerät als Client oder Server fungiert, anstatt nach IP-Adressen oder Geräte-IDs zu suchen, wie im folgenden Beispiel:

```
//Event: HTTP_REQUEST
if (Flow.server.device.hasTrigger) {
    // Incoming request
} else {
    // Outgoing request
}
```

`hwaddr:` ***Schnur***

Die MAC-Adresse des Gerät, falls vorhanden.

`id:` ***Schnur***

Die 16-stellige eindeutige ID des Geräts, wie sie im ExtraHop-System auf der Seite für dieses Gerät angezeigt wird.

`ipaddrs:` ***Reihe***

Eine Reihe von IPAddress Objekte, die die bekannten IP-Adressen des Geräts darstellen. Für L3 Geräte, das Array enthält immer eine IP-Adresse.

`isGateway:` ***Boolesch***

Der Wert ist `true` wenn das Gerät ein Gateway ist.

`isL3:` ***Boolesch***

Der Wert ist `true` wenn das Gerät ein L3 Gerät für Kinder.

Wichtig: Wenn Sie das ExtraHop-System nicht aktiviert haben Geräte anhand der IP-Adresse erkennen, ist die Eigenschaft isL3 immer auf False gesetzt, da das System nicht zwischen untergeordneten L3-Geräten und übergeordneten L2-Geräten unterscheidet.

`netbiosName:` ***Schnur***

Der NetBIOS-Name, der dem Gerät zugeordnet ist, falls vorhanden.

`vlanId:` ***Zahl***

Die VLAN-ID für das Gerät.

Beispiele für Trigger

- Beispiel: Überwachen Sie CIFS-Aktionen auf Geräten
- Beispiel: HTTP-Antworten auf 500-Ebene nach Kunden-ID und URI verfolgen
- Beispiel: Antwortmetriken für Datenbankabfragen sammeln
- Beispiel: Erkannte Gerätedaten an einen Remote-Syslog-Server senden
- Beispiel: Zugriff auf HTTP-Header-Attribute
- Beispiel: Memcache-Treffer und Fehlschläge aufzeichnen
- Beispiel: Memcache-Schlüssel analysieren
- Beispiel: Analysieren von benutzerdefinierten PoS-Nachrichten mit universeller Nutzlastanalyse
- Beispiel: Metriken zum Metric Cycle Store hinzufügen

`Discover`

Die `Discover` Mit dieser Klasse können Sie Informationen über neu entdeckte Geräte und Anwendungen abrufen.

Ereignisse

`NEW_APPLICATION`

Wird ausgeführt, wenn eine Anwendung zum ersten Mal erkannt wird. Dieses Ereignis verbraucht Erfassungsressourcen.

Hinweis Sie können Trigger, die nur bei diesem Ereignis ausgeführt werden, nicht bestimmten Geräten oder Gerätegruppen zuweisen. Trigger, die bei diesem Ereignis ausgeführt werden, werden immer dann ausgeführt, wenn dieses Ereignis eintritt.

`NEW_DEVICE`

Wird ausgeführt, wenn Aktivität zum ersten Mal auf einem Gerät beobachtet wird. Dieses Ereignis verbraucht Erfassungsressourcen.

Hinweis Sie können Trigger, die nur bei diesem Ereignis ausgeführt werden, nicht bestimmten Geräten oder Gerätegruppen zuweisen. Trigger, die bei diesem Ereignis ausgeführt werden, werden immer dann ausgeführt, wenn dieses Ereignis eintritt.

Eigenschaften

`application:` ***Bewerbung***

Eine neu entdeckte Anwendung.

Gilt nur für `NEW_APPLICATION` Ereignisse.

`device:` ***Gerät***

Ein neu entdecktes Gerät.

Gilt nur für `NEW_DEVICE` Ereignisse.

Hinweis Sie können diese Immobilie nicht als Teilnehmer an der angeben `commitDetection` Funktion.

Beispiele für Trigger

- Beispiel: Erkannte Gerätedaten an einen Remote-Syslog-Server senden

`ExternalData`

Die `ExternalData` class ermöglicht es Ihnen, Daten abzurufen, die von externen Quellen über die ExtraHop REST API an die Trigger-API gesendet wurden.

Ereignisse

`EXTERNAL_DATA`

Wird jedes Mal ausgeführt, wenn Daten über das ExtraHop-System an das ExtraHop-System gesendet werden POST-Trigger/externe Daten Betrieb.

Eigenschaften

`body:` ***Schnur***

Die externen Daten, die an den Auslöser gesendet wurden.

`type:` ***Schnur***

Ein Bezeichner, der die an den Auslöser gesendeten Daten beschreibt. Der Typ wird definiert, wenn die Daten an die ExtraHop REST API gesendet werden.

`Flow`

Flow bezieht sich auf eine Konversation zwischen zwei Endpunkten über einen Protokoll wie TCP, UDP oder ICMP. Das `Flow` Die Klasse bietet Zugriff auf Elemente dieser Konversationen, wie z. B. Endpunkt-IP-Adressen und Alter des Datenflusses. Die Flow-Klasse enthält auch einen Flow-Speicher, der entworfen wurde, um Objekte im selben Flow von der Anfrage zur Antwort zu übergeben.

Hinweis: Sie können die Flow-Klasse auf die meisten anwenden L7 Protokollereignisse, aber es wird bei Sitzungs- oder Datenspeicherereignissen nicht unterstützt.

Ereignisse

Wenn ein Fluss mit einem ExtraHop-überwachten verknüpft ist L7 Protokoll, Ereignisse, die mit dem Protokoll korrelieren, werden zusätzlich zu Flow-Ereignissen ausgeführt. Zum Beispiel ein Fluss, der verknüpft ist mit HTTP wird auch das ausführen `HTTP_REQUEST` und `HTTP_RESPONSE` Ereignisse.

`FLOW_CLASSIFY`

Wird immer dann ausgeführt, wenn das ExtraHop-System einen Fluss anfänglich als einem bestimmten Protokoll zugeordnet klassifiziert.

Hinweis: Für TCP-Flows ist der `FLOW_CLASSIFY` Ereignis läuft nach dem `TCP_OPEN` Ereignis.

Durch eine Kombination von L7 Nutzlastanalyse, Beobachtung von TCP-Handshakes und auf Portnummern basierende Heuristiken, `FLOW_CLASSIFY` Ereignis identifiziert das L7-Protokoll und die Geräterollen für die Endpunkte in einem Fluss, wie Client/server oder sender/empfänger.

Die Art eines Fluss kann sich im Laufe seiner Lebensdauer ändern, z. B. beim Tunneln über HTTP oder beim Wechsel von SMTP zu SMTP-SSL . In diesen Fällen `FLOW_CLASSIFY` läuft nach der Protokolländerung erneut.

Das `FLOW_CLASSIFY` Ereignis ist nützlich, um eine Aktion für einen Fluss zu initiieren, die auf frühester Kenntnis von Flow-Informationen wie dem L7-Protokoll, Client-/Server-IP-Adressen oder Sender-/Empfänger-Ports basiert.

Gemeinsame Maßnahmen eingeleitet am FLOW_CLASSIFY beinhaltet das Starten einer PCAP über den captureStart() Verfahren zum Zuordnen des Fluss zu einem Anwendungscontainer durch den addApplication() Methode.

Zusätzliche Optionen sind verfügbar, wenn Sie einen Auslöser erstellen, der für dieses Ereignis ausgeführt wird. In der Standardeinstellung FLOW_CLASSIFY wird nicht nach Ablauf eines Flows ausgeführt. Sie können dafür jedoch einen Auslöser konfigurieren, um Metriken für Flows zu sammeln, die vor dem Ablauf nicht klassifiziert wurden. siehe Erweiterte Trigger-Optionen für weitere Informationen.

FLOW_DETACH

Wird ausgeführt, wenn der Parser auf einen unerwarteten Fehler gestoßen ist oder der Speicher knapp wird, und folgt dem Fluss nicht mehr. Darüber hinaus kann ein Datenfeed von geringer Qualität mit fehlenden Paketen dazu führen, dass der Parser getrennt wird.

Das FLOW_DETACH Ereignis ist nützlich, um bösartige Inhalte zu erkennen, die gesendet wurden von Klienten und Server. Das Folgende ist ein Beispiel dafür, wie ein Auslöser Fehler erkennen kann DNS Antworten auf FLOW_DETACH Ereignisse:

```
if (event == "FLOW_DETACH" && Flow.l7proto== "DNS") {
    Flow.addApplication("Malformed DNS");
}
```

FLOW_RECORD

Ermöglicht das Aufzeichnen von Informationen über einen Fluss in bestimmten Intervallen. Nachher FLOW_CLASSIFY ist gelaufen, der FLOW_RECORD Die Ereignis findet jedes Jahr statt *N* Sekunden und immer dann, wenn sich ein Fluss schließt. Der Standardwert für *N*, das sogenannte Veröffentlichungsintervall, beträgt 30 Minuten; der Mindestwert beträgt 60 Sekunden. Sie können das Veröffentlichungsintervall in den Administrationseinstellungen festlegen.

FLOW_TICK

Ermöglicht das Aufzeichnen von Informationen über einen Fluss pro Datenmenge oder pro Spielzug. Das FLOW_TICK Die Ereignis findet an jedem statt FLOW_TURN oder alle 128 Pakete, je nachdem, was zuerst eintritt. Außerdem L2 Daten werden bei jedem zurückgesetzt FLOW_TICK Ereignis, mit dem Sie bei jedem Tick Daten zusammenfügen können. Wenn Sie den Durchsatz zählen, sammeln Sie Daten von FLOW_TICK Ereignisse, die vollständigere Metriken liefern als FLOW_TURN.

FLOW_TICK bietet eine Möglichkeit, in regelmäßigen Abständen nach bestimmten Bedingungen im Datenfluss zu suchen, z. B. Nullfenster und Nagle-Verzögerungen, und dann eine Aktion zu ergreifen, z. B. eine PCAP zu initiieren oder eine Syslog-Meldung zu senden.

Das Folgende ist ein Beispiel für 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

Läuft bei jedem TCP- oder UDP-Turn. Eine Kurve steht für einen vollen Zyklus eines Client Übertragung von Anforderungsdaten, gefolgt von einem Server, der eine Antwort überträgt.

FLOW_TURN entlarvt auch eine Turn Objekt.

Endpunkte

Flow bezieht sich auf eine Konversation zwischen zwei Endpunkten über ein Protokoll; ein Endpunkt kann eine der folgenden Komponenten sein:

- `client`
- `server`
- `sender`
- `receiver`

Die in diesem Abschnitt beschriebenen Methoden und Eigenschaften werden für einen bestimmten Endpunkt im Fluss aufgerufen oder abgerufen. Zum Beispiel, um auf das zuzugreifen `device` Eigenschaft von einem HTTP-Client, die Syntax ist `Flow.client.device`.

Der Endpunkt, den Sie angeben, hängt von den Ereignissen ab, die mit dem Auslöser verknüpft sind. Zum Beispiel die `ACTIVEMQ_MESSAGE` Ereignis unterstützt nur Sender- und Empfängerendpunkte. Die folgende Tabelle enthält eine Liste von Ereignissen, die einem Fluss zugeordnet werden können, sowie die Endpunkte, die für jedes Ereignis unterstützt werden:

Ereignis	Kunde/Server	Absender/ Empfänger
AAA_REQUEST	Ja	Ja
AAA_RESPONSE	Ja	Ja
AJP_REQUEST	Ja	Ja
AJP_RESPONSE	Ja	Ja
ACTIVEMQ_MESSAGE	nein	Ja
CIFS_REQUEST	Ja	Ja
CIFS_RESPONSE	Ja	Ja
DB_REQUEST	Ja	Ja
DB_RESPONSE	Ja	Ja
DHCP_REQUEST	Ja	Ja
DHCP_RESPONSE	Ja	Ja
DICOM_REQUEST	Ja	Ja
DICOM_RESPONSE	Ja	Ja
DNS_REQUEST	Ja	Ja
DNS_RESPONSE	Ja	Ja
FIX_REQUEST	Ja	Ja
FIX_RESPONSE	Ja	Ja
FLOW_CLASSIFY	Ja	nein
FLOW_DETACH	Ja	nein
FLOW_RECORD	Ja	nein
FLOW_TICK	Ja	nein
FLOW_TURN	Ja	nein
FTP_REQUEST	Ja	Ja
FTP_RESPONSE	Ja	Ja

Ereignis	Kunde/Server	Absender/ Empfänger
HL7_REQUEST	Ja	Ja
HL7_RESPONSE	Ja	Ja
HTTP_REQUEST	Ja	Ja
HTTP_RESPONSE	Ja	Ja
IBMMQ_REQUEST	Ja	Ja
IBMMQ_RESPONSE	Ja	Ja
ICA_AUTH	Ja	nein
ICA_CLOSE	Ja	nein
ICA_OPEN	Ja	nein
ICA_TICK	Ja	nein
ICMP_MESSAGE	nein	Ja
KERBEROS_REQUEST	Ja	Ja
KERBEROS_RESPONSE	Ja	Ja
LDAP_REQUEST	Ja	Ja
LDAP_RESPONSE	Ja	Ja
MEMCACHE_REQUEST	Ja	Ja
MEMCACHE_RESPONSE	Ja	Ja
MOBUS_REQUEST	Ja	Ja
MODBUS_RESPONSE	Ja	Ja
MONGODB_REQUEST	Ja	Ja
MONGODB_RESPONSE	Ja	Ja
MSMQ_MESSAGE	nein	Ja
NFS_REQUEST	Ja	Ja
NFS_RESPONSE	Ja	Ja
POP3_REQUEST	Ja	Ja
POP3_RESPONSE	Ja	Ja
REDIS_REQUEST	Ja	Ja
REDIS_RESPONSE	Ja	Ja
RDP_CLOSE	Ja	nein
RDP_OPEN	Ja	nein
RDP_TICK	Ja	nein
RTCP_MESSAGE	nein	Ja
RTP_CLOSE	nein	Ja
RTP_OPEN	nein	Ja

Ereignis	Kunde/Server	Absender/ Empfänger
RTP_TICK	nein	Ja
SCCP_MESSAGE	nein	Ja
SIP_REQUEST	Ja	Ja
SIP_RESPONSE	Ja	Ja
SMPP_REQUEST	Ja	Ja
SMPP_RESPONSE	Ja	Ja
SMTP_REQUEST	Ja	Ja
SMTP_RESPONSE	Ja	Ja
SSL_ALERT	Ja	Ja
SSL_CLOSE	Ja	nein
SSL_HEARTBEAT	Ja	Ja
SSL_OPEN	Ja	nein
SSL_PAYLOAD	Ja	Ja
SSL_RECORD	Ja	Ja
SSL_RENEGOTIATE	Ja	nein
TCP_CLOSE	Ja	nein
TCP_OPEN	Ja	nein
TCP_PAYLOAD	Ja	Ja
UDP_PAYLOAD	Ja	Ja
TELNET_MESSAGE	Ja	Ja
WEBSOCKET_OPEN	Ja	nein
WEBSOCKET_CLOSE	Ja	nein
WEBSOCKET_MESSAGE	Ja	Ja

Endpunktmethoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `FLOW_RECORD` Ereignis. Datensatz-Commits werden nicht unterstützt auf `FLOW_CLASSIFY`, `FLOW_DETACH`, `FLOW_TICK`, oder `FLOW_TURN` Ereignisse.

Bei einem Fluss bewegt sich der Verkehr in jede Richtung zwischen zwei Endpunkten. Das `commitRecord()` Die Methode zeichnet nur Flussdetails in einer Richtung auf, z. B. vom Client zum Server. Um Details über den gesamten Fluss Datensatz, müssen Sie anrufen `commitRecord()` zweimal, einmal für jede Richtung, und geben Sie den Endpunkt in der Syntax an, z. B. `Flow.client.commitRecord()` und `Flow.server.commitRecord()`.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Informationen zu den Standardeigenschaften, die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Eigenschaften von Endpunkten

`bytes:` ***Zahl***

Die Zahl der L4 Nutzdatenbytes, die von einem Gerät übertragen werden. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.bytes` oder `Flow.receiver.bytes`.

Zugriff nur auf `FLOW_TICK`, `FLOW_TURN`, oder `FLOW_RECORD` Ereignisse; andernfalls tritt ein Fehler auf.

`customDevices:` ***Reihe***

Eine Reihe von benutzerdefinierten Geräten im Fluss. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.customDevices` oder `Flow.receiver.customDevices`.

`device:` ***Gerät***

Das Device Objekt, das einem Gerät zugeordnet ist. Geben Sie die Geräterolle in der Syntax an. Um beispielsweise auf die MAC-Adresse des zuzugreifen Client Gerät, spezifizieren `Flow.client.device.hwaddr`.

`equals:` ***Boolesch***

Führt einen Gleichheitstest durch zwischen Device Objekte.

`dscp:` ***Zahl***

Die Zahl, die den letzten DSCP-Wert (Differentiated Services Code Point) des Flow-Pakets darstellt.

Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.dscp` oder `Flow.server.dscp`.

`dscpBytes:` ***Reihe***

Ein Array, das die Anzahl von enthält L2 Byte für einen bestimmten Differentiated Services Code Point (DSCP) -Wert, der von einem Gerät im Fluss übertragen wird. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.dscpBytes` oder `Flow.server.dscpBytes`.

Der Wert ist Null für jeden Eintrag, der seit dem letzten keine Byte des spezifischen DSCP enthält. `FLOW_TICK` Ereignis.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`dscpName1:` ***Schnur***

Der Name, der dem von device1 im Fluss übertragenen DSCP-Wert zugeordnet ist. In der folgenden Tabelle werden bekannte DSCP-Namen aufgeführt:

Zahl	Name
8	CS1
10	AF11
12	AF12
14	AF13
16	CS2
18	AF21
20	AF22
22	AF23
24	CS3
26	AF31
28	AF32

Zahl	Name
30	AF33
32	CS4
34	AF41
36	AF42
38	AF43
40	CS5
44	VA
46	EF
48	CS6
56	CS7

`dscpName2:` ***Zeichenfolge***

Der Name, der dem DSCP-Wert zugeordnet ist, der von device2 im Fluss übertragen wird. In der folgenden Tabelle werden bekannte DSCP-Namen aufgeführt:

Zahl	Name
8	CS1
10	AF11
12	AF12
14	AF13
16	CS2
18	AF21
20	AF22
22	AF23
24	CS3
26	AF31
28	AF32
30	AF33
32	CS4
34	AF41
36	AF42
38	AF43
40	CS5
44	VA
46	EF
48	CS6

Zahl	Name
56	CS7

`dscpPkts:` ***Reihe***

Ein Array, das die Anzahl von enthält L2 Pakete für einen bestimmten Differentiated Services Code Point (DSCP) -Wert, der von einem Gerät im Datenfluss übertragen wird. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.dscpPkts` oder `Flow.server.dscpPkts`.

Der Wert ist Null für jeden Eintrag, der seit dem letzten keine Pakete des spezifischen DSCP enthält. `FLOW_TICK` Ereignis.

Gilt nur für `FLOW_TICK` oder `FLOW_TURN` Ereignisse.

`fragPkts:` ***Zahl***

Die Anzahl der Pakete, die sich aus der IP-Fragmentierung ergeben, die von einem Client- oder Servergerät im Fluss übertragen werden. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.fragPkts` oder `Flow.server.fragPkts`.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`ipaddr1:` ***IP Adresse***

Das IPAddress Objekt, das Gerät1 im Fluss zugeordnet ist.

`equals:` ***Boolesch***

Führt einen Gleichheitstest durch zwischen IPAddress Objekte.

`ipaddr2:` ***IP Adresse***

Das IPAddress Objekt, das Gerät2 im Fluss zugeordnet ist.

`equals:` ***Boolesch***

Führt einen Gleichheitstest durch zwischen IPAddress Objekte.

`isAborted:` ***Boolesch***

Der Wert ist `true` wenn ein TCP-Flow durch einen TCP-Reset (RST) abgebrochen wurde. Der Fluss kann durch ein Gerät abgebrochen werden. Geben Sie gegebenenfalls die Geräterolle in der Syntax an, z. B. `Flow.client.isAborted` oder `Flow.receiver.isAborted`.

Dieser Zustand kann in der `TCP_CLOSE` Ereignis und in allen betroffenen L7 Ereignisse (zum Beispiel `HTTP_REQUEST` oder `DB_RESPONSE`).

Hinweis:
- Ein L4 Ein Abbruch tritt auf, wenn eine TCP-Verbindung mit einem RST statt mit einem ordnungsgemäßen Herunterfahren geschlossen wird.
- Ein L7-Antwortabbruch tritt auf, wenn eine Verbindung während einer Antwort geschlossen wird. Dies kann an einem RST, einem ordnungsgemäßen FIN-Shutdown oder einem Ablauf liegen.
- Ein L7-Anforderungsabbruch erfolgt, wenn eine Verbindung mitten in einer Anfrage geschlossen wird. Dies kann auch an einem RST, einem ordnungsgemäßen FIN-Shutdown oder einem Ablauf liegen.

`isShutdown:` ***Boolesch***

Der Wert ist `true` wenn das Gerät das Herunterfahren der TCP-Verbindung initiiert hat. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.isShutdown` oder `Flow.receiver.isShutdown`.

`l2Bytes:` ***Zahl***

Die Zahl der L2 Byte, einschließlich der Ethernet-Header, die von einem Gerät im Datenfluss übertragen werden. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.l2Bytes` oder `Flow.server.l2Bytes`.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`nagleDelay:` ***Zahl***

Die Anzahl der Nagle-Verzögerungen, die mit einem Gerät im Fluss verbunden sind. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.nagleDelay` oder `Flow.server.nagleDelay`.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`overlapFragPkts:` ***Zahl***

Die Anzahl der nicht identischen IP-Fragmentpakete mit überlappenden Daten, die von einem Gerät im Datenfluss übertragen werden. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.overlapFragPkts` oder `Flow.server.overlapFragPkts`.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`overlapSegments:` ***Zahl***

Die Anzahl der nicht identischen TCP-Segmente, die von einem Gerät im Fluss übertragen werden, wobei zwei oder mehr TCP-Segmente Daten für denselben Teil des Datenflusses enthalten. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.overlapSegments` oder `Flow.server.overlapSegments`.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`payload:` ***Puffer***

Die Nutzlast Puffer einem Gerät im Fluss zugeordnet. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.payload` oder `Flow.receiver.payload`.

Zugriff nur auf `TCP_PAYLOAD`, `UDP_PAYLOAD`, oder `SSL_PAYLOAD` Ereignisse; andernfalls tritt ein Fehler auf.

`pkts:` ***Zahl***

Die Anzahl der Pakete, die von einem Gerät im Fluss übertragen werden. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.pkts` oder `Flow.server.pkts`.

Zugriff nur auf `FLOW_TICK`, `FLOW_TURN`, oder `FLOW_RECORD` Ereignisse; andernfalls tritt ein Fehler auf.

`port:` ***Zahl***

Die Portnummer, die einem Gerät im Fluss zugeordnet ist. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.port` oder `Flow.receiver.port`.

`rcvWndThrottle:` ***Zahl***

Die Anzahl der Empfangsfensterdrosseln, die von einem Gerät im Fluss gesendet wurden. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.rcvWndThrottle` oder `Flow.server.rcvWndThrottle`.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `Flow.commitRecord()` auf einem `FLOW_RECORD` Ereignis. Das Record-Objekt stellt Daten aus einer einzigen Flussrichtung dar.

Das Standard-Datensatzobjekt kann die folgenden Eigenschaften enthalten:

- `age`
- `bytes (L3)`

  Hinweis: Diese Eigenschaft stellt die Gesamtzahl der Byte dar, die vom Flow zum Zeitpunkt der Ausführung des FLOW_RECORD-Ereignisses übertragen wurden. Das FLOW_RECORD-Ereignis wird im Verlauf jedes Flows mehrmals ausgeführt, sodass der Wert bei jeder Ausführung des Ereignis erhöht wird.
- `clientIsExternal`

- dscpName
- first
- firstPayloadBytes

  Eine hexadezimale Darstellung der ersten 16 Nutzdatenbytes im Datenfluss.
- last
- pkts
- proto
- receiverAddr
- receiverIsExternal
- receiverPort
- roundTripTime

  Die letzte Hin- und Rückflugzeit (RTT) in diesem Fluss. Ein RTT ist die Zeit, die ein Gerät benötigt hat, um ein Paket zu senden und eine sofortige Bestätigung (ACK) zu empfangen.
- senderAddr
- senderIsExternal
- senderPort
- serverIsExternal
- tcpFlags

Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.record` oder `Flow.server.record`.

Greifen Sie nur auf das Datensatzobjekt zu `FLOW_RECORD` Ereignisse; andernfalls tritt ein Fehler auf.

`rto:` ***Zahl***

Die Zahl der Zeitüberschreitungen bei der erneuten Übertragung (RTOs), das einem Gerät im Fluss zugeordnet ist. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.rto` oder `Flow.server.rto`.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`totalL2Bytes`

Die Anzahl der L2-Byte, die von einem Gerät während des Datenflusses gesendet wurden. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.totalL2Bytes` oder `Flow.server.totalL2Bytes`.

`totalL2Bytes1:` ***Zahl***

Die Anzahl der L2-Byte, die während des Datenflusses von Gerät1 gesendet wurden.

`totalL2Bytes2:` ***Zahl***

Die Anzahl der L2-Byte, die während des Datenflusses von device2 gesendet wurden.

`zeroWnd:` ***Zahl***

Die Anzahl der Nullfenster, die von einem Gerät im Fluss gesendet wurden. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.zeroWnd` oder `Flow.server.zeroWnd`.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

Methoden

`addApplication(name:` ***Schnur*** `, turnTiming:` ***Boolesch*** `):` ***Leere***

Erstellt eine Anwendung mit dem angegebenen Namen und sammelt L2-L4-Metriken aus dem Fluss. Die Anwendung kann im ExtraHop-System angesehen werden und die Metriken werden auf einer L4-Seite in der Anwendung angezeigt. Ein Fluss kann zu einem bestimmten Zeitpunkt einer oder mehreren Anwendungen zugeordnet werden. Die von jeder Anwendung gesammelten L2-L4-Metriken sind dieselben.

Anrufen `Flow.addApplication(name)` auf einem `FLOW_CLASSIFY` Ereignis tritt häufig bei nicht unterstützten Protokollen auf. Für Datenflüsse auf unterstützten Protokollen mit L7 Ereignisse Auslöser, es wird empfohlen, den `Application(name).commit()` Methode, die einen größeren Satz von Protokollmetriken sammelt.

Das optionale `turnTiming` flag ist standardmäßig auf false gesetzt. Wenn der Wert auf „true“ gesetzt ist, erfasst das ExtraHop-System zusätzliche Abbiege-Timing-Metriken für den Fluss. Wenn dieses Flag weggelassen wird, werden für die Anwendung keine Abbiegezeit-Metriken für den zugehörigen Fluss aufgezeichnet. Analysen zur Abbiegezeitanalyse L4 Verhalten , um L7-Verarbeitungszeiten abzuleiten, wenn das überwachte Protokoll einem Client-Anforderungs- und Server-Antwortmuster folgt und in dem der Client die erste Nachricht sendet. „Banner“ -Protokolle (bei denen der Server die erste Nachricht sendet) und Protokolle, bei denen Daten gleichzeitig in beide Richtungen fließen, werden für die Abbiegezeitanalyse nicht empfohlen.

`captureStart(name:` ***Schnur*** `, options:` ***Objekt*** `):` ***Schnur***

Initiiert eine Precision Packet Capture (PPCAP) für den Fluss und gibt eine eindeutige Kennung der PCAP im Format einer Dezimalzahl als Zeichenfolge zurück. kehrt zurück `null` wenn die PCAP nicht gestartet werden kann.

`name:` ***Schnur***

Der Name der Paketerfassungsdatei.

- Die maximale Länge beträgt 256 Zeichen
- Für jeden Fluss wird eine separate Erfassung erstellt.
- Erfassungsdateien mit demselben Namen werden durch Zeitstempel unterschieden.

`options:` ***Objekt***

Die im Capture-Objekt enthaltenen Optionen. Lassen Sie eine der Optionen weg, um eine unbegrenzte Größe für diese Option anzugeben. Alle Optionen gelten für den gesamten Fluss, mit Ausnahme der „Lookback“ -Optionen, die nur für den Teil des Fluss gelten, der vor dem Triggerereignis lag, das die PCAP gestartet hat.

`maxBytes:` ***Zahl***

Die maximale Gesamtanzahl von Byte.

`maxBytesLookback:` ***Zahl***

Die maximale Gesamtanzahl von Byte aus dem Lookback-Puffer. Der Lookback-Puffer bezieht sich auf Pakete, die vor dem Aufruf von erfasst wurden `Flow.captureStart()`.

`maxDurationMSec:` ***Zahl***

Die maximale Dauer der PCAP, ausgedrückt in Millisekunden.

`maxPackets:` ***Zahl***

Die maximale Gesamtanzahl von Paketen. Der Maximalwert könnte überschritten werden , wenn Triggerlast ist schwer.

`maxPacketsLookback:` ***Zahl***

Die maximale Anzahl von Paketen aus dem Lookback-Puffer. Der Lookback-Puffer bezieht sich auf Pakete, die vor dem Aufruf von erfasst wurden `Flow.captureStart()`.

Das Folgende ist ein Beispiel für `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);
}
```

Hinweis: Das `Flow.captureStart()` Für den Funktionsaufruf benötigen Sie eine Lizenz für die präzise PCAP.

- Sie können die Anzahl der Byte pro Paket (Snaplen) angeben, die Sie erfassen möchten, wenn Sie den Auslöser im ExtraHop-System konfigurieren. Diese Option ist nur bei einigen Veranstaltungen verfügbar. siehe Erweiterte Trigger-Optionen für weitere Informationen.
- Auf ExtraHop Performance-Systemen sind die erfassten Dateien in den Administrationseinstellungen verfügbar. Auf Reveal (x) -Systemen sind erfasste Dateien auf der Seite Pakete im ExtraHop-System verfügbar.
- Wenn auf ExtraHop Performance-Systemen die Precision-Paketerfassungsdiskette voll ist, werden keine neuen Captures aufgezeichnet, bis der Benutzer die Dateien manuell löscht. Auf Reveal-Systemen werden ältere Paketerfassungen gelöscht, wenn die Precision PCAP Capture-Festplatte voll ist, damit das System weiterhin neue Paketerfassungen aufzeichnen kann.
- Die maximale Länge der Zeichenfolge für Dateinamen beträgt 256 Zeichen. Wenn der Name 256 Zeichen überschreitet, wird er gekürzt und eine Warnmeldung wird im Debug-Log angezeigt, aber der Auslöser wird weiterhin ausgeführt.
- Die Größe der Aufnahmedatei ist das Maximum, das zuerst erreicht wird zwischen `maxPackets` und `maxBytes` Optionen.
- Die Größe des Capture-Lookback-Puffers ist das Maximum, das zuerst erreicht wird zwischen den `maxPacketsLookback` und `maxBytesLookback` Optionen.
- Jeder hat bestanden `max*` Der Parameter wird bis zur nächsten Paketgrenze erfasst.
- Wenn die PCAP bereits im aktuellen Fluss gestartet wurde, `Flow.captureStart()` Aufrufe führen zu einer Warnung, die im Debug-Log sichtbar ist, aber der Auslöser wird weiterhin ausgeführt.
- Es gibt maximal 128 gleichzeitige Paketerfassungen im System. Wenn dieses Limit erreicht ist, werden nachfolgende Aufrufe an `Flow.captureStart()` generiert eine Warnung, die im Debug-Log sichtbar ist, aber der Auslöser wird weiterhin ausgeführt.

`captureStop()`: ***Boolesch***

Stoppt eine PCAP, die im aktuellen Datenfluss ausgeführt wird.

`commitRecord1()`: ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore, der Daten darstellt, die gesendet wurden von `device1` in einer einzigen Strömungsrichtung.

Sie können diese Methode nur aufrufen `FLOW_RECORD` Ereignisse, und jeder eindeutige Datensatz wird für integrierte Datensätze nur einmal festgeschrieben.

Informationen zu den Eigenschaften, die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

`commitRecord2()`: ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore, der Daten darstellt, die gesendet wurden von `device2` in einer einzigen Strömungsrichtung.

Sie können diese Methode nur aufrufen `FLOW_RECORD` Ereignisse, und jeder eindeutige Datensatz wird für integrierte Datensätze nur einmal festgeschrieben.

Informationen zu den Eigenschaften, die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

`findCustomDevice(deviceID:` ***Schnur*** `):` ***Gerät***

Gibt eine einzelne zurück Device Objekt, das dem angegebenen DeviceID-Parameter entspricht, wenn sich das Gerät auf beiden Seiten des Fluss befindet. kehrt zurück `null` wenn kein entsprechendes Gerät gefunden wird.

`getApplications():` ***Schnur***

Ruft alle mit dem Fluss verknüpften Anwendungen ab.

Eigenschaften

Die in diesem Abschnitt erläuterten Eigenschaften und Methoden des Flow-Objekts sind für jeden verfügbar. L7 Auslöser ein mit dem Fluss verbundenes Ereignis aus.

Standardmäßig verwendet das ExtraHop-System eine lose initiierte Protokollklassifizierung, sodass es versucht, Datenflüsse auch dann zu klassifizieren, wenn die Verbindung initiiert wurde. Die lose Initiierung kann für Ports deaktiviert werden, die nicht immer den Protokollverkehr übertragen (z. B. der Platzhalterport 0). Für solche Ströme `device1`, `port1`, und `ipaddr1` das Gerät mit der numerisch niedrigeren IP-Adresse darstellen und `device2`, `port2`, und `ipaddr2` stellen das Gerät mit der numerisch höheren IP-Adresse dar.

`age:` ***Zahl***

Die seit der Initiierung des Fluss verstrichene Zeit, ausgedrückt in Sekunden.

`bytes1:` ***Zahl***

Die Zahl der L4 Nutzdatenbytes, die von einem von zwei Geräten im Fluss übertragen werden; das andere Gerät wird dargestellt durch `bytes2`. Das Gerät, dargestellt durch `bytes1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK`, `FLOW_TURN`, oder `FLOW_RECORD` Ereignisse; andernfalls tritt ein Fehler auf.

`bytes2:` ***Zahl***

Die Zahl der L4 Nutzdatenbytes, die von einem von zwei Geräten im Fluss übertragen werden; das andere Gerät wird dargestellt durch `bytes1`. Das Gerät, dargestellt durch `bytes2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK`, `FLOW_TURN`, oder `FLOW_RECORD` Ereignisse; andernfalls tritt ein Fehler auf.

`customDevices1:` ***Reihe***

Eine Reihe von benutzerdefinierten Device Objekte in einem Fluss. Benutzerdefinierte Geräte auf der anderen Seite des Fluss sind verfügbar, indem Sie darauf zugreifen `customDevices2`. Das Gerät, dargestellt durch `customDevices1` bleibt für den Fluss konsistent.

`customDevices2:` ***Reihe***

Eine Reihe von benutzerdefinierten Device Objekte in einem Fluss. Benutzerdefinierte Geräte auf der anderen Seite des Fluss sind verfügbar, indem Sie darauf zugreifen `customDevices1`. Das Gerät, dargestellt durch `customDevices2` bleibt für den Fluss konsistent.

`device1:` ***Gerät***

Das Device Objekt, das einem von zwei Geräten im Fluss zugeordnet ist; das andere Gerät wird dargestellt durch `device2`. Das Gerät, dargestellt durch `device1` bleibt für den Fluss konsistent. Zum Beispiel `Flow.device1.hwaddr` greift im Fluss auf die MAC-Adressen dieses Gerät zu.

`equals:` ***Boolesch***

Führt einen Gleichheitstest durch zwischen Device Objekte.

`device2:` ***Gerät***

Das Device Objekt, das einem von zwei Geräten im Fluss zugeordnet ist; das andere Gerät wird dargestellt durch `device1`. Das Gerät, dargestellt durch `device2` bleibt für den Fluss konsistent. Zum Beispiel `Flow.device2.hwaddr` greift im Fluss auf die MAC-Adressen dieses Gerät zu.

`equals:` ***Boolesch***

Führt einen Gleichheitstest durch zwischen Device Objekte.

`dscp1:` ***Zahl***

Die Zahl, die den letzten Differentiated Services Code Point (DSCP) -Wert darstellt, der von einem von zwei Geräten im Fluss übertragen wurde; das andere Gerät wird dargestellt durch `dscp2`. Das Gerät, dargestellt durch `dscp1` bleibt für den Fluss konsistent.

`dscp2:` ***Zahl***

Die L-Nummer, die den letzten Differentiated Services Code Point (DSCP) -Wert darstellt, der von einem von zwei Geräten im Fluss übertragen wurde; das andere Gerät wird dargestellt durch `dscp1`. Das Gerät, dargestellt durch `dscp2` bleibt für den Fluss konsistent.

`dscpBytes1:` ***Reihe***

Ein Array, das die Anzahl von enthält L2 Byte für einen bestimmten Differentiated Services Code Point (DSCP) -Wert, der von einem von zwei Geräten im Fluss übertragen wird; das andere Gerät wird dargestellt durch `dscpBytes2`. Das Gerät, dargestellt durch `dscpBytes1` bleibt für den Fluss konsistent.

Der Wert ist Null für jeden Eintrag, der seit dem letzten keine Byte des spezifischen DSCP enthält. `FLOW_TICK` Ereignis.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`dscpBytes2:` ***Reihe***

Ein Array, das die Anzahl von enthält L2 Byte für einen bestimmten Differentiated Services Code Point (DSCP) -Wert, der von einem von zwei Geräten im Fluss übertragen wird; das andere Gerät wird dargestellt durch `dscpBytes1`. Das Gerät, dargestellt durch `dscpBytes2` bleibt für den Fluss konsistent.

Der Wert ist Null für jeden Eintrag, der seit dem letzten keine Byte des spezifischen DSCP enthält. `FLOW_TICK` Ereignis.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`dcspName1:` ***Schnur***

Der Name, der dem DSCP-Wert zugeordnet ist, der von einem von zwei Geräten im Fluss übertragen wird; das andere Gerät wird dargestellt durch `dscpName2`. Das Gerät, dargestellt durch `dscpName1` bleibt für den Fluss konsistent.

Sehen Sie die `dscpName` Eigentum in der Endpunkte Abschnitt für eine Liste der unterstützten DSCP-Codenamen.

`dcspName2:` ***Schnur***

Der Name, der dem DSCP-Wert zugeordnet ist, der von einem von zwei Geräten im Fluss übertragen wird; das andere Gerät wird dargestellt durch `dscpName1`. Das Gerät, dargestellt durch `dscpName2` bleibt für den Fluss konsistent.

Sehen Sie die `dscpName` Eigentum in der Endpunkte Abschnitt für eine Liste der unterstützten DSCP-Codenamen.

`dscpPkts1:` ***Reihe***

Ein Array, das die Anzahl von enthält L2 Pakete für einen bestimmten Differentiated Services Code Point (DSCP) -Wert, der von einem von zwei Geräten im Datenfluss übertragen wird; das andere Gerät wird dargestellt durch `dscpPkts2`. Das Gerät, dargestellt durch `dscpPkts1` bleibt für den Fluss konsistent.

Der Wert ist Null für jeden Eintrag, der seit dem letzten keine Pakete des spezifischen DSCP enthält. `FLOW_TICK` Ereignis.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`dscpPkts2:` ***Reihe***

Ein Array, das die Anzahl von enthält L2 Pakete für einen bestimmten Differentiated Services Code Point (DSCP) -Wert, der von einem von zwei Geräten im Datenfluss übertragen wird; das andere Gerät wird dargestellt durch `dscpPkts1`. Das Gerät, dargestellt durch `dscpPkts2` bleibt für den Fluss konsistent.

Der Wert ist Null für jeden Eintrag, der seit dem letzten keine Pakete des spezifischen DSCP enthält. `FLOW_TICK` Ereignis.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`fragPkts1:` ***Zahl***

Die Anzahl der Pakete, die aus der IP-Fragmentierung resultieren und von einem von zwei Geräten im Datenfluss übertragen werden; das andere Gerät wird dargestellt durch `fragPkts2`. Das Gerät, dargestellt durch `fragPkts1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`fragPkts2:` ***Zahl***

Die Anzahl der Pakete, die aus der IP-Fragmentierung resultieren und von einem von zwei Geräten im Datenfluss übertragen werden; das andere Gerät wird dargestellt durch `fragPkts1`. Das Gerät, dargestellt durch `fragPkts2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`id:` ***Schnur***

Die eindeutige Kennung eines Flow-Datensatzes.

`ipaddr:` ***IP Adresse***

Das IPAddress Objekt, das einem Gerät im Fluss zugeordnet ist. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.ipaddr` oder `Flow.receiver.ipaddr`.

`equals:` ***Boolesch***

Führt einen Gleichheitstest durch zwischen IPAddress Objekte.

`ipproto:` ***Schnur***

Das mit dem Fluss verknüpfte IP-Protokoll, z. B. TCP oder UDP.

`ipver:` ***Schnur***

Die dem Fluss zugeordnete IP-Version, z. B. IPv4 oder IPv6.

`isAborted:` ***Boolesch***

Der Wert ist `true` wenn ein TCP-Flow durch einen TCP-Reset (RST) abgebrochen wurde. Der Fluss kann durch ein Gerät abgebrochen werden. Geben Sie gegebenenfalls die Geräterolle in der Syntax an, z. B. `Flow.client.isAborted` oder `Flow.receiver.isAborted`.

Dieser Zustand kann in der `TCP_CLOSE` Ereignis und in allen betroffenen L7 Ereignisse (zum Beispiel `HTTP_REQUEST` oder `DB_RESPONSE`).

Hinweis: Ein L4 Ein Abbruch tritt auf, wenn eine TCP-Verbindung mit einem RST statt mit einem ordnungsgemäßen Herunterfahren geschlossen wird.

- Ein L7-Antwortabbruch tritt auf, wenn eine Verbindung während einer Antwort geschlossen wird. Dies kann an einem RST, einem ordnungsgemäßen FIN-Shutdown oder einem Ablauf liegen.
- Ein L7-Anforderungsabbruch erfolgt, wenn eine Verbindung mitten in einer Anfrage geschlossen wird. Dies kann auch an einem RST, einem ordnungsgemäßen FIN-Shutdown oder einem Ablauf liegen.

`isExpired:` ***Boolesch***

Der Wert ist `true` wenn der Fluss zum Zeitpunkt des Ereignisses abgelaufen ist.

`isShutdown:` ***Boolesch***

Der Wert ist `true` wenn das Gerät das Herunterfahren der TCP-Verbindung initiiert hat. Geben Sie die Geräterolle in der Syntax an, z. B. `Flow.client.isShutdown` oder `Flow.receiver.isShutdown`.

`l2Bytes1:` ***Zahl***

Die Zahl der L2 Byte, einschließlich der Ethernet-Header, werden von einem von zwei Geräten im Datenfluss übertragen; das andere Gerät wird dargestellt durch `l2Bytes2`. Das Gerät, dargestellt durch `l2Bytes1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`l2Bytes2:` ***Zahl***

Die Zahl der L2 Byte, einschließlich der Ethernet-Header, werden von einem von zwei Geräten im Datenfluss übertragen; das andere Gerät wird dargestellt durch `l2Bytes1`. Das Gerät, dargestellt durch `l2Bytes2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`l7proto:` ***Schnur***

Das mit dem Fluss verknüpfte L7-Protokoll. Bei bekannten Protokollen gibt die Eigenschaft eine Zeichenfolge zurück, die den Protokollnamen darstellt, z. B. HTTP, DHCP, Memcache. Für weniger bekannte Protokolle gibt die Eigenschaft eine Zeichenfolge zurück, die formatiert ist als `ipproto:port—tcp:13724` oder `udp:11258` Für benutzerdefinierte Protokollnamen gibt die Eigenschaft eine Zeichenfolge zurück, die den Namen darstellt, der im Abschnitt Protokollklassifizierung in den Administrationseinstellungen festgelegt wurde.

Diese Eigenschaft ist nicht gültig während `TCP_OPEN` Ereignisse.

`nagleDelay1:` ***Zahl***

Die Anzahl der Nagle-Verzögerungen, die einem von zwei Geräten im Fluss zugeordnet sind; das andere Gerät wird dargestellt durch `nagleDelay2`. Das Gerät, dargestellt durch `nagleDelay1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`nagleDelay2:` ***Zahl***

Die Anzahl der Nagle-Verzögerungen, die einem von zwei Geräten im Fluss zugeordnet sind; das andere Gerät wird dargestellt durch `nagleDelay1`. Das Gerät, dargestellt durch `nagleDelay2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`overlapFragPkts1:` ***Zahl***

Die Anzahl der nicht identischen IP-Fragmentpakete, die von einem von zwei Geräten im Fluss übertragen werden; das andere Gerät wird dargestellt durch `overlapFragPkts2`. Das Gerät, dargestellt durch `overlapFragPkts1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`overlapFragPkts2:` ***Zahl***

Die Anzahl der nicht identischen IP-Fragmentpakete, die von einem von zwei Geräten im Fluss übertragen werden; das andere Gerät wird dargestellt durch `overlapFragPkts1`. Das Gerät, dargestellt durch `overlapFragPkts2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`overlapSegments1:` ***Zahl***

Die Anzahl der nicht identischen TCP-Segmente, bei denen zwei oder mehr Segmente Daten für denselben Teil des Fluss enthalten. Die TCP-Segmente werden von einem von zwei Geräten im Fluss

übertragen; das andere Gerät wird dargestellt durch `overlapSegments2`. Das Gerät, dargestellt durch `overlapSegments1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`overlapSegments2:` ***Zahl***

Die Anzahl der nicht identischen TCP-Segmente, bei denen zwei oder mehr Segmente Daten für denselben Teil des Fluss enthalten. Die TCP-Segmente werden von einem von zwei Geräten im Fluss übertragen; das andere Gerät wird dargestellt durch `overlapSegments1`. Das Gerät, dargestellt durch `overlapSegments2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`payload1:` ***Puffer***

Die Nutzlast Puffer einem von zwei Geräten im Fluss zugeordnet; das andere Gerät wird dargestellt durch `payload2`. Das Gerät, dargestellt durch `payload1` bleibt für den Fluss konsistent.

Zugriff nur auf `TCP_PAYLOAD`, `UDP_PAYLOAD`, und `SSL_PAYLOAD` Ereignisse; andernfalls tritt ein Fehler auf.

`payload2:` ***Puffer***

Die Nutzlast Puffer einem von zwei Geräten im Fluss zugeordnet; das andere Gerät wird dargestellt durch `payload1`. Das Gerät, dargestellt durch `payload2` bleibt für den Fluss konsistent.

Zugriff nur auf `TCP_PAYLOAD`, `UDP_PAYLOAD`, oder `SSL_PAYLOAD` Ereignisse; andernfalls tritt ein Fehler auf.

`pkts1:` ***Zahl***

Die Anzahl der Pakete, die von einem von zwei Geräten im Datenfluss übertragen werden; das andere Gerät wird dargestellt durch `pkts2`. Das Gerät, dargestellt durch `pkts1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK`, `FLOW_TURN`, oder `FLOW_RECORD` Ereignisse; andernfalls tritt ein Fehler auf.

`pkts2:` ***Zahl***

Die Anzahl der Pakete, die von einem von zwei Geräten im Datenfluss übertragen werden; das andere Gerät wird dargestellt durch `pkts1`. Das Gerät, dargestellt durch `pkts2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK`, `FLOW_TURN`, oder `FLOW_RECORD` Ereignisse; andernfalls tritt ein Fehler auf.

`port1:` ***Zahl***

Die Portnummer, die einem von zwei Geräten in einem Fluss zugeordnet ist; das andere Gerät wird dargestellt durch `port2`. Das Gerät, dargestellt durch `port1` bleibt für den Fluss konsistent.

`port2:` ***Zahl***

Die Portnummer, die einem von zwei Geräten in einem Fluss zugeordnet ist; das andere Gerät wird dargestellt durch `port1`. Das Gerät, dargestellt durch `port2` bleibt für den Fluss konsistent.

`rcvWndThrottle1:` ***Zahl***

Die Anzahl der Empfangsfensterdrosseln, die von einem von zwei Geräten im Fluss gesendet wurden; das andere Gerät wird dargestellt durch `rcvWndThrottle2`. Das Gerät, dargestellt durch `rcvWndThrottle1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`rcvWndThrottle2:` ***Zahl***

Die Anzahl der Empfangsfensterdrosseln, die von einem von zwei Geräten im Fluss gesendet wurden; das andere Gerät wird dargestellt durch `rcvWndThrottle1`. Das Gerät, dargestellt durch `rcvWndThrottle2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`record1:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `Flow.commitRecord1()` auf einem `FLOW_RECORD` Ereignis.

Das Objekt stellt den Verkehr dar, der von einem von zwei Geräten im Datenfluss in eine einzige Richtung gesendet wird. Das andere Gerät wird durch das `record2` Eigentum. Das Gerät, dargestellt durch den `record1` Die Eigenschaft bleibt für den Fluss konstant.

Greifen Sie nur auf das Datensatzobjekt zu `FLOW_RECORD` Ereignisse; andernfalls tritt ein Fehler auf.

Das Standard-Datensatzobjekt kann die folgenden Eigenschaften enthalten:

- `age`
- `bytes (L3)`
- `clientIsExternal`
- `dscpName`
- `first`
- `last`
- `pkts`
- `proto`
- `receiverAddr`
- `receiverIsExternal`
- `receiverPort`
- `roundTripTime`

  Die letzte Hin- und Rückflugzeit (RTT) in diesem Fluss. Ein RTT ist die Zeit, die ein Gerät benötigt hat, um ein Paket zu senden und eine sofortige Bestätigung (ACK) zu empfangen.
- `senderAddr`
- `senderIsExternal`
- `senderPort`
- `serverIsExternal`
- `tcpOrigin`

  Dieses Datensatzfeld ist nur enthalten, wenn der Datensatz den von einem Client oder Absendergerät gesendeten Verkehr darstellt.
- `tcpFlags`

`record2:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `Flow.commitRecord2()` auf einem `FLOW_RECORD` Ereignis.

Das Objekt stellt den Verkehr dar, der von einem von zwei Geräten im Datenfluss in eine einzige Richtung gesendet wird. Das andere Gerät wird durch das `record1` Eigentum. Das Gerät, dargestellt durch den `record2` Die Eigenschaft bleibt für den Fluss konstant.

Greifen Sie nur auf das Datensatzobjekt zu `FLOW_RECORD` Ereignisse; andernfalls tritt ein Fehler auf.

Das Standard-Datensatzobjekt kann die folgenden Eigenschaften enthalten:

- `age`
- `bytes (L3)`
- `clientIsExternal`
- `dscpName`
- `first`
- `last`
- `pkts`
- `proto`
- `receiverAddr`

- `receiverIsExternal`
- `receiverPort`
- `roundTripTime`

  Die letzte Hin- und Rückflugzeit (RTT) in diesem Fluss. Ein RTT ist die Zeit, die ein Gerät benötigt hat, um ein Paket zu senden und eine sofortige Bestätigung (ACK) zu empfangen.
- `senderAddr`
- `senderIsExternal`
- `senderPort`
- `serverIsExternal`
- `tcpOrigin`

  Dieses Datensatzfeld ist nur enthalten, wenn der Datensatz den von einem Client oder Absendergerät gesendeten Verkehr darstellt.
- `tcpFlags`

`roundTripTime:` ***Zahl***

Die mittlere Roundtrip-Zeit (RTT) für die Dauer des Ereignis, ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Samples gibt.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`rto1:` ***Zahl***

Die Zahl der Zeitüberschreitungen bei der erneuten Übertragung (RTOs), das einem von zwei Geräten im Fluss zugeordnet ist; das andere Gerät wird dargestellt durch `rto2`. Das Gerät, dargestellt durch `rto1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`rto2:` ***Zahl***

Die Zahl der Zeitüberschreitungen bei der erneuten Übertragung (RTOs), das einem von zwei Geräten im Fluss zugeordnet ist; das andere Gerät wird dargestellt durch `rto1`. Das Gerät, dargestellt durch `rto2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`store:` ***Objekt***

Der Flow-Speicher ist so konzipiert, dass er Objekte im selben Flow von der Anfrage zur Antwort weiterleitet. Das `store` object ist eine Instanz eines leeren JavaScript-Objekts. Objekte können als Eigenschaften an den Speicher angehängt werden, indem der Eigenschaftsschlüssel und der Eigenschaftswert definiert werden. Zum Beispiel:

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

Für Ereignisse , die im selben Fluss auftreten, können Sie den Flow-Store anstelle der Sitzungstabelle verwenden , um Informationen gemeinsam zu nutzen. Zum Beispiel:

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

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

Wichtig: Die Werte Fluss Flow-Speichers bleiben für alle Anfragen und Antworten bestehen, die in diesem Flow ausgeführt werden. Bei der Arbeit mit dem Flow-Store empfiehlt es sich, die Flow-Store-Variable auf zu setzen `null` wenn sein Wert nicht an die nächste Anfrage oder Antwort weitergegeben werden soll. Diese Vorgehensweise hat den zusätzlichen Vorteil, dass der Flow-Store-Speicher geschont wird.

Die meisten Fluss Store-Trigger sollten eine ähnliche Struktur wie das folgende Beispiel haben:

```
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';
                // for example, commit a metric
        }
}
```

Hinweis: Da DHCP-Anfragen häufig in anderen Abläufen als die entsprechenden DHCP-Antworten erfolgen, empfehlen wir, DHCP-Anforderungs- und Antwortinformationen zu kombinieren, indem Sie DHCP-Transaktions-IDs in der Sitzungstabelle speichern. Beispielsweise erstellt der folgende Triggercode eine Metrik, die verfolgt, wie viele DHCP-Discover-Nachrichten eine entsprechende DHCP-Angebotsnachricht erhalten haben:

```
if (event === 'DHCP_REQUEST'){
    var opts = {
        expire: 30
    };
    Session.add(DHCP.txId.toString(), DHCP.msgType, opts);
}
else if (event === 'DHCP_RESPONSE'){
    var reqMsgType = Session.lookup(DHCP.txId.toString());
    if (reqMsgType && DHCP.msgType === 'DHCPOFFER') {
        Device.metricAddCount('dhcp-discover-offer', 1);
    }
}
```

`tcpOrigin:` ***IP-Adresse | Null***

Die ursprüngliche IP-Adresse des Client oder Absenders, wenn sie von einem Netzwerk-Proxy in TCP-Option 28 angegeben wurde.

`vlan:` ***Zahl***

Die dem Fluss zugeordnete VLAN-Nummer. Wenn kein VLAN-Tag vorhanden ist, wird dieser Wert auf gesetzt `0`.

`vxlanVNI:` ***Zahl***

Die dem Fluss zugeordnete VXLAN-Netzwerkkennnummer. Wenn kein VXLAN-Tag vorhanden ist, wird dieser Wert auf gesetzt `NaN` .

`zeroWnd1:` ***Zahl***

Die Anzahl der Nullfenster, die einem von zwei Geräten im Fluss zugeordnet sind; das andere Gerät wird dargestellt durch `zeroWnd2`. Das Gerät, dargestellt durch `zeroWnd1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`zeroWnd2:` ***Zahl***

Die Anzahl der Nullfenster, die einem von zwei Geräten im Fluss zugeordnet sind; das andere Gerät wird dargestellt durch `zeroWnd1`. Das Gerät, dargestellt durch `zeroWnd2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

Beispiele für Trigger

- Beispiel: Überwachen Sie CIFS-Aktionen auf Geräten
- Beispiel: HTTP-Antworten auf 500-Ebene nach Kunden-ID und URI verfolgen
- Beispiel: Analysieren von benutzerdefinierten PoS-Nachrichten mit universeller Nutzlastanalyse
- Beispiel: Syslog über TCP mit universeller Nutzlastanalyse analysieren
- Beispiel: NTP mit universeller Nutzlastanalyse analysieren
- Beispiel: SOAP-Anfragen verfolgen

`FlowInterface`

Das `FlowInterface` Mit der Klasse können Sie Flow-Schnittstellenattribute abrufen und benutzerdefinierte Metriken auf Schnittstellenebene hinzufügen.

Methoden

`FlowInterface(id:` ***Schnur*** `)`

Ein Konstruktor für das FlowInterface-Objekt, der eine Flow-Schnittstellen-ID akzeptiert. Ein Fehler tritt auf, wenn die Flow-Interface-ID auf dem ExtraHop-System nicht existiert.

Instanzmethoden

Mit den Methoden in diesem Abschnitt können Sie benutzerdefinierte Metriken auf einer Flussschnittstelle erstellen. Die Methoden sind nur auf Instanzen von vorhanden NetFlow Klasse. Die folgende Anweisung sammelt beispielsweise Metriken aus dem NetFlow-Verkehr auf der Eingangsschnittstelle:

```
NetFlow.ingressInterface.metricAddCount("slow_rsp", 1);
```

Sie können die FlowInterface-Methode jedoch als statische Methode aufrufen `NETFLOW_RECORD` Ereignisse. Die folgende Anweisung sammelt beispielsweise Metriken aus dem NetFlow-Verkehr sowohl auf der Eingangs- als auch auf der Ausgangsschnittstelle:

```
FlowInterface.metricAddCount("slow_rsp", 1);
```

`metricAddCount(metric_name:` ***Schnur*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Metrik zählen. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der Metrik für die Zählung der obersten Ebene.

`count:` ***Zahl***

Der Inkrementwert. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. Ein `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgende Eigenschaft enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailCount(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Metrik zählen anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der Metrik für die Detailzählung.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`count:` ***Zahl***

Der Inkrementwert. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. Ein `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgende Eigenschaft enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDataset(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Datensatz-Metrik. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der Datensatzmetrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`freq:` ***Zahl***

Eine Option, die es Ihnen ermöglicht, mehrere Vorkommen bestimmter Werte im Datensatz gleichzeitig aufzuzeichnen, wenn sie auf die Anzahl von Vorkommen gesetzt ist, die durch `val` Parameter. Wenn kein Wert angegeben ist, ist der Standardwert 1.

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailDataset(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Datensatz-Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der Metrik für die Detailzählung.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`freq:` ***Zahl***

Eine Option, die es Ihnen ermöglicht, mehrere Vorkommen bestimmter Werte im Datensatz gleichzeitig aufzuzeichnen, wenn sie auf die Anzahl von Vorkommen gesetzt ist, die durch `val` Parameter. Wenn kein Wert angegeben ist, ist der Standardwert 1.

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDistinct(metric_name:` ***Schnur*** `, item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse*** `:void`

Erstellt ein benutzerdefiniertes oberste Ebene unterschiedliche Zählmetrik. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der Metrik für die eindeutige Anzahl auf oberster Ebene.

`item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse***

Der Wert, der in das Set aufgenommen werden soll. Der Wert wird in eine Zeichenfolge umgewandelt, bevor er in den Satz aufgenommen wird.

`metricAddDetailDistinct(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse*** `:void`

Erstellt ein benutzerdefiniertes Detail unterschiedliche Zählmetrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der detaillierten Metrik für unterschiedliche Zählungen.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse***

Der Wert, der in das Set aufgenommen werden soll. Der Wert wird in eine Zeichenfolge umgewandelt, bevor er in den Satz aufgenommen wird.

`metricAddMax(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene maximale Metrik. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der maximalen Metrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailMax(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail maximale Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der maximalen Detailmetrik.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddSampleset(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Stichprobensatz, Metrik. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der Sampleset-Metrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailSampleset(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Stichprobensatz, Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der Detail-Sampleset-Metrik.

`key:` ***Schnur*** `|` ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddSnap(metric_name:` ***Schnur*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Snapshot-Metrik. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der Snapshot-Metrik der obersten Ebene.

`count:` ***Zahl***

Der beobachtete Wert, z. B. aktuell hergestellte Verbindungen. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

highPrecision: ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailSnap(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Snapshot-Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an die angegebene Flussschnittstelle.

`metric_name:` ***Schnur***

Der Name der Detail-Sampleset-Metrik.

`key:` ***Schnur*** `|` ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`count:` ***Zahl***

Der beobachtete Wert, z. B. aktuell hergestellte Verbindungen. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

highPrecision: ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

Instanzeigenschaften

`id:` ***Schnur***

Eine Zeichenfolge, die das Flussschnittstelle eindeutig identifiziert.

`number:` ***Zahl***

Die vom NetFlow-Datensatz gemeldete Flow-Schnittstellennummer.

`FlowNetwork`

Das `FlowNetwork` Mit dieser Klasse können Sie Flow-Netzwerk-Attribute abrufen und benutzerdefinierte Metriken auf Flow-Netzwerkebene hinzufügen.

Methoden

`FlowNetwork(id:` ***Schnur*** `)`

Ein Konstruktor für das FlowNetwork-Objekt, der eine Flow-Netzwerk-ID akzeptiert. Ein Fehler tritt auf, wenn die Flow-Netzwerk-ID auf dem ExtraHop-System nicht existiert.

Instanzmethoden

Mit den Methoden in diesem Abschnitt können Sie benutzerdefinierte Metriken in einem Flussnetz erstellen. Die Methoden sind nur auf Instanzen von vorhanden NetFlow Klasse. Die folgende Anweisung sammelt beispielsweise Metriken aus dem NetFlow-Verkehr in einem einzelnen Netzwerk:

```
NetFlow.network.metricAddCount("slow_rsp", 1);
```

Sie können die FlowNetwork-Methode jedoch als statische Methode aufrufen `NETFLOW_RECORD` Ereignisse. Die folgende Anweisung sammelt beispielsweise Metriken aus dem NetFlow-Verkehr auf beiden Geräten im Flussnetz:

```
FlowNetwork.metricAddCount("slow_rsp", 1);
```

`metricAddCount(metric_name:` ***Schnur*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Metrik zählen. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der Metrik für die Zählung der obersten Ebene.

`count:` ***Zahl***

Der Inkrementwert. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. Ein `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgende Eigenschaft enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailCount(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Metrik zählen anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der Metrik für die Detailzählung.

`key:` ***Schnur | IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`count:` ***Zahl***

Der Inkrementwert. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. Ein `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgende Eigenschaft enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDataset(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Datensatz-Metrik. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der Datensatzmetrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`freq:` ***Zahl***

Eine Option, die es Ihnen ermöglicht, mehrere Vorkommen bestimmter Werte im Datensatz gleichzeitig aufzuzeichnen, wenn sie auf die Anzahl von Vorkommen gesetzt ist, die durch `val` Parameter. Wenn kein Wert angegeben ist, ist der Standardwert 1.

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailDataset(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Datensatz-Metrik anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der Metrik für die Detailzählung.

`key:` ***Schnur | IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`freq:` ***Zahl***

Eine Option, die es Ihnen ermöglicht, mehrere Vorkommen bestimmter Werte im Datensatz gleichzeitig aufzuzeichnen, wenn sie auf die Anzahl von Vorkommen gesetzt ist, die durch `val` Parameter. Wenn kein Wert angegeben ist, ist der Standardwert 1.

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDistinct(metric_name:` ***Schnur*** `, item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse*** `:void`

Erstellt ein benutzerdefiniertes oberste Ebene unterschiedliche Zählmetrik. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der Metrik für die eindeutige Anzahl auf oberster Ebene.

`item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse***

Der Wert, der in das Set aufgenommen werden soll. Der Wert wird in eine Zeichenfolge umgewandelt, bevor er in den Satz aufgenommen wird.

`metricAddDetailDistinct(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse*** `:void`

Erstellt ein benutzerdefiniertes Detail unterschiedliche Zählmetrik anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der detaillierten Metrik für unterschiedliche Zählungen.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse***

Der Wert, der in das Set aufgenommen werden soll. Der Wert wird in eine Zeichenfolge umgewandelt, bevor er in den Satz aufgenommen wird.

`metricAddMax(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene maximale Metrik. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der maximalen Metrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailMax(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail maximale Metrik anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der maximalen Detailmetrik.

`key:` ***Schnur | IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddSampleset(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Stichprobensatz, Metrik. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der Sampleset-Metrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailSampleset(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Stichprobensatz, Metrik anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der Detail-Sampleset-Metrik.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddSnap(metric_name:` ***Schnur*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Snapshot-Metrik. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der Snapshot-Metrik der obersten Ebene.

`count:` ***Zahl***

Der beobachtete Wert, z. B. aktuell hergestellte Verbindungen. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailSnap(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Snapshot-Metrik anhand derer Sie einen Drilldown durchführen können. Überträgt die Metrik Daten in das angegebene Flussnetz.

`metric_name:` ***Schnur***

Der Name der Detail-Sampleset-Metrik.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`count:` ***Zahl***

Der beobachtete Wert, z. B. aktuell hergestellte Verbindungen. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

Instanzeigenschaften

`id:` ***Schnur***

Eine Zeichenfolge, die das Flussnetz eindeutig identifiziert.

`ipaddr:` ***IP-Adresse***

Die IP-Adresse der Verwaltungsschnittstelle im Flussnetz.

`GeoIP`

Die `GeoIP` Mit class können Sie den ungefähren Standort einer bestimmten Adresse auf Landes - oder Stadtebene abrufen.

Methoden

Werte, die von GeoIP-Methoden zurückgegeben werden, stammen aus dem MaxMind GeoLite2 Länderdatenbank oder der MaxMind GeoLite2 City-Datenbank sofern nicht anders konfiguriert von Geomap-Datenquelle Einstellungen in den Administrationseinstellungen.

In den Einstellungen für die Geomap-Datenquelle können Sie benutzerdefinierte Datenbanken hochladen und angeben, auf welche Datenbank standardmäßig für die Suche nach Städten oder Ländern verwiesen werden soll.

Wir empfehlen, nur eine benutzerdefinierte Datenbank auf Stadtebene hochzuladen, wenn Sie beide anrufen möchten `GeoIP.getCountry()` und `GeoIP.getPreciseLocation()` Methoden in Triggern. Wenn beide Arten von benutzerdefinierten Datenbanken hochgeladen werden, ruft das ExtraHop-System Werte für beide Methoden aus der Datenbank auf Stadtebene ab und ignoriert die Datenbank auf Landesebene, die als Teilmenge der Datenbank auf Stadtebene betrachtet wird.

`getCountry(ipaddr:` ***IP-Addresse*** `):` ***Objekt***

Gibt Details auf Landesebene für das angegebene Objekt zurück IPAddress in einem Objekt, das die folgenden Felder enthält:

`continentName:` ***Schnur***

Der Name des Kontinents, wie `Europe`, das dem Land zugeordnet ist, aus dem die angegebene IP-Adresse stammt. Der Wert ist derselbe wie `continentName` Feld zurückgegeben von `getPreciseLocation()` Methode.

`continentCode:` ***Zahl***

Der Code des Kontinents, wie `EU`, das ist verbunden mit dem Wert von `countryCode` Feld, gemäß ISO 3166. Der Wert ist derselbe wie `continentCode` Feld zurückgegeben von `getPreciseLocation()` Methode.

`countryName:` ***Schnur***

Der Name des Landes, aus dem die angegebene IP-Adresse stammt, z. B. `United States`. Der Wert ist derselbe wie `countryName` Feld zurückgegeben von `getPreciseLocation()` Methode.

`countryCode:` ***Schnur***

Der dem Land zugeordnete Code gemäß ISO 3166, z. B. `US`. Der Wert ist derselbe wie `countryCode` Feld zurückgegeben von `getPreciseLocation()` Methode.

Retouren `null` in jedem Feld, für das keine Daten verfügbar sind, oder gibt ein `null` Objekt, wenn alle Felddaten nicht verfügbar sind.

Hinweis: Die `getCountry()` Diese Methode erfordert insgesamt 20 MB RAM auf dem ExtraHop-System, was die Systemleistung beeinträchtigen kann. Wenn diese Methode zum ersten Mal in einem Auslöser aufgerufen wird, reserviert das ExtraHop-System die erforderliche Menge an RAM, es sei denn `getPreciseLocation()` Methode wurde bereits aufgerufen. Die

`getPreciseLocation()` Die Methode benötigt 100 MB RAM, sodass bereits ausreichend RAM für den Aufruf von verfügbar ist `getCountry()` Methode. Die erforderliche RAM-Menge wird nicht pro Auslöser oder Methodenaufruf benötigt; das ExtraHop-System reserviert die benötigte RAM-Menge nur einmal.

Im folgenden Codebeispiel ist der `getCountry()` Methode wird bei jedem angegebenen Ereignis aufgerufen und ruft grobe Standortdaten für jede Client-IP-Adresse ab:

```
// ignore if the IP address is non-routable
if (Flow.client.ipaddr.isRFC1918) return;
var results=GeoIP.getCountry(Flow.client.ipaddr);
if (results) {
    countryCode=results.countryCode;
    // log the 2-letter country code of each IP address
    debug ("Country Code is " + results.countryCode);
}
```

`getPreciseLocation(ipaddr:` ***IP-Address*** `):` ***Objekt***

Gibt Details auf Stadtebene für das angegebene Objekt zurück IPAddress in einem Objekt, das die folgenden Felder enthält:

`continentName:` ***Schnur***

Der Name des Kontinents, wie `Europe`, das dem Land zugeordnet ist, aus dem die angegebene IP-Adresse stammt. Der Wert ist derselbe wie `continentName` Feld zurückgegeben von `getCountry()` Methode.

`continentCode:` ***Zahl***

Der Code des Kontinents, wie EU, das ist verbunden mit dem Wert von `countryCode` Feld, gemäß ISO 3166. Der Wert ist derselbe wie `continentCode` Feld zurückgegeben von `getCountry()` Methode.

`countryName:` ***Schnur***

Der Name des Landes, aus dem die angegebene IP-Adresse stammt, z. B. `United States`. Der Wert ist derselbe wie `countryName` Feld zurückgegeben von `getCountry()` Methode.

`countryCode:` ***Schnur***

Der dem Land zugeordnete Code gemäß ISO 3166, z. B. US. Der Wert ist derselbe wie `countryCode` Feld zurückgegeben von `getCountry()` Methode.

`region:` ***Schnur***

Die Region, z. B. ein Bundesstaat oder eine Provinz, wie `Washington`.

`city:` ***Schnur***

Die Stadt, aus der die IP-Adresse stammt, wie `Seattle`.

`latitude:` ***Zahl***

Der Breitengrad des IP-Adressstandorts.

`longitude:` ***Zahl***

Der Längengrad des Standorts der IP-Adresse.

`radius:` ***Zahl***

Der Radius, ausgedrückt in Kilometern, um die Längen- und Breitengradkoordinaten des Standorts der IP-Adresse.

Retouren `null` in jedem Feld, für das keine Daten verfügbar sind, oder gibt ein `null` Objekt, wenn alle Felddaten nicht verfügbar sind.

Hinweis: Die `getPreciseLocation()` Diese Methode erfordert 100 MB Gesamt-RAM auf dem ExtraHop-System, was die Systemleistung beeinträchtigen kann. Wenn diese Methode zum ersten Mal in einem Auslöser aufgerufen wird, reserviert das ExtraHop-System die erforderliche Menge an RAM, es sei denn `getCountry()` Methode wurde bereits aufgerufen. Die `getCountry()` Diese Methode benötigt

20 MB RAM, sodass das ExtraHop-System weitere 80 MB RAM reserviert. Die erforderliche RAM-Menge wird nicht pro Auslöser oder Methodenaufruf benötigt; das ExtraHop-System reserviert die benötigte RAM-Menge nur einmal.

IPAddress

Das IPAddress Klasse ermöglicht das Abrufen von IP-Adressattributen. Die IPAddress-Klasse ist auch als Eigenschaft für die Flow-Klasse verfügbar.

Methoden

`IPAddress(ip:` ***Schnur*** `|` ***Zahl*** `, mask:` ***Zahl*** `)`

Konstruktor für die Klasse IPAddress, der zwei Parameter akzeptiert:

`ip:` ***Schnur***

Die IP-Adresszeichenfolge im CIDR-Format.

`mask:` ***Zahl***

Die optionale Subnetzmaske in einem numerischen Format, die die Anzahl der „1“ -Bits ganz links in der Maske darstellt (optional).

Instanzmethoden

`equals(equals:` ***IP-Adresse*** `):` ***Boolesch***

Führt einen Gleichheitstest zwischen IPAddress-Objekten durch, wie im folgenden Beispiel gezeigt:

```
if (Flow.client.ipaddr.toString() === "10.10.10.10")
{ // perform a task }
```

`mask(mask:` ***Zahl*** `):` ***IP-Adresse***

Legt die Subnetzmaske des IPAddress-Objekts fest, wie im folgenden Beispiel gezeigt:

```
if ((Flow.ipaddr1.mask(24).toString() === "173.194.33.0")||
(Flow.ipaddr2.mask(24).toString() === "173.194.33.0"))
{Flow.setApplication("My L4 App");}
```

Das `mask` Der Parameter gibt die Subnetzmaske in einem numerischen Format an und stellt die Anzahl der „1“ -Bits ganz links in der Maske dar (optional).

`toJSON():` ***Schnur***

Konvertiert das IPAddress-Objekt in das JSON-Format.

`toString():` ***Schnur***

Konvertiert das IPAddress-Objekt in eine druckbare Zeichenfolge.

Eigenschaften

`hostNames:` ***Reihe von Zeichenketten***

Ein Array von Hostnamen, die der IP-Adresse zugeordnet sind.

`isBroadcast:` ***Boolesch***

Der Wert ist `true` wenn die IP-Adresse eine Broadcast-Adresse ist.

`isExternal:` ***Boolesch***

Der Wert ist `true` wenn die IP-Adresse außerhalb Ihres Netzwerk liegt.

`isLinkLocal:` ***Boolesch***

Der Wert ist `true` wenn es sich bei der IP-Adresse um eine lokale Linkadresse wie (169.254.0.0/16) handelt.

`isMulticast:` ***Boolesch***

Der Wert ist `true` wenn die IP-Adresse eine Multicast-Adresse ist.

`isRFC1918:` ***Boolesch***

Der Wert ist `true` wenn die IP-Adresse zu einem der privaten IP-Bereiche von RFC1918 gehört (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16). Der Wert ist immer `false` für IPv6-Adressen.

`isV4:` ***Boolesch***

Der Wert ist `true` wenn die IP-Adresse eine IPv4-Adresse ist.

`isV6:` ***Boolesch***

Der Wert ist `true` wenn die IP-Adresse eine IPv6-Adresse ist.

`localityName:` ***Schnur*** | ***null***

Der Name der Netzwerklokalität, in der sich die IP-Adresse befindet. Wenn sich die IP-Adresse in keiner Netzwerklokalität befindet, ist der Wert Null.

`Network`

Das `Network` Mit dieser Klasse können Sie benutzerdefinierte Metriken auf globaler Ebene hinzufügen.

Methoden

`metricAddCount(metric_name:` ***Schnur*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Metrik zählen. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der Metrik für die Zählung der obersten Ebene.

`count:` ***Zahl***

Der Inkrementwert. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. Ein `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgende Eigenschaft enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailCount(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Metrik zählen anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der Metrik für die Detailanzahl.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`count:` ***Zahl***

Der Inkrementwert. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. Ein `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgende Eigenschaft enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDataset(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Datensatz-Metrik. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der Datensatzmetrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`freq:` ***Zahl***

Eine Option, mit der Sie mehrere Vorkommen bestimmter Werte im Datensatz gleichzeitig aufzeichnen können, wenn sie auf die Anzahl von Vorkommen gesetzt ist, die durch die `val` Parameter. Wenn kein Wert angegeben ist, ist der Standardwert 1.

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailDataset(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Datensatz-Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der Metrik für die Detailanzahl.

`key:` ***Schnur*** `|` ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`freq:` ***Zahl***

Eine Option, mit der Sie mehrere Vorkommen bestimmter Werte im Datensatz gleichzeitig aufzeichnen können, wenn sie auf die Anzahl von Vorkommen gesetzt ist, die durch die `val` Parameter. Wenn kein Wert angegeben ist, ist der Standardwert 1.

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDistinct(metric_name:` ***Schnur*** `, item:` ***Zahl*** `|` ***Schnur*** `|` ***IP-Adresse*** `:void`

Erstellt ein benutzerdefiniertes oberste Ebene unterschiedliche Zählmetrik. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der Metrik für die eindeutige Anzahl auf oberster Ebene.

`item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse***

Der Wert, der in das Set aufgenommen werden soll. Der Wert wird in eine Zeichenfolge umgewandelt, bevor er in den Satz aufgenommen wird.

MetricAddDetailDistinct (metrischer Name: *Schnur*, Schlüssel: *Schnur | IP-Adresse*, artikel: *Zahl | Schnur | IP-Adresse*: nichtig

Erstellt ein benutzerdefiniertes Detail unterschiedliche Zählmetrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der detaillierten Metrik für unterschiedliche Zählungen.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`item:` ***Zahl*** | ***Schnur*** | ***IP-Adresse***

Der Wert, der in das Set aufgenommen werden soll. Der Wert wird in eine Zeichenfolge umgewandelt, bevor er in den Satz aufgenommen wird.

`metricAddMax(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene maximale Metrik. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der maximalen Metrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailMax(metric_name:` ***Schnur*** `, key:` ***Schnur*** | ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail maximale Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der maximalen Detailmetrik.

`key:` ***Schnur | IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddSampleset(metric_name:` ***Schnur*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Stichprobensatz, Metrik. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der Sampleset-Metrik der obersten Ebene.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailSampleset(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, val:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Stichprobensatz, Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der Detail-Sampleset-Metrik.

`key:` ***Schnur | IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`val:` ***Zahl***

Der beobachtete Wert, z. B. eine Verarbeitungszeit. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddSnap(metric_name:` ***Schnur*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes oberste Ebene Snapshot-Metrik. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der Snapshot-Metrik der obersten Ebene.

`count:` ***Zahl***

Der beobachtete Wert, z. B. aktuell hergestellte Verbindungen. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

`metricAddDetailSnap(metric_name:` ***Schnur*** `, key:` ***Schnur*** `|` ***IP-Adresse*** `, count:` ***Zahl*** `, options:` ***Objekt*** `):void`

Erstellt ein benutzerdefiniertes Detail Snapshot-Metrik anhand derer Sie einen Drilldown durchführen können. Übergibt die Metrikdaten an das angegebene Netzwerk.

`metric_name:` ***Schnur***

Der Name der Detail-Sampleset-Metrik.

`key:` ***Schnur*** | ***IP-Adresse***

Der für die Detail-Metrik angegebene Schlüssel. EIN `null` Der Wert wird stillschweigend verworfen.

`count:` ***Zahl***

Der beobachtete Wert, z. B. aktuell hergestellte Verbindungen. Muss eine 64-Bit-Ganzzahl ungleich Null mit positivem Vorzeichen sein. EIN `NaN` Der Wert wird stillschweigend verworfen.

`options:` ***Objekt***

Ein optionales Objekt, das die folgenden Eigenschaften enthalten kann:

`highPrecision:` ***Boolesch***

Ein Flag, das eine Granularität von einer Sekunde für die benutzerdefinierte Metrik aktiviert, wenn es auf gesetzt ist `true`.

Beispiele für Trigger

- Beispiel: Syslog über TCP mit universeller Nutzlastanalyse analysieren
- Beispiel: Daten in einer Sitzungstabelle aufzeichnen
- Beispiel: SOAP-Anfragen verfolgen

`Session`

Die `Session` Die Klasse bietet Zugriff auf die Sitzungstabelle. Es wurde entwickelt , um die Koordination mehrerer unabhängig voneinander ausgeführter Trigger zu unterstützen. Der globale Status der Sitzungstabelle bedeutet, dass alle Änderungen durch einen Auslöser oder externen Prozess für alle anderen Benutzer der Sitzungstabelle sichtbar werden. Da sich die Sitzungstabelle im Arbeitsspeicher befindet, werden Änderungen nicht gespeichert, wenn Sie das ExtraHop-System oder den Capture-Prozess neu starten.

Hier sind einige wichtige Dinge, die Sie über Sitzungstabellen wissen sollten:

- Die Sitzungstabelle unterstützt normale JavaScript-Werte, sodass Sie der Tabelle JS-Objekte hinzufügen können.
- Sitzungs-Tabelleneinträge können entfernt werden, wenn die Tabelle zu groß wird oder wenn das konfigurierte Ablaufdatum erreicht ist.
- Weil der Sitzungstisch auf einem Sensor wird nicht geteilt mit Konsole, die Werte in der Sitzungstabelle werden nicht mit anderen verbundenen Benutzern geteilt Sensoren.
- Die ExtraHop Open Data Context API macht die Sitzungstabelle über das Verwaltungsnetzwerk verfügbar und ermöglicht so die Koordination mit externen Prozessen über memcache protokoll.

Ereignisse

Die Session-Klasse ist nicht nur beschränkt auf `SESSION_EXPIRE` Ereignis. Sie können die Session-Klasse auf jedes ExtraHop-Ereignis anwenden.

`SESSION_EXPIRE`

Wird regelmäßig (in Schritten von etwa 30 Sekunden) ausgeführt, solange die Sitzungstabelle verwendet wird. Wenn der `SESSION_EXPIRE` Ereignis werden ausgelöst, Schlüssel, die im letzten 30-Sekunden-Intervall abgelaufen sind, sind verfügbar über `Session.expiredKeys` Eigentum.

Die `SESSION_EXPIRE` Das Ereignis ist keinem bestimmten Fluss zugeordnet, wird also ausgelöst `SESSION_EXPIRE` Ereignisse können Gerätemetriken nicht übertragen `Device.metricAdd*()` Methoden oder `Flow.client.device.metricAdd*()` Methoden. Um Gerätekennzahlen für dieses Ereignis zu bestätigen, müssen Sie Folgendes hinzufügen Device Objekte in die Sitzungstabelle über `Device()` Instanzmethode.

Hinweis Sie können Trigger, die nur bei diesem Ereignis ausgeführt werden, nicht bestimmten Geräten oder Gerätegruppen zuweisen. Trigger, die bei diesem Ereignis ausgeführt werden, werden immer dann ausgeführt, wenn dieses Ereignis eintritt.

`TIMER_30SEC`

Läuft genau alle 30 Sekunden. Dieses Ereignis ermöglicht es Ihnen, periodische Verarbeitungen durchzuführen, z. B. regelmäßig auf Session-Tabelleneinträge zuzugreifen, die über die Öffnen Sie die Datenkontext-API.

Hinweis Sie können eine beliebige Triggerklasse auf das TIMER_30SEC-Ereignis anwenden.

Hinweis Sie können Trigger, die nur bei diesem Ereignis ausgeführt werden, nicht bestimmten Geräten oder Gerätegruppen zuweisen. Trigger, die bei diesem Ereignis ausgeführt werden, werden immer dann ausgeführt, wenn dieses Ereignis eintritt.

Methoden

`add(key: `***Schnur***`, value*, options: `***Objekt***`): *`

Fügt den angegebenen Schlüssel in die Sitzungstabelle ein. Wenn der Schlüssel vorhanden ist, wird der entsprechende Wert zurückgegeben, ohne den Schlüsseleintrag in der Tabelle zu ändern. Wenn der Schlüssel nicht vorhanden ist, wird ein neuer Eintrag für den Schlüssel und den Wert erstellt, und der neue Wert wird zurückgegeben.

Sie können ein optionales Optionen Objekt für den angegebenen Schlüssel.

`getOptions(key: `***Schnur***`): `***Objekt***

Gibt den Optionen Objekt für den angegebenen Schlüssel. Sie konfigurieren Optionen bei Aufrufen von `Session.add()`, `Session.modify()`, oder `Session.replace()`.

`increment(key: `***Schnur***`, count: `***Zahl***`): `***Zahl***` | `***null***

Sucht nach dem angegebenen Schlüssel und erhöht den Schlüsselwert um die angegebene Zahl. Der Standardwert des optionalen Zählparameters ist 1. Gibt den neuen Schlüsselwert zurück, wenn der Aufruf erfolgreich ist. Retouren `null` wenn die Suche fehlschlägt. Gibt einen Fehler zurück , wenn der Schlüsselwert keine Zahl ist.

`lookup(key: `***Schnur***`): *`

Sucht den angegebenen Schlüssel in der Sitzungstabelle und gibt den entsprechenden Wert zurück. Retouren `null` wenn der Schlüssel nicht vorhanden ist.

`modify(key: `***Schnur***`, value: *, options: `***Objekt***`): *`

Ändert den angegebenen Schlüsselwert, wenn der Schlüssel in der Sitzungstabelle vorhanden ist, und gibt den vorherigen Wert zurück. Wenn der Schlüssel nicht vorhanden ist, wird kein neuer Eintrag erstellt.

Bei Änderungen zum optionalen Optionen Objekte sind enthalten, die wichtigsten Optionen werden aktualisiert. und alte Optionen werden mit neuen zusammengeführt. Wenn der `expire` Option wurde geändert, der Ablauftimer wurde zurückgesetzt.

`remove(key: `***Schnur***`): *`

Entfernt den Eintrag für den angegebenen Schlüssel und gibt den zugehörigen Wert zurück.

`replace(key: `***Schnur***`, value: *, options: `***Objekt***`): *`

Aktualisiert den Eintrag, der dem angegebenen Schlüssel zugeordnet ist. Wenn der Schlüssel vorhanden ist, aktualisieren Sie den Wert und geben Sie den vorherigen Wert zurück. Wenn der Schlüssel nicht vorhanden ist, fügen Sie den Eintrag hinzu und geben Sie den vorherigen Wert (Null) zurück.

Bei Änderungen zum optionalen Optionen Objekte sind enthalten, die wichtigsten Optionen werden aktualisiert und alte Optionen werden mit neuen zusammengeführt. Wenn der `expire` Option ist vorhanden, der Ablauftimer wird zurückgesetzt.

Optionen

`expire:` ***Zahl***

Die Dauer, nach der die Räumung erfolgt, ausgedrückt in Sekunden. Wenn der Wert `null` oder `undefined`, wird der Eintrag nur entfernt, wenn die Sitzungstabelle zu groß wird.

`notify:` ***Boolescher Wert***

Zeigt an, ob der Schlüssel verfügbar ist am `SESSION_EXPIRE` Ereignisse. Der Standardwert ist False.

`priority:` ***Schnur***

Prioritätsstufe, die bestimmt, welche Einträge entfernt werden sollen, wenn die Sitzungstabelle zu groß wird. Gültige Werte sind `PRIORITY_LOW`, `PRIORITY_NORMAL`, und `PRIORITY_HIGH`. Der Standardwert ist `PRIORITY_NORMAL`.

Konstanten

`PRIORITY_LOW:` ***Zahl***

Die numerische Darstellung der niedrigsten Prioritätsstufe. Der Wert ist 0. Prioritätsstufen bestimmen die Reihenfolge, in der Einträge aus der Sitzungstabelle entfernt werden, wenn die Tabelle zu groß wird.

`PRIORITY_NORMAL:` ***Zahl***

Die numerische Darstellung der Standardprioritätsstufe. Der Wert ist 1. Prioritätsstufen bestimmen die Reihenfolge, in der Einträge aus der Sitzungstabelle entfernt werden, wenn die Tabelle zu groß wird.

`PRIORITY_HIGH:` ***Zahl***

Die numerische Darstellung der höchsten Prioritätsstufe. Der Wert ist 2. Prioritätsstufen bestimmen die Reihenfolge, in der Einträge aus der Sitzungstabelle entfernt werden, wenn die Tabelle zu groß wird.

Eigenschaften

`expiredKeys:` ***Reihe***

Eine Reihe von Objekten mit den folgenden Eigenschaften:

`age:` ***Zahl***

Das Alter des abgelaufenen Objekts, ausgedrückt in Millisekunden. Alter ist die Zeitspanne, die zwischen dem Hinzufügen des Objekts in der Sitzungstabelle oder der Änderung der Ablaufoption des Objekts verstrichen ist, und `SESSION_EXPIRE` Ereignis. Das Alter bestimmt, ob der Schlüssel entfernt wurde oder abgelaufen ist.

`name:` ***Schnur***

Der Schlüssel des abgelaufenen Objekts.

`value:` ***Zahl*** | ***Schnur*** | ***IP-Adresse*** | ***Boolescher Wert*** | ***Gerät***

Der Wert des Eintrags in der Sitzungstabelle.

Zu den abgelaufenen Schlüsseln gehören Schlüssel, die gelöscht wurden, weil die Tabelle zu groß wurde.

Die `expiredKeys` Auf die Immobilie kann nur zugegriffen werden auf `SESSION_EXPIRE` Ereignisse; andernfalls tritt ein Fehler auf.

Beispiele für Trigger

- Beispiel: Daten in einer Sitzungstabelle aufzeichnen

System

Die System Klasse ermöglicht das Abrufen von Informationen über Sensor oder Konsole auf dem ein Auslöser läuft. Diese Information ist nützlich in Umgebungen mit mehreren Sensoren.

Eigenschaften

uuid: ***Schnur***

Der Universally Unique Identifier (UUID) des Sensor oder Konsole.

ipaddr: ***IP-Addresse***

Die IPAddress Objekt der primären Verwaltungsschnittstelle (Interface 1) auf dem Sensor.

hostname: ***Schnur***

Der Hostname für den Sensor oder Konsole in den Administrationseinstellungen konfiguriert.

version: ***Schnur***

Die Firmware-Version läuft auf dem Sensor oder Konsole.

ThreatIntel

Die ThreatIntel Mit dieser Klasse können Sie sehen, ob Bedrohungen für IP-Adressen, Hostnamen oder URIs gefunden wurden. (Nur ExtraHop Reveal (x) Premium und Ultra)

Methoden

hasIP(address: ***IP-Addresse***): ***boolesch***

Der Wert ist true ob die Bedrohungen für die angegebene IP-Adresse gefunden wurden. Wenn im ExtraHop-System keine nachrichtendienstlichen Informationen verfügbar sind, ist der Wert null.

hasDomain(domain: ***Schnur***): ***boolesch***

Der Wert ist true ob die Bedrohungen für die angegebene Domain gefunden wurden. Wenn im ExtraHop-System keine nachrichtendienstlichen Informationen verfügbar sind, ist der Wert null.

hasURI(uri: ***Schnur***): ***boolesch***

Der Wert ist true ob die Bedrohungen für den angegebenen URI gefunden wurden. Wenn im ExtraHop-System keine nachrichtendienstlichen Informationen verfügbar sind, ist der Wert null.

Eigenschaften

isAvailable: ***boolesch***

Der Wert ist true wenn Bedrohungsinformationen auf dem ExtraHop-System verfügbar sind.

Trigger

Die Trigger Mit dieser Klasse können Sie auf Details zu einem laufenden Auslöser zugreifen.

Eigenschaften

isDebugEnabled: ***boolesch***

Der Wert ist true wenn das Debugging für den Auslöser aktiviert ist. Der Wert wird durch den Zustand des bestimmt **Debug-Log aktivieren** Checkbox im Bereich Trigger bearbeiten im ExtraHop-System.

VLAN

Die VLAN Klasse steht für ein VLAN im Netzwerk.

Eigenschaften der Instanz

`id:` ***Zahl***

Die numerische ID für ein VLAN.

Protokoll- und Netzwerkdatenklassen

Mit den Trigger-API-Klassen in diesem Abschnitt können Sie auf Eigenschaften zugreifen und Metriken Datensatz von Protokoll, Nachrichten und Flow-Aktivitäten, die auf dem ExtraHop ExtraHop-System auftreten.

Klasse	Beschreibung
AAA	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf AAA_REQUEST oder AAA_RESPONSE Ereignisse.
ActiveMQ	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf ACTIVEMQ_MESSAGE Ereignisse.
AJP	Mit der AJP-Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen AJP_REQUEST und AJP_RESPONSE Ereignisse.
CDP	Die CDP-Klasse ermöglicht es Ihnen, Metriken zu speichern und auf Eigenschaften zuzugreifen CDP_FRAME Ereignisse.
CIFS	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf CIFS_REQUEST und CIFS_RESPONSE Ereignisse.
DB	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf DB_REQUEST und DB_RESPONSE Ereignisse.
DHCP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf DHCP_REQUEST und DHCP_RESPONSE Ereignisse.
DICOM	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf DICOM_REQUEST und DICOM_RESPONSE Ereignisse.
DNS	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf DNS_REQUEST und DNS_RESPONSE Ereignisse.
FIX	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf FIX_REQUEST und FIX_RESPONSE Ereignisse.
FTP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf FTP_REQUEST und FTP_RESPONSE Ereignisse.
HL7	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf HL7_REQUEST und HL7_RESPONSE Ereignisse.
HTTP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf HTTP_REQUEST und HTTP_RESPONSE Ereignisse.

Klasse	Beschreibung
IBMMQ	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `IBMMQ_REQUEST` und `IBMMQ_RESPONSE` Ereignisse.
ICA	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `ICA_OPEN`, `ICA_AUTH`, `ICA_TICK`, und `ICA_CLOSE` Ereignisse.
ICMP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `ICMP_MESSAGE` Ereignisse.
Kerberos	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `KERBEROS_REQUEST` und `KERBEROS_RESPONSE` Ereignisse.
LDAP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `LDAP_REQUEST` und `LDAP_RESPONSE` Ereignisse.
LLDP	Ermöglicht den Zugriff auf Eigenschaften von `LLDP_FRAME` Ereignisse.
Memcache	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `MEMCACHE_REQUEST` und `MEMCACHE_RESPONSE` Ereignisse.
Modbus	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `MODBUS_REQUEST` und `MODBUS_RESPONSE` Ereignisse.
MongoDB	Mit der MongoDB-Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `MONGODB_REQUEST` und `MONGODB_RESPONSE` Ereignisse.
MSMQ	Mit der MSMQ-Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `MSMQ_MESSAGE` Ereignis.
NetFlow	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `NETFLOW_RECORD` Ereignisse.
NFS	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `NFS_REQUEST` und `NFS_RESPONSE` Ereignisse.
NTLM	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `NTLM_MESSAGE` Ereignisse.
POP3	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf `POP3_REQUEST` und `POP3_RESPONSE` Ereignisse.

Klasse	Beschreibung
RDP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf RDP_OPEN, RDP_CLOSE, und RDP_TICK Ereignisse.
Redis	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf REDIS_REQUEST und REDIS_RESPONSE Ereignisse.
RPC	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf RPC_REQUEST und RPC_RESPONSE Ereignisse.
RTCP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf RTCP_MESSAGE Ereignisse.
RTP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf RTP_OPEN, RTP_CLOSE, und RTP_TICK Ereignisse.
SCCP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf SCCP_MESSAGE Ereignisse.
SDP	Ermöglicht den Zugriff auf Eigenschaften von SIP_REQUEST und SIP_RESPONSE Ereignisse.
SFlow	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf SFLOW_RECORD Ereignisse.
SIP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf SIP_REQUEST und SIP_RESPONSE Ereignisse.
SMPP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf SMPP_REQUEST und SMPP_RESPONSE Ereignisse.
SMTP	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf SMTP_REQUEST und SMTP_RESPONSE Ereignisse.
SSH	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf SSH_CLOSE, SSH_OPEN und SSH_TICK Ereignisse.
SSL	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf SSL_OPEN, SSL_CLOSE, SSL_ALERT, SSL_RECORD, SSL_HEARTBEAT, und SSL_RENEGOTIATE Ereignisse.
TCP	Ermöglicht den Zugriff auf Eigenschaften und das Abrufen von Metriken aus TCP-Ereignissen und mehr FLOW_TICK und FLOW_TURN Ereignisse.
Telnet	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf TELNET_MESSAGE Ereignisse.

Klasse	Beschreibung
Turn	Ermöglicht das Speichern von Metriken und Zugreifen auf Eigenschaften auf FLOW_TURN Ereignisse.
UDP	Ermöglicht den Zugriff auf Eigenschaften und das Abrufen von Metriken aus UDP-Ereignissen und mehr FLOW_TICK und FLOW_TURN Ereignisse.
WebSocket	Ermöglicht den Zugriff auf Eigenschaften von WEBSOCKET_OPEN, WEBSOCKET_CLOSE, und WEBSOCKET_MESSAGE Ereignisse.

AAA

Die AAA Mit der Klasse (AAA) können Sie Metriken speichern und auf Eigenschaften zugreifen AAA_REQUEST oder AAA_RESPONSE Ereignisse.

Ereignisse

AAA_REQUEST

Wird ausgeführt, wenn das ExtraHop-System die Verarbeitung einer AAA-Anfrage abgeschlossen hat.

AAA_RESPONSE

Läuft mit jeder AAA-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord()`: ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem AAA_REQUEST oder AAA_RESPONSE Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die Standardeigenschaften, die für jedes Ereignis übernommen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`authenticator`: ***Schnur***

Der Wert des Authenticator-Felds (nur RADIUS).

`avps`: ***Reihe***

Ein Array von AVP-Objekten mit den folgenden Eigenschaften:

`avpLength`: ***Zahl***

Die Größe des AVP, ausgedrückt in Byte. Dieser Wert beinhaltet die AVP-Header-Daten sowie den Wert.

`id`: ***Zahl***

Die numerische ID des Attributs, dargestellt als Ganzzahl.

`isGrouped`: ***Boolescher Wert***

Der Wert ist `true` wenn es sich um ein gruppiertes AVP handelt (nur Durchmesser).

`name`: ***Schnur***

Der Name für das angegebene AVP.

`vendor:` ***Schnur***

Der Anbietername für Hersteller-AVPs (nur Diameter).

`value:` ***Schnur*** | ***Reihe*** | ***Zahl***

Für einzelne AVPs eine Zeichenfolge oder ein numerischer Wert. Für gruppierte AVPs (nur Durchmesser) eine Reihe von Objekten.

`isDiameter:` ***Boolescher Wert***

Der Wert ist `true` wenn die Anfrage oder Antwort Durchmesser lautet.

`isError:` ***Boolescher Wert***

Der Wert ist `true` wenn die Antwort ein Fehler ist. Um die Fehlerdetails in Diameter abzurufen, überprüfen Sie `AAA.statusCode`. Um die Fehlerdetails in RADIUS abzurufen, überprüfen Sie das AVP mit dem Code 18 (Reply-Message).

Zugriff nur auf `AAA_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isRadius:` ***Boolescher Wert***

Der Wert ist `true` wenn die Anfrage oder Antwort RADIUS ist.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn der `AAA_RESPONSE` Ereignis wurde abgebrochen.

Zugriff nur auf `AAA_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`method:` ***Zahl***

Die Methode, die dem Befehlscode in RADIUS oder Diameter entspricht.

Die folgende Tabelle enthält gültige Diameter-Befehlscodes:

Name des Befehls	Abbrr.	Kode
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-Answer	RAA	258
Re-Auth-Request	RAR	258
Session-Termination-Request	STR	275

Name des Befehls	Abbrr.	Kode
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
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

Die folgende Tabelle enthält gültige RADIUS-Befehlscodes:

Name des Befehls	Kode
Access-Request	1

Name des Befehls	Kode
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

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn das Timing ungültig ist.

Zugriff nur auf AAA_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann AAA.`commitRecord()` entweder auf einem AAA_REQUEST oder AAA_RESPONSE Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

AAA_ANFRAGE	AAA_RESPONSE
authenticator	authenticator
clientIsExternal	clientIsExternal
clientZeroWnd	clientZeroWnd
method	isError
receiverIsExternal	isRspAborted
reqBytes	method
reqL2Bytes	processingTime
reqPkts	receiverIsExternal
reqRTO	roundTripTime
senderIsExternal	rspBytes
serverIsExternal	rspL2Bytes
serverZeroWnd	rspPkts
txId	rspRTO
	statusCode
	senderIsExternal
	serverIsExternal
	serverZeroWnd

AAA_ANFRAGE	AAA_RESPONSE
	txId

`reqBytes:` ***Zahl***
Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

`reqL2Bytes:` ***Zahl***
Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Headern.

`reqPkts:` ***Zahl***
Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***
Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf `AAA_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqZeroWnd:` ***Zahl***
Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***
Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`rspBytes:` ***Zahl***
Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

`rspL2Bytes:` ***Zahl***
Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

`rspPkts:` ***Zahl***
Die Anzahl der Antwortpakete.

`rspRTO:` ***Zahl***
Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf `AAA_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***
Die Anzahl der Nullfenster in der Antwort.

`statusCode:` ***Schnur***
Eine Zeichenkettendarstellung der AVP-ID 268 (Ergebniscode).

Zugriff nur auf `AAA_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`txId:` ***Zahl***
Ein Wert, der dem Hop-By-Hop-Bezeichner in Diameter und der msg-id in RADIUS entspricht.

`ActiveMQ`

Das ActiveMQ Klasse ermöglicht es Ihnen, Metriken zu speichern und auf Eigenschaften zuzugreifen `ACTIVEMQ_MESSAGE` Ereignisse. ActiveMQ ist eine Implementierung des Java Messaging Service (JMS).

Ereignisse

`ACTIVEMQ_MESSAGE`
Läuft auf jeder JMS-Nachricht, die vom Gerät verarbeitet wird.

Methoden

`commitRecord()`: ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `ACTIVEMQ_MESSAGE` Ereignis.

Informationen zu den Standardeigenschaften, die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`correlationId`: ***Schnur***

Das jmsCorrelationId-Feld der Nachricht.

`exceptionResponse`: ***Objekt | Null***

Das JMSException-Feld der Nachricht. Wenn der Befehl der Nachricht nicht `ExceptionResponse`, der Wert ist Null. Das Objekt enthält die folgenden Felder:

`message`: ***Schnur***

Die Ausnahmeantwortnachricht.

`class`: ***Schnur***

Die Unterklasse der JMSException.

`expiration`: ***Zahl***

Das JMSExpiration-Feld der Nachricht.

`msg`: ***Puffer***

Der Nachrichtentext. Bei Nachrichten im TEXT_MESSAGE-Format wird der Nachrichtentext als UTF-8-Zeichenfolge zurückgegeben. Für alle anderen Nachrichtenformate gibt dies die Rohbytes zurück.

`msgFormat`: ***Schnur***

Das Nachrichtenformat. Mögliche Werte sind:

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

`msgId`: ***Schnur***

Das JMSMessageID-Feld der Nachricht.

`persistent`: ***Boolesch***

Der Wert ist `true` wenn der JMSDeliveryMode PERSISTENT ist.

`priority`: ***Zahl***

Das JMSPriority-Feld der Nachricht.

- 0 ist die niedrigste Priorität.
- 9 hat die höchste Priorität.
- 0-4 sind Abstufungen mit normaler Priorität.
- 5-9 sind Abstufungen mit beschleunigter Priorität.

`properties`: ***Objekt***

Null oder mehr Eigenschaften, die an die Nachricht angehängt sind. Die Schlüssel sind beliebige Zeichenketten und die Werte können boolesche Werte, Zahlen oder Zeichenketten sein.

`queue:` ***Schnur***

Das JmsDestination-Feld der Nachricht.

`receiverBytes:` ***Zahl***

Die Anzahl der Bytes auf Anwendungsebene vom Empfänger.

`receiverIsBroker:` ***Boolesch***

Der Wert ist `true` wenn der Empfänger der Nachricht auf Flow-Level ein Broker ist.

`receiverL2Bytes:` ***Zahl***

Die Zahl der L2 Byte vom Empfänger.

`receiverPkts:` ***Zahl***

Die Anzahl der Pakete vom Empfänger.

`receiverRTO:` ***Zahl***

Die Anzahl der RTOs vom Empfänger.

`receiverZeroWnd:` ***Zahl***

Die Anzahl der vom Empfänger gesendeten Nullfenster.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `ActiveMQ.commitRecord()` auf einem `ACTIVEMQ_MESSAGE` Ereignis.

Das Standard-Datensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`
- `correlationId`
- `expiration`
- `msgFormat`
- `msgId`
- `persistent`
- `priority`
- `queue`
- `receiverBytes`
- `receiverIsBroker`
- `receiverIsExternal`
- `receiverL2Bytes`
- `receiverPkts`
- `receiverRTO`
- `receiverZeroWnd`
- `redeliveryCount`
- `replyTo`
- `roundTripTime`
- `senderBytes`
- `senderIsBroker`
- `senderIsExternal`
- `senderL2Bytes`
- `senderPkts`
- `senderRTO`
- `senderZeroWnd`
- `serverIsExternal`
- `timeStamp`
- `totalMsgLength`

`redeliveryCount:` ***Zahl***

Die Anzahl der Rücklieferungen.

`replyTo:` ***Schnur***

Das JmsReplyTo-Feld der Nachricht, umgewandelt in eine Zeichenfolge.

`roundTripTime:` ***Zahl***

Die mittlere Roundtrip-Zeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Samples gibt.

`senderBytes:` ***Zahl***

Die Anzahl der Byte auf Anwendungsebene vom Absender.

`senderIsBroker:` ***Boolesch***

Der Wert ist `true` wenn der Absender der Nachricht auf Flow-Ebene ein Broker ist.

`senderL2Bytes:` ***Zahl***

Die Zahl der L2 Byte vom Absender.

`senderPkts:` ***Zahl***

Die Anzahl der Pakete vom Absender.

`senderRTO:` ***Zahl***

Die Anzahl der RTOs des Absenders.

`senderZeroWnd:` ***Zahl***

Die Anzahl der vom Absender gesendeten Nullfenster.

`timestamp:` ***Zahl***

Der Zeitpunkt, zu dem die Nachricht zum Versand an einen Anbieter übergeben wurde, ausgedrückt in GMT. Dies ist das jmsTimestamp-Feld der Nachricht.

`totalMsgLength:` ***Zahl***

Die Länge der Nachricht, ausgedrückt in Byte.

AJP

Das Apache JServ Protocol (AJP) leitet eingehende Anfragen von einem Webserver an einen Anwendungsserver weiter und wird häufig in Umgebungen mit Lastenausgleich eingesetzt, in denen ein oder mehrere Frontend-Webserver Anfragen an einen oder mehrere Anwendungsserver weiterleiten. Die `AJP` Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `AJP_REQUEST` und `AJP_RESPONSE` Ereignisse.

Ereignisse

`AJP_REQUEST`

Wird ausgeführt, nachdem der Server eine AJP-Forward-Request-Nachricht an einen Servlet-Container gesendet hat und anschließend alle nachfolgenden Anforderungstexte übertragen hat.

`AJP_RESPONSE`

Wird ausgeführt, nachdem ein Servlet-Container eine AJP End Response-Nachricht gesendet hat, um zu signalisieren, dass der Servlet-Container die Verarbeitung einer AJP-Forward-Anfrage abgeschlossen und die angeforderten Informationen zurückgesendet hat.

Methoden

`commitRecord():` ***Leer***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `AJP_RESPONSE` Ereignis. Commits aufzeichnen für `AJP_REQUEST` Ereignisse werden nicht unterstützt.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

`findHeaders(name:` ***Schnur*** `):` ***Reihe***

Greift auf AJP-Header-Werte zu und gibt ein Array von Header-Objekten (mit Namens- und Werteigenschaften) zurück, wobei die Namen mit dem Präfix der angegebenen Zeichenfolge übereinstimmen. Greift auf Anforderungsheader zu `AJP_REQUEST` Ereignisse und Antwortheader auf `AJP_RESPONSE` Anfragen.

Eigenschaften

`attributes:` ***Objekt***

Ein Array optionaler AJP-Attribute, die mit der Anfrage gesendet werden, wie remote_user, auth_type, query_string, jvm_route, ssl_cert, ssl_cipher und ssl_session.

Zugriff nur auf `AJP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`fwdReqClientAddr:` ***IP-Adresse***

Die IPAddress des HTTP-Clients, der die ursprüngliche Anfrage an den Server gestellt hat. Der Wert ist `null` wenn die verfügbaren Informationen nicht auf eine IP-Adresse geparst werden können.

`fwdReqHost:` ***Schnur***

Der HTTP-Host, der vom HTTP-Client angegeben wurde, der die ursprüngliche Anfrage an den Server gestellt hat.

`fwdReqIsEncrypted:` ***Boolescher Wert***

Der Wert ist `true` wenn die SSL-Verschlüsselung vom HTTP-Client angewendet wurde , der die ursprüngliche Anfrage an den Server gestellt hat.

`fwdReqServerName:` ***Schnur***

Der Name des Server, an den der HTTP-Client die ursprüngliche Anfrage gestellt hat.

FWDREQ-Serverport: ***Zahl***

Der TCP-Port auf dem Server, an den der HTTP-Client die ursprüngliche Anfrage gestellt hat.

`headers:` ***Objekt***

Bei Zugriff am `AJP_REQUEST` events, ein Array von Header-Namen und Werten, die mit der Anfrage gesendet werden.

Bei Zugriff am `AJP_RESPONSE` Ereignisse, eine Reihe von Headern, die in der AJP Send Headers-Nachricht vom Server an den Browser des Endbenutzers übermittelt werden.

`method:` ***Schnur***

Die HTTP-Methode der Anfrage, z. B. POST oder GET, vom Server an den Servlet-Container.

`processingTime:` ***Zahl***

Die Zeit zwischen dem letzten Byte der empfangenen Anfrage und dem ersten Byte der gesendeten Antwort-Nutzlast, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `AJP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`protocol:` ***Schnur***

Das Protokoll der Anfrage vom Server an den Servlet-Container. Nicht für andere Nachrichtentypen festgelegt.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `AJP.commitRecord()` auf einem `AJP_RESPONSE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- clientIsExternal
- fwdReqClientAddr
- fwdReqHost
- fwdReqIsEncrypted
- fwdReqServerName
- fwdReqServerPort
- method
- processingTime
- protocol
- receiverIsExternal
- reqSize
- rspSize
- statusCode
- senderIsExternal
- serverIsExternal
- uri

Zugriff nur auf AJP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf AJP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

`reqSize:` ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne AJP-Header.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf AJP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf AJP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

Zugriff nur auf AJP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf AJP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes, ohne AJP-Header.

Zugriff nur auf AJP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`statusCode:` ***Zahl***

Der vom Servlet-Container zurückgegebene HTTP-Statuscode für Antworten auf AJP Forward Request-Nachrichten.

Zugriff nur auf `AJP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`uri:` ***Schnur***

Der URI für die Anfrage vom Server an den Servlet-Container. Nicht für Nicht-AJP-Nachrichtentypen festgelegt.

`CDP`

Das Cisco Discovery Protocol (CDP) ist ein proprietäres Protokoll, das es verbundenen Cisco-Geräten ermöglicht, Informationen aneinander zu senden. Die `CDP` Klasse ermöglicht den Zugriff auf Eigenschaften von `CDP_FRAME` Ereignisse.

Ereignisse

`CDP_FRAME`

Läuft auf jedem CDP-Frame, der vom Gerät verarbeitet wird.

Eigenschaften

`destination:` ***Schnur***

Die Ziel-MAC-Adresse. Das häufigste Ziel ist `01:00:0c:cc:cc:cc`, gibt eine Multicast-Adresse an.

`checksum:` ***Zahl***

Die CDP-Prüfsumme.

`source:` ***Gerät***

Das Gerät, das den CDP-Frame sendet.

`ttl:` ***Zahl***

Die Lebenszeit, ausgedrückt in Sekunden. Dies ist der Zeitraum , für den die Informationen in diesem Frame gültig sind, beginnend mit dem Zeitpunkt, an dem die Informationen empfangen wurden.

`tlvs:` ***Reihe von Objekten***

Ein Array, das jedes Feld vom Typ, Länge und Wert (TLV) enthält. Ein TLV-Feld enthält Informationen wie Geräte-ID, Adresse und Plattform. Jedes Feld ist ein Objekt mit den folgenden Eigenschaften:

`type:` ***Zahl***

Der Typ von TLV.

`value:` ***Puffer***

Der Wert des TLV.

`version:` ***Zahl***

Die CDP-Protokollversion.

`CIFS`

Die `CIFS` Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `CIFS_REQUEST` und `CIFS_RESPONSE` Ereignisse.

Ereignisse

`CIFS_REQUEST`

Läuft auf jedem CIFS Anfrage, die vom Gerät verarbeitet wurde.

`CIFS_RESPONSE`

Läuft auf jeder CIFS-Antwort, die vom Gerät verarbeitet wird.

Hinweis: Die `CIFS_RESPONSE` Die Ereignis läuft nach jedem `CIFS_REQUEST` Ereignis, auch wenn die entsprechende Reaktion vom ExtraHop-System nie beobachtet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `CIFS_RESPONSE` Ereignis. Commits aufzeichnen für `CIFS_REQUEST` Ereignisse werden nicht unterstützt.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

Wichtig: Die Zugriffszeit ist die Zeit, die ein CIFS-Server benötigt, um einen angeforderten Block zu empfangen. Es gibt keine Zugriffszeit für Operationen, die nicht auf tatsächliche Blockdaten innerhalb einer Datei zugreifen. Die Verarbeitungszeit ist die Zeit, die ein CIFS-Server benötigt, um auf den vom Client angeforderten Vorgang zu antworten, z. B. eine Anforderung zum Abrufen von Metadaten.

Es gibt keine Zugriffszeiten für SMB2_CREATE-Befehle, die eine Datei erstellen, auf die in der Antwort von einem SMB2_FILEID-Befehl verwiesen wird. Die referenzierten Dateiblöcke werden dann vom NAS-Speichergerät gelesen oder darauf geschrieben. Diese Datei-Lese- und Schreiboperationen werden als Zugriffszeiten berechnet.

`accessTime:` ***Zahl***

Die Zeit, die der Server für den Zugriff auf eine Datei auf der Festplatte benötigt, ausgedrückt in Millisekunden. Für CIFS ist dies die Zeit vom ersten READ-Befehl in einem CIFS-Flow bis zum ersten Byte der Antwortnutzlast. Der Wert ist `NaN` wenn die Messung oder der Zeitpunkt ungültig sind.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`dialect:` ***Schnur***

Der SMB-Dialekt, der zwischen dem Client und dem Server ausgehandelt wurde.

`encryptedBytes:` ***Zahl***

Die Anzahl der verschlüsselten Byte in der Anfrage oder Antwort.

`encryptionProtocol:` ***Schnur***

Das Protokoll, mit dem die Transaktion verschlüsselt ist.

`error:` ***Schnur***

Die detaillierte Fehlermeldung, die vom ExtraHop-System aufgezeichnet wurde.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isCommandCreate:` ***Boolescher Wert***

Der Wert ist `true` wenn die Nachricht einen Befehl zur SMB-Dateierstellung enthält.

`isCommandDelete:` ***Boolescher Wert***

Der Wert ist `true` wenn die Nachricht einen SMB DELETE-Befehl enthält.

`isCommandFileInfo:` ***Boolescher Wert***

Der Wert ist `true` wenn die Nachricht einen SMB-Dateiinformationsbefehl enthält.

`isCommandLock:` ***Boolescher Wert***

Der Wert ist `true` wenn die Nachricht einen SMB-Sperrbefehl enthält.

`isCommandRead:` ***Boolescher Wert***

Der Wert ist `true` wenn die Nachricht einen SMB READ-Befehl enthält.

`isCommandRename:` ***Boolescher Wert***

Der Wert ist `true` wenn die Nachricht einen SMB RENAME-Befehl enthält.

`isCommandWrite:` ***Boolescher Wert***

Der Wert ist `true` wenn die Nachricht einen SMB WRITE-Befehl enthält.

`isDecrypted:` ***Boolescher Wert***

Der Wert ist wahr, wenn das ExtraHop-System die Transaktion sicher entschlüsselt und analysiert hat. Durch die Analyse des entschlüsselten Datenverkehrs können komplexe Bedrohungen aufgedeckt werden, die sich im verschlüsselten Verkehr verstecken.

`isEncrypted:` ***Boolescher Wert***

Der Wert ist wahr, wenn die Transaktion verschlüsselt ist.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist wahr, wenn die Verbindung geschlossen wird, bevor die CIFS-Antwort abgeschlossen war.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isRspSigned:` ***Boolescher Wert***

Der Wert ist wahr, wenn die Antwort vom CIFS-Server signiert ist.

`method:` ***Schnur***

Die CIFS-Methode. Entspricht den Methoden, die unter der CIFS-Metrik im ExtraHop-System aufgeführt sind.

`msgID:` ***Zahl***

Die SMB-Transaktions-ID.

`payload:` ***Puffer***

Die Puffer Objekt, das die Payload-Bytes ab dem READ- oder WRITE-Befehl in der CIFS-Nachricht enthält.

Der Puffer enthält den *N* erste Byte der Nutzlast, wobei *N* ist die Anzahl der Payload-Bytes, spezifiziert durch L7-Nutzdaten-Bytes zum Puffer Option, wenn der Auslöser über die ExtraHop WebUI konfiguriert wurde. Die Standardanzahl von Bytes ist 2048. Weitere Informationen finden Sie unter Erweiterte Trigger-Optionen.

Hinweis: Der Puffer darf nicht mehr als 4 KB enthalten, auch wenn L7-Nutzdaten-Bytes zum Puffer Option ist auf einen höheren Wert gesetzt.

Bei größeren Mengen von Nutzdatenbytes kann die Nutzlast auf eine Reihe von READ- oder WRITE-Befehlen verteilt werden, sodass kein einzelnes Trigger-Ereignis die gesamte angeforderte Nutzlast enthält. Sie können die Nutzlast wieder zu einem einzigen, konsolidierten Puffer zusammensetzen, indem Sie `Flow.store` und `payloadOffset` Eigenschaften.

`payloadOffset:` ***Zahl***

Der Datei-Offset, ausgedrückt in Byte, innerhalb der `resource` Eigentum. Die Payload-Eigenschaft wird von der `resource` Eigentum am Offset.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `CIFS.commitRecord` auf einem `CIFS_RESPONSE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- accessTime
- clientIsExternal
- clientZeroWnd
- error
- isCommandCreate
- isCommandDelete
- isCommandFileInfo
- isCommandLock
- isCommandRead
- isCommandRename
- isCommandWrite
- method
- processingTime
- receiverIsExternal
- reqSize
- reqXfer
- resource
- rspBytes
- rspXfer
- senderIsExternal
- serverIsExternal
- serverZeroWnd
- share
- statusCode
- user
- warning

Zugriff nur auf CIFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqBytes: ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf CIFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqL2Bytes: ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

Zugriff nur auf CIFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqPkts: ***Zahl***

Die Anzahl der Anforderungspakete.

Zugriff nur auf CIFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqRTO: ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf CIFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqSize: ***Zahl***

Die Anzahl der L7-Anforderungsbytes ohne CIFS-Header.

reqTransferTime: ***Zahl***

Die Übertragungszeit der Anfrage, ausgedrückt in Millisekunden. Wenn die Anfrage in einem einzigen Paket enthalten ist, ist die Übertragungszeit Null. Wenn sich die Anfrage über mehrere Pakete erstreckt, ist der Wert die Zeitspanne zwischen der Erkennung des ersten CIFS-Anforderungspakets und der Erkennung des letzten Paket durch das ExtraHop-System. Ein hoher

Wert kann auf eine große CIFS-Anfrage oder eine Netzwerkverzögerung hinweisen. Der Wert ist `NaN` wenn es keine gültige Messung gibt oder wenn der Zeitpunkt ungültig ist.

Zugriff nur auf `CIFS_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqVersion:` ***Schnur***

Die Version von SMB, die auf der Anfrage ausgeführt wird.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`resource:` ***Schnur***

Die gemeinsame Nutzung, der Pfad und der Dateiname, miteinander verknüpft.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes ohne CIFS-Header.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTransferTime:` ***Zahl***

Die Antwortübertragungszeit, ausgedrückt in Millisekunden. Wenn die Antwort in einem einzigen Paket enthalten ist, ist die Übertragungszeit Null. Wenn sich die Antwort über mehrere Pakete erstreckt, ist der Wert die Zeitspanne zwischen der Erkennung des ersten CIFS-Antwortpakets und der Erkennung des letzten Paket durch das ExtraHop-System. Ein hoher Wert kann auf eine große CIFS-Antwort oder eine Netzwerkverzögerung hinweisen. Der Wert ist `NaN` wenn es keine gültige Messung gibt oder wenn der Zeitpunkt ungültig ist.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspVersion:` ***Schnur***

Die Version von SMB, die auf der Antwort ausgeführt wird.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`share:` ***Schnur***

Der Name der Aktie, mit der der Benutzer verbunden ist.

`statusCode:` ***Zahl***

Der numerische Statuscode der Antwort (nur SMB1 und SMB2).

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`user:` ***Schnur***

Der Nutzername, falls verfügbar. In einigen Fällen, z. B. wenn das Anmeldeereignis nicht sichtbar war oder der Zugriff anonym war, ist der Benutzername nicht verfügbar.

`warning:` ***Schnur***

Die detaillierte Warnmeldung, die vom ExtraHop-System aufgezeichnet wurde.

Zugriff nur auf `CIFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

Beispiele für Trigger

- Beispiel: Überwachen Sie CIFS-Aktionen auf Geräten

`DB`

Die `DB`, oder Datenbank, Klasse ermöglicht es Ihnen , Metriken zu speichern und auf Eigenschaften zuzugreifen `DB_REQUEST` und `DB_RESPONSE` Ereignisse.

Ereignisse

`DB_REQUEST`

Wird bei jeder Datenbankanfrage ausgeführt, die vom Gerät verarbeitet wird.

`DB_RESPONSE`

Läuft auf jeder Datenbankantwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `DB_RESPONSE` Ereignis. Commits aufzeichnen für `DB_REQUEST` Ereignisse werden nicht unterstützt.

Die Standardeigenschaften, die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`appName:` ***Schnur***

Die Client Anwendungsname, der nur für MS SQL-Verbindungen extrahiert wird.

`correlationId:` ***Zahl***

Die Korrelations-ID für DB2-Anwendungen. Der Wert ist `null` für Nicht-DB2-Anwendungen.

`database:` ***Schnur***

Die Datenbankinstanz. In einigen Fällen, z. B. wenn Anmeldeereignisse verschlüsselt sind, ist der Datenbankname nicht verfügbar.

`encryptionProtocol:` ***Schnur***

Das Protokoll, mit dem die Transaktion verschlüsselt ist.

`error:` ***Schnur***

Die detaillierten Fehlermeldungen, die vom ExtraHop-System im Zeichenkettenformat aufgezeichnet wurden. Wenn eine Antwort mehrere Fehler enthält, werden die Fehler zu einer Zeichenfolge verkettet.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`errors:` ***Reihe von Zeichenketten***

Die detaillierten Fehlermeldungen, die vom ExtraHop-System im Array-Format aufgezeichnet wurden. Wenn die Antwort nur einen einzigen Fehler enthält, wird der Fehler als Array zurückgegeben, das eine Zeichenfolge enthält.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isDecrypted:` ***Boolescher Wert***

Der Wert ist wahr, wenn das ExtraHop-System die Transaktion sicher entschlüsselt und analysiert hat. Durch die Analyse des entschlüsselten Datenverkehrs können komplexe Bedrohungen aufgedeckt werden, die sich im verschlüsselten Verkehr verstecken.

`isEncrypted:` ***Boolescher Wert***

Der Wert ist wahr, wenn die Transaktion verschlüsselt ist.

`isReqAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die DB-Anfrage abgeschlossen ist.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die DB-Antwort abgeschlossen ist.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`method:` ***Schnur***

Die Datenbankmethode, die mit den Methoden korreliert, die unter der Datenbankmetrik im ExtraHop-System aufgeführt sind.

`params:` ***Reihe***

Ein Array von Remote-Prozeduraufrufen (RPC) Parameter, die nur für Microsoft SQL-, PostgreSQL- und DB2-Datenbanken verfügbar sind.

Das Array enthält jeden der folgenden Parameter:

`name:` ***Schnur***

Der optionale Name des angegebenen RPC-Parameters.

`value:` ***Schnur | Zahl***

Ein Text-, Ganzzahl- oder Zeit- und Datumsfeld. Wenn der Wert kein Text-, Ganzzahl- oder Zeit- und Datumsfeld ist, wird der Wert in die HEX/ASCII-Form umgewandelt.

Der Wert des `params` Die Eigenschaft ist dieselbe, wenn auf eine der folgenden Seiten zugegriffen wird `DB_REQUEST` oder der `DB_RESPONSE` Ereignis.

`procedure:` ***Schnur***

Der Name der gespeicherten Prozedur. Entspricht den Verfahren, die unter Datenbankmethoden im ExtraHop-System aufgeführt sind.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden (entspricht `rspTimeToFirstByte - reqTimeToLastByte`). Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `DB.commitRecord` auf einem `DB_RESPONSE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- App-Name
- Kunde ist extern
- Kunde ZeroWND
- Korrelations-ID
- Datenbank
- Fehler
- wird neu abgebrochen
- Ist RS abgebrochen
- Methode
- Verfahren
- Empfänger ist extern
- REQ-Größe
- ReqTime bis LastByte
- RSP-Größe
- RSP-Zeit bis zum ersten Byte
- RSP-TimezuLastByte
- Bearbeitungszeit
- Absender ist extern
- Server ist extern
- Server Zero WND
- Aussage
- Tabelle
- Benutzer

Zugriff nur auf DB_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf DB_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Headern.

Zugriff nur auf DB_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

Zugriff nur auf DB_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf DB_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqSize:` ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne Datenbankprotokoll-Header.

`reqTimeToLastByte:` ***Zahl***

Die Zeit vom ersten Byte der Anforderung bis zum letzten Byte der Anforderung, ausgedrückt in Millisekunden. Retouren `NaN` bei falsch formatierten und abgebrochenen Anfragen oder wenn das Timing ungültig ist.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes, ohne Datenbankprotokoll-Header.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToFirstByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum ersten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei fehlerhaften und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToLastByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum letzten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei fehlerhaften und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `DB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`serverVersion:` ***Schnur***

Die MS SQL Server-Version.

`statement:` ***Schnur***

Die vollständige SQL-Anweisung, die möglicherweise nicht für alle Datenbankmethoden verfügbar ist.

`table:` ***Schnur***

Der Name der Datenbanktabelle, die in der aktuellen Anweisung angegeben wurde. Die folgenden Datenbanken werden unterstützt:

- `Sybase`
- `Sybase IQ`
- `MySQL`
- `PostgreSQL`
- `IBM Informix`

- `MS SQL TDS`
- `Oracle TNS`
- `DB2`

Gibt ein leeres Feld zurück, wenn die Anfrage keinen Tabellennamen enthält.

`user:` ***Schnur***

Der Nutzername, falls verfügbar. In einigen Fällen, z. B. wenn Anmeldeereignisse verschlüsselt sind, ist der Benutzername nicht verfügbar.

Beispiele für Trigger

- Beispiel: Antwortmetriken für Datenbankabfragen sammeln
- Beispiel: Erstellen Sie einen Anwendungscontainer

`DHCP`

Die DHCP Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `DHCP_REQUEST` und `DHCP_RESPONSE` Ereignisse.

Ereignisse

`DHCP_REQUEST`

Wird bei jeder vom Gerät verarbeiteten DHCP-Anfrage ausgeführt.

`DHCP_RESPONSE`

Läuft auf jeder vom Gerät verarbeiteten DHCP-Antwort.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `DHCP_REQUEST` oder `DHCP_RESPONSE` Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die Standardeigenschaften, die für jedes Ereignis übernommen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

`getOption(optionCode:` ***Zahl*** `):` ***Objekt***

Akzeptiert eine Ganzzahl für den DHCP-Optionscode als Eingabe und gibt ein Objekt zurück, das die folgenden Felder enthält:

`code:` ***Zahl***

Der DHCP-Optionscode.

`name:` ***Schnur***

Der Name der DHCP-Option.

`payload:` ***Zahl*** | ***Schnur***

Der Typ der zurückgegebenen Nutzlast ist unabhängig vom Typ für diese spezifische Option, z. B. eine IP-Adresse, ein Array von IP-Adressen oder ein Pufferobjekt.

Retouren `null` wenn der angegebene Optionscode in der Nachricht nicht vorhanden ist.

Eigenschaften

`chaddr:` ***Schnur***

Die Client-Hardwareadresse des DHCP-Clients.

`clientReqDelay:` ***Zahl***

Die Zeit verging vor dem Client versucht, eine DHCP-Lease zu erwerben oder zu erneuern, ausgedrückt in Sekunden.

Zugriff nur auf `DHCP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`error:` ***Schnur***

Die mit dem Optionscode 56 verbundene Fehlermeldung. Der Wert ist `null` wenn es keinen Fehler gibt.

Zugriff nur auf `DHCP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`gwAddr:` ***IP-Adresse***

Die IP-Adresse, über die Router Anforderungs- und Antwortnachrichten weiterleiten.

`htype:` ***Zahl***

Der Hardwaretypcode.

`msgType:` ***Schnur***

Der DHCP-Nachrichtentyp. Folgende Nachrichtentypen werden unterstützt:

- `DHCPDISCOVER`
- `DHCPOFFER`
- `DHCPREQUEST`
- `DHCPDECLINE`
- `DHCPACK`
- `DHCPNAK`
- `DHCPRELEASE`
- `DHCPINFORM`
- `DHCPFORCERENEW`
- `DHCPLEASEQUERY`
- `DHCPLEASEUNASSIGNED`
- `DHCPLEASEUNKNOWN`
- `DHCPLEASEACTIVE`
- `DHCPBULKLEASEQUERY`
- `DHCPLEASEQUERYDONE`

`offeredAddr:` ***IP-Adresse***

Die IP-Adresse, die der DHCP-Server anbietet oder dem zuweist Client.

Zugriff nur auf `DHCP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`options:` ***Reihe von Objekten***

Eine Reihe von Objekten, wobei jedes Objekt die folgenden Felder enthält:

`code:` ***Zahl***

Der DHCP-Optionscode.

`name:` ***Schnur***

Der Name der DHCP-Option.

`payload:` ***Zahl | Schnur***

Der Typ der zurückgegebenen Nutzlast ist unabhängig vom Typ für diese spezifische Option, z. B. eine IP-Adresse, ein Array von IP-Adressen oder ein Pufferobjekt. IP-Adressen werden in ein Array geparst, aber wenn die Anzahl der Byte nicht durch 4 teilbar ist, wird sie stattdessen als Puffer zurückgegeben.

`paramReqList:` ***Schnur***

Eine durch Kommas getrennte Liste von Zahlen, die die vom Client vom Server angeforderten DHCP-Optionen darstellt. Eine vollständige Liste der DHCP-Optionen finden Sie unter https://www.iana.org/assignments/bootp-dhcp-parameters/bootp-dhcp-parameters.xhtml.

`processingTime:` ***Zahl***

Die Prozesszeit, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist .

Zugriff nur auf `DHCP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `DHCP.commitRecord` entweder auf einem `DHCP_REQUEST` oder `DHCP_RESPONSE` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

DHCP_REQUEST	DHCP_RESPONSE
clientIsExternal	clientIsExternal
clientReqDelay	error
gwAddr	gwAddr
htype	htype
msgType	msgType
receiverIsExternal	offeredAddr
reqBytes	processingTime
reqL2Bytes	rspBytes
reqPkts	rspL2Bytes
senderIsExternal	rspPkts
serverIsExternal	receiverIsExternal
txId	senderIsExternal
	serverIsExternal
	txId

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf `DHCP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Headern.

Zugriff nur auf `DHCP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

Zugriff nur auf `DHCP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen L4-Protokoll-Overhead, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf `DHCP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf `DHCP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

Zugriff nur auf `DHCP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`txId:` ***Zahl***

Die Transaktions-ID.

`vendor:` ***Schnur***

Der Vendor Class Identifier (VCI), der den Anbieter angibt, der auf dem Client oder Server ausgeführt wird.

`DICOM`

Die DICOM In der Klasse (DICOM) können Sie Metriken speichern und auf Eigenschaften zugreifen `DICOM_REQUEST` und `DICOM_RESPONSE` Ereignisse.

Ereignisse

`DICOM_REQUEST`

Läuft bei jeder DICOM-Anfrage, die vom Gerät verarbeitet wird.

`DICOM_RESPONSE`

Läuft auf jeder vom Gerät verarbeiteten DICOM-Antwort.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `DICOM_REQUEST` oder `DICOM_RESPONSE` Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die Standardeigenschaften, die für jedes Ereignis festgeschrieben wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

`findElement(groupTag:` ***Zahl*** `, elementTag:` ***Zahl*** `):` ***Puffer***

Gibt einen Puffer zurück, der das DICOM-Datenelement enthält, das durch die übergebenen Gruppen - und Element-Tag-Nummern angegeben ist.

Das Datenelement wird durch ein eindeutiges geordnetes Ganzzahlpaar dargestellt, das die Gruppen-Tag- und Element-Tag-Nummern darstellt. Beispielsweise steht das geordnete Paar „0008, 0008“ für das Element „Bildtyp“. EIN Registrierung der DICOM-Datenelemente und definierte Tags sind verfügbar unter dicom.nema.org.

`groupTag:` ***Zahl***

Die erste Zahl in dem eindeutigen geordneten Ganzzahlpaar, das ein bestimmtes Datenelement darstellt.

`elementTag:` ***Zahl***

Die zweite Zahl in dem eindeutigen geordneten Paar oder den ganzen Zahlen, die ein bestimmtes Datenelement darstellen.

Eigenschaften

`calledAETitle:` ***Schnur***

Der Titel der Anwendungseinheit (AE) des Zielgeräts oder -programms.

`callingAETitle:` ***Schnur***

Der Titel der Anwendungseinheit (AE) des Quellgeräts oder -programms.

`elements:` ***Reihe***

Eine Reihe von PDV-Befehlselementen (Presentation Data Values) und Datenelementen, die eine DICOM-Nachricht umfassen.

`error:` ***Schnur***

Die detaillierte Fehlermeldung, die vom ExtraHop-System aufgezeichnet wurde.

`isReqAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die DICOM-Anforderung abgeschlossen ist.

Zugriff nur auf `DICOM_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die DICOM-Antwort abgeschlossen ist.

Zugriff nur auf `DICOM_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isSubOperation:` ***Boolescher Wert***

Der Wert ist `true` wenn die Timing-Metrik auf einem L7 Die Protokollnachricht ist nicht verfügbar, da die primäre Anfrage oder Antwort nicht vollständig ist.

`methods:` ***Reihe von Zeichenketten***

Eine Reihe von Befehlsfeldern in der Nachricht. Jedes Befehlsfeld gibt einen DIMSE-Operationsnamen an, z. B. N-CREATE-RSP.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `DICOM_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `DICOM.commitRecord` entweder auf einem `DICOM_REQUEST` oder `DICOM_RESPONSE` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

DICOM_REQUEST	DICOM_RESPONSE
calledAETitle	calledAETitle
callingAETitle	callingAETitle
clientIsExternal	clientIsExternal
clientZeroWnd	clientZeroWnd
error	error
isReqAborted	isRspAborted
isSubOperation	isSubOperation
method	method

DICOM_REQUEST	DICOM_RESPONSE
receiverIsExternal	processingTime
reqPDU	receiverIsExternal
reqSize	rspPDU
reqTransferTime	rspSize
senderIsExternal	rspTransferTime
serverIsExternal	senderIsExternal
serverZeroWnd	serverIsExternal
version	serverZeroWnd
	version

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf `DICOM_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

`reqPDU:` ***Schnur***

Die Protocol Data Unit (PDU) oder das Nachrichtenformat der Anfrage.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

`reqSize:` ***Zahl***

Die Anzahl der L7-Anforderungsbytes.

Zugriff nur auf `DICOM_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqTransferTime:` ***Zahl***

Die Übertragungszeit der Anfrage, ausgedrückt in Millisekunden.

Zugriff nur auf `DICOM_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

Zugriff nur auf `DICOM_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf `DICOM_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

`rspPDU:` ***Schnur***

Die Protocol Data Unit (PDU) oder das Nachrichtenformat der Antwort.

Zugriff nur auf DICOM_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes.

Zugriff nur auf DICOM_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspTransferTime:` ***Zahl***

Die Antwortübertragungszeit, ausgedrückt in Millisekunden.

Zugriff nur auf DICOM_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`version:` ***Zahl***

Die DICOM-Versionsnummer.

DNS

Die DNS Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen DNS_REQUEST und DNS_RESPONSE Ereignisse.

Ereignisse

`DNS_REQUEST`

Wird bei jeder DNS-Anfrage ausgeführt, die vom Gerät verarbeitet wird.

`DNS_RESPONSE`

Läuft auf jeder DNS-Antwort, die vom Gerät verarbeitet wird.

Methoden

`answersInclude(term:` ***Schnur*** | ***IP-Addresse*** `):` ***Boolescher Wert***

Retouren `true` wenn der angegebene Begriff in einer DNS-Antwort vorhanden ist. Bei Zeichenkettenbegriffen überprüft die Methode sowohl den Namen als auch den Datensatz im Antwortabschnitt der Antwort. Für IPAddress Begriffe, die Methode überprüft nur den Datensatz im Antwortabschnitt.

Kann nur angerufen werden am DNS_RESPONSE Ereignisse.

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem DNS_REQUEST oder DNS_RESPONSE Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die Standardeigenschaften, die für jedes Ereignis übernommen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`answers:` ***Reihe***

Eine Reihe von Objekten, die Antwortressourceneinträgen entsprechen.

Zugriff nur auf DNS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

Die Objekte enthalten die folgenden Eigenschaften:

data: ***Schnur*** | ***IP-Addresse***

Der Wert der Daten hängt vom Typ ab. Der Wert ist null für nicht unterstützte Datensatztypen. Zu den unterstützten Datensatztypen gehören:

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

name: ***Schnur***

Der Name des Datensatz.

ttl: ***Zahl***

Der Time-to-Live-Wert.

type: ***Schnur***

Der DNS-Eintragstyp.

typeNum: ***Zahl***

Die numerische Darstellung des DNS-Eintragstyps.

error: ***Schnur***

Der Name des DNS-Fehlercodes gemäß den IANA-DNS-Parametern.

Gibt OTHER für Fehlercodes zurück, die vom System nicht erkannt werden; jedoch errorNum gibt den numerischen Codewert an.

Zugriff nur auf DNS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

errorNum: ***Zahl***

Die numerische Darstellung des DNS-Fehlercodes gemäß den IANA-DNS-Parametern.

Zugriff nur auf DNS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

isAuthenticData: ***Boolescher Wert***

Der Wert ist true wenn die Antwort über DNSSEC validiert wurde.

Zugriff nur auf DNS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

isAuthoritative: ***Boolescher Wert***

Der Wert ist true wenn die verbindliche Antwort in der Antwort angegeben ist.

Zugriff nur auf DNS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

isCheckingDisabled: ***Boolescher Wert***

Der Wert ist true wenn eine Antwort zurückgegeben werden soll, obwohl die Anfrage nicht authentifiziert werden konnte.

Zugriff nur auf DNS_REQUEST Ereignisse; andernfalls tritt ein Fehler auf.

isDGADomain: ***Boolescher Wert***

Der Wert ist true ob die Domäne des Server möglicherweise durch einen Domänengenerierungsalgorithmus (DGA) generiert wurde. Einige Arten von Malware erzeugen eine große Anzahl von Domainnamen mit DGAs, um Command-and-Control-Server zu verstecken. Der Wert ist null wenn die Domain nicht verdächtig war.

`isRecursionAvailable:` ***Boolescher Wert***

Der Wert ist `true` wenn der Nameserver rekursive Abfragen unterstützt.

Zugriff nur auf `DNS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isRecursionDesired:` ***Boolescher Wert***

Der Wert ist `true` ob der Nameserver die Abfrage rekursiv durchführen soll.

Zugriff nur auf `DNS_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`isReqTimeout:` ***Boolescher Wert***

Der Wert ist `true` wenn das Zeitlimit für die Anfrage überschritten wurde.

Zugriff nur auf `DNS_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`isRspTruncated:` ***Boolescher Wert***

Der Wert ist `true` wenn die Antwort gekürzt ist.

Zugriff nur auf `DNS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`opcode:` ***Schnur***

Der Name des DNS-Operationscodes gemäß den IANA-DNS-Parametern. Die folgenden Codes werden vom ExtraHop-System erkannt:

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

Gibt OTHER für Codes zurück, die vom System nicht erkannt werden; der `opcodeNum` Eigenschaft gibt den numerischen Codewert an.

`opcodeNum:` ***Zahl***

Die numerische Darstellung des DNS-Operationscodes gemäß den IANA-DNS-Parametern.

`payload:` ***Puffer***

Die Puffer Objekt, das die rohen Payload-Bytes der Ereignistransaktion enthält.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Byte. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `DNS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`qname:` ***Schnur*** | ***null***

Der abgefragte Hostname.

Dieser Wert ist `null` wenn der `opcode` Eigentum ist `UPDATE`.

`qtype:` ***Schnur*** | ***null***

Der Name des DNS-Anforderungsdatentyps gemäß den IANA-DNS-Parametern.

Retouren `OTHER` für Typen, die vom System nicht erkannt werden; jedoch `qtypeNum` property gibt den numerischen Typwert an.

Dieser Wert ist `null` wenn der `opcode` Eigentum ist `UPDATE`.

`qtypeNum:` ***Zahl*** | ***null***

Die numerische Darstellung des DNS-Anforderungsdatentyps gemäß den IANA-DNS-Parametern.

Dieser Wert ist `null` wenn der `opcode` Eigentum ist `UPDATE`.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `DNS.commitRecord()` entweder auf einem `DNS_REQUEST` oder `DNS_RESPONSE` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

DNS_REQUEST	DNS_RESPONSE
clientIsExternal	answers
clientZeroWnd	clientIsExternal
isCheckingDisabled	clientZeroWnd
isDGADomain	error
isRecursionDesired	isAuthoritative
isReqTimeout	isCheckingDisabled
opcode	isDGADomain
qname	isRecursionAvailable
qtype	isRspTruncated
receiverIsExternal	opcode
reqBytes	processingTime
reqL2Bytes	receiverIsExternal
reqPkts	qname
senderIsExternal	qtype
serverIsExternal	rspBytes
serverZeroWnd	rspL2Bytes
	rspPkts
	senderIsExternal
	serverIsExternal
	serverZeroWnd

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf `DNS_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Headern.

Zugriff nur auf `DNS_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

Zugriff nur auf DNS_REQUEST Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf DNS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf DNS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortbytes auf Anwendungsebene.

Zugriff nur auf DNS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`txId:` ***Zahl***

Die Transaktions-ID der DNS-Anfrage oder -Antwort.

`zname:` ***Schnur*** | ***null***

Die DNS-Zone wird aktualisiert.

Dieser Wert ist `null` wenn der `opcode` Eigentum ist nicht `UPDATE`.

`ztype:` ***Schnur*** | ***null***

Der Typ der DNS-Zone, die aktualisiert wird. Retouren `OTHER` für Typen, die vom System nicht erkannt werden.

Dieser Wert ist `null` wenn der `opcode` Eigentum ist nicht `UPDATE`.

`ztypeNum:` ***Zahl*** | ***null***

Die numerische Darstellung des DNS-Zonentyps.

Dieser Wert ist `null` wenn der `opcode` Eigentum ist nicht `UPDATE`.

FIX

Die FIX Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen FIX_REQUEST und FIX_RESPONSE Ereignisse.

Ereignisse

`FIX_REQUEST`

Läuft bei jeder FIX-Anfrage, die vom Gerät verarbeitet wird.

`FIX_RESPONSE`

Läuft auf jeder vom Gerät verarbeiteten FIX-Antwort.

Hinweis: Die FIX_RESPONSE Das Ereignis wird einer Anfrage zugeordnet, die auf der Bestellnummer basiert. Es gibt keine Eins-zu-Eins-Korrelation zwischen Anfrage und Antwort. Möglicherweise gibt es Anfragen ohne Antwort, und manchmal werden Daten an die Client, wodurch die Verfügbarkeit von Anforderungsdaten bei einem Antwortereignis eingeschränkt wird. Sie können die Sitzungstabelle jedoch aufrufen, um komplexe Szenarien wie die ID der Übermittlungsreihenfolge zu lösen.

Methoden

`commitRecord()`: ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `FIX_REQUEST` oder `FIX_RESPONSE` Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die für jedes Ereignis festgeschriebenen Standardeigenschaften finden Sie unter `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`fields`: ***Reihe***

Eine Liste von FIX-Feldern. Da sie textbasiert sind, werden die Schlüssel-Wert-Protokollfelder als ein Array von Objekten mit Namens- und Werteigenschaften bereitgestellt, die Zeichenketten enthalten. Zum Beispiel:

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

übersetzt zu:

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

Die Darstellung von Schlüsselzeichenfolgen wird, wenn möglich, übersetzt. Bei Erweiterungen wird eine numerische Darstellung verwendet. Es ist beispielsweise nicht möglich, 9178=0 zu bestimmen (wie bei tatsächlichen Aufnahmen). Der Schlüssel wird stattdessen in „9178" übersetzt. Felder werden extrahiert, nachdem Nachrichtenlänge und Versionsfelder bis zur Prüfsumme (letztes Feld) extrahiert wurden. Die Prüfsumme wird nicht extrahiert.

Im folgenden Beispiel ist der Auslöser `debug(JSON.stringify(FIX.fields));` zeigt die folgenden Felder:

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

Um alle FIX-Felder zu debuggen und zu drucken, aktivieren Sie das Debuggen auf dem Auslöser und geben Sie den folgenden Code ein:

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

Die folgende Ausgabe wird im Debug-Log des Triggers angezeigt:

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

`msgType:` ***Schnur***

Der Wert des MessageCompID-Schlüssels.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn das Timing ungültig ist.

Zugriff nur auf `ANTWORT KORRIGIEREN` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `FIX.commitRecord` entweder auf einem `FIX_REQUEST` oder `FIX_RESPONSE` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

FIX_REQUEST	FIX_RESPONSE
clientIsExternal	clientIsExternal
clientZeroWnd	clientZeroWnd
msgType	msgType
receiverIsExternal	receiverIsExternal
reqBytes	rspBytes
reqL2Bytes	rspL2Bytes
reqPkts	rspPkts
reqRTO	rspRTO
sender	sender
senderIsExternal	senderIsExternal
serverIsExternal	serverIsExternal
serverZeroWnd	serverZeroWnd
target	target
version	version

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`sender:` ***Schnur***

Der Wert des SenderCompID-Schlüssels.

`target:` ***Schnur***

Der Wert des TargetCompID-Schlüssels.

`version:` ***Schnur***

Die Protokollversion.

FTP

Die FTP Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `FTP_REQUEST` und `FTP_RESPONSE` Ereignisse.

Ereignisse

`FTP_REQUEST`

Läuft bei jeder FTP-Anfrage, die vom Gerät verarbeitet wird.

`FTP_RESPONSE`

Läuft auf jeder FTP-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `FTP_RESPONSE` Ereignis. Commits aufzeichnen für `FTP_REQUEST` Ereignisse werden nicht unterstützt.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`args:` ***Schnur***

Die Argumente für den Befehl.

Zugriff nur auf `FTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`cwd:` ***Schnur***

Im Fall eines Benutzers bei `/`, wenn der Client sendet „CWD subdir“:

- Der Wert ist `/` wenn Methode == „CWD“.
- Der Wert ist `/subdir` für nachfolgende Befehle (anstatt dass CWD als Teil des CWD-Antwortauslösers zum Verzeichnis „Changeto“ wird).

Schließt „...“ am Anfang des Pfads ein, Ereignis eine Neusynchronisierung erfolgt oder der Pfad gekürzt wird.

Schließt „...“ am Ende des Pfads ein, wenn der Pfad zu lang ist. Der Pfad wird bei 4096 Zeichen gekürzt.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`error:` ***Schnur***

Die detaillierte Fehlermeldung, die vom ExtraHop-System aufgezeichnet wurde.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`isReqAborted:` ***Boolescher Wert***

Der Wert ist `true` Die Verbindung wurde geschlossen, bevor die FTP-Anfrage abgeschlossen war.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wurde, bevor die FTP-Antwort abgeschlossen war.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`method:` ***Schnur***

Die FTP-Methode.

`path:` ***Schnur***

Der Pfad für FTP-Befehle. Schließt „...“ am Anfang des Pfads ein, Ereignis eine Neusynchronisierung erfolgt oder der Pfad gekürzt wird. Schließt „...“ am Ende des Pfads ein, wenn der Pfad zu lang ist. Der Pfad wird bei 4096 Zeichen gekürzt.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden (entspricht `rspTimeToFirstPayload - reqTimeToLastByte`). Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `FTP.commitRecord()` auf einem FTP_RESPONSE Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `args`
- `clientIsExternal`
- `clientZeroWnd`
- `cwd`
- `error`
- `isReqAborted`
- `isRspAborted`
- `method`
- `path`
- `processingTime`
- `receiverIsExternal`
- `reqBytes`
- `reqL2Bytes`
- `reqPkts`
- `reqRTO`
- `roundTripTime`
- `rspBytes`
- `rspL2Bytes`
- `rspPkts`

- rspRTO
- senderIsExternal
- serverIsExternal
- serverZeroWnd
- statusCode
- transferBytes
- user

Greifen Sie nur auf das Datensatzobjekt zu unter FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`statusCode:` ***Zahl***

Der FTP-Statuscode der Antwort.

Zugriff nur auf FTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

Die folgenden Codes sind gültig:

Kode	Beschreibung
110	Starten Sie die Marker-Wiedergabe neu.
120	Service bereit in *nnn* Minuten.
125	Datenverbindung ist bereits geöffnet; Übertragung wird gestartet.
150	Dateistatus okay; Datenverbindung wird gerade geöffnet.
202	Befehl nicht implementiert, an dieser Standort überflüssig.
211	Systemstatus oder Antwort der Systemhilfe.
212	Verzeichnisstatus.
213	Status der Datei.
214	Hilfemeldung.
215	NAME-Systemtyp.
220	Service bereit für neue Benutzer.
221	Der Dienst schließt die Steuerverbindung.
225	Datenverbindung geöffnet; keine Übertragung im Gange.
226	Datenverbindung wird geschlossen. Die angeforderte Dateiaktion war erfolgreich.
227	In den passiven Modus wechseln.
228	Wechsel in den Long Passive-Modus.
229	Wechsel in den erweiterten passiven Modus.
230	Benutzer hat sich angemeldet, fahren Sie fort. Ggf. ausgeloggt.
231	Der Benutzer hat sich abgemeldet; der Dienst wurde beendet.
232	Abmeldebefehl notiert, wird abgeschlossen, wenn die Übertragung abgeschlossen ist
250	Angeforderte Dateiaktion okay, abgeschlossen.
257	„PATHNAME“ wurde erstellt.
331	Benutzername okay, Passwort erforderlich.
332	Benötige ein Konto für die Anmeldung.
350	Angeforderte Dateiaktion, bis weitere Informationen vorliegen.
421	Dienst nicht verfügbar, Steuerverbindung wird geschlossen.
425	Datenverbindung kann nicht geöffnet werden.
426	Verbindung geschlossen; Übertragung abgebrochen.
430	Ungültiger Nutzername oder Passwort.
434	Der angeforderte Host ist nicht verfügbar.
450	Die angeforderte Dateiaktion wurde nicht ausgeführt.

Kode	Beschreibung
451	Die angeforderte Aktion wurde abgebrochen. Lokaler Fehler bei der Verarbeitung.
452	Die angeforderte Aktion wurde nicht ausgeführt.
501	Syntaxfehler in Parametern oder Argumenten.
502	Befehl nicht implementiert.
503	Schlechte Befehlsfolge.
504	Befehl für diesen Parameter nicht implementiert.
530	Nicht eingeloggt.
532	Benötige ein Konto zum Speichern von Dateien.
550	Die angeforderte Aktion wurde nicht ausgeführt. Datei nicht verfügbar.
551	Die angeforderte Aktion wurde abgebrochen. Seitentyp unbekannt.
552	Die angeforderte Dateiaktion wurde abgebrochen. Speicherzuweisung überschritten.
553	Die angeforderte Aktion wurde nicht ausgeführt. Dateiname nicht zulässig.
631	Integritätsgeschützte Antwort.
632	Vertraulichkeits- und integritätsgeschützte Antwort.
633	Vertraulichkeitsgeschützte Antwort.
10054	Die Verbindung wurde vom Peer zurückgesetzt.
10060	Es kann keine Verbindung zum Remoteserver hergestellt werden.
10061	Es kann keine Verbindung zum Remoteserver hergestellt werden. Die Verbindung ist aktiv und wurde abgelehnt.
10066	Das Verzeichnis ist nicht leer.
10068	Zu viele Benutzer, der Server ist voll.

`transferBytes:` ***Zahl***

Die Anzahl der Byte, die während eines `FTP_RESPONSE` Ereignis.

Zugriff nur auf `FTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`user:` ***Schnur***

Der Benutzername, falls verfügbar. In einigen Fällen, z. B. wenn Anmeldeereignisse verschlüsselt sind, ist der Benutzername nicht verfügbar.

`HL7`

Die HL7 Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `HL7_REQUEST` und `HL7_RESPONSE` Ereignisse.

Ereignisse

`HL7_REQUEST`

Läuft bei jeder HL7-Anfrage, die vom Gerät verarbeitet wird.

`HL7_RESPONSE`

Läuft auf jeder HL7-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `HL7_RESPONSE` Ereignis. Commits aufzeichnen für `HL7_REQUEST` Ereignisse werden nicht unterstützt.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`ackCode:` ***Schnur***

Der zweistellige Bestätigungscode.

Zugriff nur auf `HL7_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`ackId:` ***Schnur***

Der Bezeichner für die Nachricht, die bestätigt wird.

Zugriff nur auf `HL7_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`msgId:` ***Schnur***

Die eindeutige Kennung für diese Nachricht.

`msgType:` ***Schnur***

Das gesamte Feld für den Nachrichtentyp, einschließlich des Unterfeldes MsgID.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `HL7_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `HL7.commitRecord()` auf einem `HL7_RESPONSE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `ackCode`
- `ackId`
- `clientIsExternal`
- `clientZeroWnd`
- `msgId`
- `msgType`
- `receiverIsExternal`
- `roundTripTime`
- `processingTime`
- `senderIsExternal`
- `serverIsExternal`
- `serverZeroWnd`
- `version`

Greifen Sie nur auf das Datensatzobjekt zu unter `HL7_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

Zugriff nur auf `HL7_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`segments:` ***Reihe***

Eine Reihe von Segmentobjekten mit den folgenden Feldern:

`name:` ***Schnur***

Der Name des Segments.

`fields:` ***Reihe von Zeichenketten***

Die Segmentfeldwerte. Da die Indizes des Arrays bei 0 beginnen und die HL7-Feldnummern bei 1 beginnen, ist der Index die HL7-Feldnummer minus 1. Um beispielsweise Feld 16 eines PRT-Segments (die Teilnehmer-Geräte-ID) auszuwählen, geben Sie 15 ein, wie im folgenden Beispielcode gezeigt:

```
HL7.segments[5].fields[15]
```

Hinweis: Wenn ein Segment leer ist, enthält das Array eine leere Zeichenfolge im Segmentindex.

`subfieldDelimiter:` ***Schnur***

Unterstützt nicht standardmäßige Feldbegrenzer.

`version:` ***Schnur***

Die im MSH-Segment beworbene Version.

Hinweis: Die Menge der gepufferten Daten wird durch die folgende Erfassungsoption begrenzt: `("message_length_max": number)`

`HTTP`

Das HTTP Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `HTTP_REQUEST` und `HTTP_RESPONSE` Ereignisse.

Ereignisse

`HTTP_REQUEST`

Wird bei jeder HTTP-Anfrage ausgeführt, die vom Gerät verarbeitet wird.

`HTTP_RESPONSE`

Läuft auf jeder HTTP-Antwort, die vom Gerät verarbeitet wird.

Zusätzliche Payload-Optionen sind verfügbar, wenn Sie einen Auslöser erstellen, der bei einem dieser Ereignisse ausgeführt wird. siehe Erweiterte Trigger-Optionen für weitere Informationen.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `HTTP_REQUEST` oder `HTTP_RESPONSE` Ereignis. Informationen zu den Standardeigenschaften, die dem Record-Objekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Wenn der `commitRecord()` Methode wird auf einem aufgerufen `HTTP_REQUEST` Ereignis, der Datensatz wird erst erstellt, wenn `HTTP_RESPONSE` Ereignis läuft. Wenn der `commitRecord()` Methode wird sowohl für die aufgerufen `HTTP_REQUEST` und das entsprechende `HTTP_RESPONSE`, es wird nur ein Datensatz für Anfrage und Antwort erstellt, auch wenn `commitRecord()` Die Methode wird mehrmals bei denselben Triggerereignissen aufgerufen.

`findHeaders(name:` ***Schnur*** `):` ***Reihe***

Ermöglicht den Zugriff auf HTTP-Header-Werte und gibt ein Array von Header-Objekten (mit Namens - und Werteigenschaften) zurück, deren Namen dem Präfix des Zeichenfolgenwerts entsprechen. siehe Beispiel: Zugriff auf HTTP-Header-Attribute für weitere Informationen.

`parseQuery(String):` ***Objekt***

Akzeptiert eine Abfragezeichenfolge und gibt ein Objekt mit Namen und Werten zurück, die denen in der Abfragezeichenfolge entsprechen, wie im folgenden Beispiel gezeigt:

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

Hinweis: Wenn die Abfragezeichenfolge wiederholte Schlüssel enthält, werden die entsprechenden Werte in einem Array zurückgegeben. Zum Beispiel die Abfragezeichenfolge `event_type=status_update_event&event_type=api_post_event` gibt das folgende Objekt zurück:

```
{
  "event_type": ["status_update_event", "api_post_event"]
}
```

Eigenschaften

`age:` ***Zahl***

Für `HTTP_REQUEST` Ereignisse, die Zeit vom ersten Byte der Anfrage bis zum letzten sichtbaren Byte der Anfrage. Für `HTTP_RESPONSE` Ereignisse, die Zeit vom ersten Byte der Anfrage bis zum letzten sichtbaren Byte der Antwort. Die Zeit wird in Millisekunden ausgedrückt. Gibt einen gültigen Wert für falsch formatierte und abgebrochene Anfragen an. Der Wert ist `NaN` bei abgelaufenen Anfragen und Antworten oder wenn das Timing ungültig ist.

`contentType:` ***Schnur***

Der Wert des HTTP-Headers vom Inhaltstyp.

`cookies:` ***Reihe***

Ein Array von Objekten, das Cookies darstellt und Eigenschaften wie „domain“ und „expires“ enthält. Die Eigenschaften entsprechen den Attributen jedes Cookies, wie im folgenden Beispiel gezeigt:

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

`encryptionProtocol:` ***Schnur***

Das Protokoll, mit dem die Transaktion verschlüsselt ist.

`headers:` ***Objekt***

Ein Array-ähnliches Objekt, das den Zugriff auf HTTP-Header-Namen und -Werte ermöglicht. Header-Informationen sind über eine der folgenden Eigenschaften verfügbar:

`length:` ***Zahl***

Die Anzahl der Header.

`string property:`

Der Name des Headers, auf den wie ein Wörterbuch zugegriffen werden kann, wie im folgenden Beispiel gezeigt:

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

`numeric property:`

Entspricht der Reihenfolge, in der die Header auf dem Kabel erscheinen. Das zurückgegebene Objekt hat einen Namen und eine Werteigenschaft. Numerische Eigenschaften sind nützlich, um über alle Header zu iterieren und Header mit doppelten Namen zu disambiguieren, wie im folgenden Beispiel gezeigt:

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

Hinweis: Sparen `HTTP.headers` in den Flow-Speicher speichert nicht alle einzelnen Header-Werte. Es hat sich bewährt, die einzelnen Header-Werte im Flow-Speicher zu speichern. Beziehen Sie sich auf Flow Klassenbereich für Details.

`headersRaw:` ***Schnur***

Der unveränderte Block von HTTP-Headern, ausgedrückt als Zeichenfolge.

`host:` ***Schnur***

Der Wert im HTTP-Host-Header.

`isClientReset:` ***Boolesch***

Der Wert ist `true` wenn der HTTP/2-Stream vom Client zurückgesetzt wird. Wenn das Protokoll HTTP1.1 ist, ist der Wert `false`.

`isDesync:` ***Boolesch***

Der Wert ist `true` wenn der Protokollparser aufgrund fehlender Pakete desynchronisiert wurde.

`isEncrypted:` ***Boolesch***

Der Wert ist `true` wenn die Transaktion über sicheres HTTP erfolgt.

`isDecrypted:` ***Boolesch***

Der Wert ist wahr, wenn das ExtraHop-System die Transaktion sicher entschlüsselt und analysiert hat. Die Analyse des entschlüsselten Datenverkehrs kann komplexe Bedrohungen aufdecken, die sich im verschlüsselten Datenverkehr verstecken.

`isPipelined:` ***Boolesch***

Der Wert ist `true` wenn die Transaktion in der Pipeline ist.

`isReqAborted:` ***Boolesch***

Der Wert ist `true` wenn die Verbindung geschlossen wurde, bevor die HTTP-Anfrage abgeschlossen war.

`isRspAborted:` ***Boolesch***

Der Wert ist `true` wenn die Verbindung geschlossen wurde, bevor die HTTP-Antwort abgeschlossen war.

Zugriff nur am `HTTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isRspChunked:` ***Boolesch***

Der Wert ist `true` wenn die Antwort aufgeteilt ist.

Zugriff nur am `HTTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isRspCompressed:` ***Boolesch***

Der Wert ist `true` wenn die Antwort komprimiert ist.

Zugriff nur am `HTTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isServerPush:` ***Boolesch***

Der Wert ist `true` wenn die Transaktion das Ergebnis eines Server-Pushs ist.

`isServerReset:` ***Boolesch***

Der Wert ist `true` wenn der HTTP/2-Stream vom Server zurückgesetzt wird.

`isSQLi:` ***Boolesch***

Der Wert ist wahr, wenn die Anfrage ein oder mehrere verdächtige SQL-Fragmente enthielt. Diese Fragmente weisen auf eine mögliche SQL-Injection (SQLi) hin. SQLi ist eine Technik, bei der ein Angreifer auf Daten zugreifen und diese manipulieren kann, indem er bösartige SQL-Anweisungen in eine SQL-Abfrage einfügt.

`isXSS:` ***Boolesch***

Der Wert ist wahr, wenn die HTTP-Anfrage potenzielle Cross-Site-Scripting-Versuche (XSS) enthielt. Ein erfolgreicher XSS-Versuch kann ein bösartiges clientseitiges Skript oder eine Nutzlast in eine vertrauenswürdige Website oder Anwendung injizieren. Wenn ein Opfer die Website besucht, wird das bösartige Skript dann in den Browser des Opfers injiziert.

`method:` ***Schnur***

Die HTTP-Methode der Transaktion wie POST und GET.

`origin:` ***IP-Adresse | Schnur***

Der Wert im X-Forwarded-For- oder True-Client-IP-Header.

`path:` ***Schnur***

Der Pfadteil der URI: `/path/`.

`payload:` ***Puffer | null***

Das Puffer Objekt, das die rohen Payload-Bytes der Ereignistransaktion enthält. Wenn die Nutzlast komprimiert wurde, wird der dekomprimierte Inhalt zurückgegeben .

Der Puffer enthält *N* erste Byte der Nutzlast, wobei *N* ist die Anzahl der Payload-Bytes, angegeben durch Byte zum Puffer Feld, als der Auslöser über die ExtraHop-WebUI konfiguriert wurde. Die Standardanzahl von Byte ist 2048. Weitere Informationen finden Sie unter Erweiterte Trigger-Optionen.

Das folgende Skript ist ein Beispiel für eine HTTP-Payload-Analyse:

```
// 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) {
        debug("user: " + user[1]);
    }
}
```

Hinweis: Wenn zwei HTTP-Payload-Puffering-Trigger demselben Gerät zugewiesen sind, wird der höhere Wert ausgewählt und der Wert von `HTTP.payload` ist für beide Trigger gleich.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden (entspricht `rspTimeToFirstPayload` - `reqTimeToLastByte`). Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur am `HTTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`query:` ***Schnur***

Der Teil der Abfragezeichenfolge von `URI: query=string`. Dies folgt normalerweise der URL und ist durch ein Fragezeichen von dieser getrennt. Mehrere Abfragezeichenfolgen werden durch ein Und-Zeichen (&) oder ein Semikolon (;) als Trennzeichen getrennt.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `HTTP.commitRecord()`.

Das Standard-Record-Objekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`
- `clientZeroWnd`
- `contentType`
- `host`
- `isPipelined`
- `isReqAborted`
- `isRspAborted`
- `isRspChunked`
- `isRspCompressed`
- `method`
- `origin`
- `query`
- `receiverIsExternal`
- `referer`
- `reqBytes`
- `reqL2Bytes`
- `reqPkts`
- `reqRTO`
- `reqSize`
- `reqTimeToLastByte`
- `roundTripTime`
- `rspBytes`
- `rspL2Bytes`
- `rspPkts`
- `rspRTO`
- `rspSize`
- `rspTimeToFirstHeader`
- `rspTimeToFirstPayload`
- `rspTimeToLastByte`
- `rspVersion`
- `senderIsExternal`
- `serverIsExternal`
- `serverZeroWnd`
- `statusCode`
- `thinkTime`
- `title`

- processingTime
- uri
- userAgent

Greifen Sie nur auf das Datensatzobjekt zu HTTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

referer: ***Schnur***

Der Wert im HTTP-Referrer-Header.

reqBytes: ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur am HTTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqL2Bytes: ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

Zugriff nur am HTTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqPkts: ***Zahl***

Die Anzahl der Anforderungspakete.

Zugriff nur am HTTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqRTO: ***Zahl***

Die Nummer der Anfrage Timeouts für die erneute Übertragung (RTOs).

Zugriff nur am HTTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqSize: ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne HTTP-Header.

reqTimeToLastByte: ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum letzten Byte der Anfrage, ausgedrückt in Millisekunden. Der Wert ist NaN bei abgelaufenen Anfragen und Antworten oder wenn das Timing ungültig ist.

reqZeroWnd: ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

roundTripTime: ***Zahl***

Die mittlere TCP-Roundtrip-Zeit (RTT), ausgedrückt in Millisekunden. Der Wert ist NaN wenn es keine RTT-Proben gibt.

Zugriff nur am HTTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspBytes: ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen L4-Protokoll-Overhead, wie ACKs, Header und Neuübertragungen.

Zugriff nur am HTTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspL2Bytes: ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, z. B. Header.

Zugriff nur am HTTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspPkts: ***Zahl***

Die Anzahl der Antwortpakete.

Zugriff nur am HTTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspRTO: ***Zahl***

Die Anzahl der Antworten Timeouts für die erneute Übertragung (RTOs).

Zugriff nur am HTTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes, ohne HTTP-Header.

Zugriff nur am `HTTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToFirstHeader:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zur Statuszeile, die den Antwortheadern vorausgeht, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist .

Zugriff nur am `HTTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToFirstPayload:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum ersten Nutzdatenbyte der Antwort, ausgedrückt in Millisekunden. Gibt den Wert Null zurück, wenn die Antwort keine Nutzlast enthält. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur am `HTTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToLastByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum letzten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur am `HTTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspVersion:` ***Schnur***

Die HTTP-Version der Antwort.

Zugriff nur am `HTTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`sqli:` ***Reihe von Zeichenketten***

Eine Reihe verdächtiger SQL-Fragmente, die in der Anfrage enthalten sind. Diese Fragmente könnten eine potenzielle SQL-Injection (SQLi) enthalten. SQLi ist eine Technik, bei der ein Angreifer auf Daten zugreifen und diese manipulieren kann, indem er bösartige SQL-Anweisungen in eine SQL-Abfrage einfügt.

`statusCode:` ***Zahl***

Der HTTP-Statuscode der Antwort.

Zugriff nur am `HTTP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

Hinweis: Gibt einen Statuscode von 0 zurück, wenn er nicht gültig ist `HTTP_RESPONSE` wird empfangen.

`streamId:` ***Zahl***

Die ID des Streams, der die Ressource übertragen hat. Da Antworten möglicherweise nicht in der richtigen Reihenfolge zurückgegeben werden, ist diese Eigenschaft für HTTP/2-Transaktionen erforderlich, um Anfragen mit Antworten abzugleichen. Der Wert ist `1` für die HTTP/1.1-Upgrade-Anfrage und `null` für frühere HTTP-Versionen.

`title:` ***Schnur***

Der Wert im Titelelement des HTML-Inhalts, falls vorhanden. Wenn der Titel komprimiert wurde, wird der dekomprimierte Inhalt zurückgegeben.

`thinkTime:` ***Zahl***

Die Zeit, die zwischen der Übertragung der Antwort durch den Server an den verstrichen ist Client und der Client überträgt eine neue Anfrage an den Server, ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine gültige Messung gibt.

`uri:` ***Schnur***

Die URI ohne Abfragezeichenfolge: `f.q.d.n/path/`.

`userAgent:` ***Schnur***

Der Wert im HTTP-User-Agent-Header.

`xss:` ***Reihe von Zeichenketten***

Eine Reihe verdächtiger HTTP-Anforderungsfragmente, die in der Anfrage enthalten sind. Diese Fragmente könnten ein bösartiges clientseitiges Skript oder eine Nutzlast in eine vertrauenswürdige Website oder Anwendung injizieren. Wenn ein Opfer die Website besucht, wird das bösartige Skript dann in den Browser des Opfers injiziert.

Beispiele für Trigger

- Beispiel: HTTP-Antworten auf 500-Ebene nach Kunden-ID und URI verfolgen
- Beispiel: SOAP-Anfragen verfolgen
- Beispiel: Zugriff auf HTTP-Header-Attribute
- Beispiel: Daten in einer Sitzungstabelle aufzeichnen
- Beispiel: Erstellen Sie einen Anwendungscontainer

`IBMMQ`

Die IBMMQ Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `IBMMQ_REQUEST` und `IBMMQ_RESPONSE` Ereignisse.

Hinweis: Das IBM MQ-Protokoll unterstützt die EBCDIC-Kodierung.

Ereignisse

`IBMMQ_REQUEST`

Läuft bei jeder IBM MQ-Anfrage, die vom Gerät verarbeitet wird.

`IBMMQ_RESPONSE`

Läuft auf jeder IBM MQ-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `IBMMQ_REQUEST` oder `IBMMQ_RESPONSE` Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die Standardeigenschaften, die für jedes Ereignis festgeschrieben wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`channel:` ***Schnur***

Der Name des Kommunikationskanals.

`conversationId:` ***Zahl***

Die Kennung für die MQ-Konversation.

`correlationId:` ***Schnur***

Die IBMMQ-Korrelations-ID.

`error:` ***Schnur***

Die Fehlerzeichenfolge, die dem Fehlercode auf dem Kabel entspricht.

`method:` ***Schnur***

Der Name der Anforderungs- oder Antwortmethode für das Wire-Protokoll.

Die folgenden ExtraHop-Methodennamen unterscheiden sich von den Wireshark-Methodennamen:

ExtraHop	Wireshark
`ASYNC_MSG_V7`	`ASYNC_MESSAGE`
`MQCLOSEv7`	`SOCKET_ACTION`
`MQGETv7`	`REQUEST_MSGS`
`MQGETv7_REPLY`	`NOTIFICATION`

`msg:` ***Puffer***

EIN Puffer Objekt, das die Nachrichten MQPUT, MQPUT1, MQGET_REPLY, ASYNC_MSG_V7 und MESSAGE_DATA enthält.

Warteschlangennachrichten, die größer als 32 KB sind, können in mehr als ein Segment aufgeteilt werden. Für jedes Segment wird ein Auslöser ausgeführt, und nur das erste Segment hat eine Nachricht ungleich Null.

Pufferdaten können in eine druckbare Zeichenfolge umgewandelt werden über `toString()` Funktion oder durch Unpack-Befehle formatiert.

`msgFormat:` ***Schnur***

Das Nachrichtenformat.

`msgId:` ***Schnur***

Die IBM MQ-Meldungs-ID.

`pcfError:` ***Schnur***

Die Fehlerzeichenfolge, die dem Fehlercode auf der Leitung für den PCF-Kanal (Programmable Command Formats) entspricht.

`pcfMethod:` ***Schnur***

Der Name der Wire-Protocol-Anforderungs- oder Antwortmethode für den PCF-Kanal (Programmable Command Formats).

`pcfWarning:` ***Schnur***

Die Warnzeichenfolge, die der Warnzeichenfolge auf der Leitung für den PCF-Kanal (Programmable Command Formats) entspricht.

`putAppName:` ***Schnur***

Der mit der MQPUT-Nachricht verknüpfte Anwendungsname.

`queue:` ***Schnur***

Der Name der lokalen Warteschlange. Der Wert ist `null` wenn es keine gibt `MQOPEN`, `MQOPEN_REPLY`, `MQSP1(Open)`, oder `MQSP1_REPLY` Nachricht.

`queueMgr:` ***Schnur***

Der lokale Warteschlangenmanager. Der Wert ist `null` wenn es keine gibt `INITIAL_DATA` Nachricht zu Beginn der Verbindung.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `IBMMQ.commitRecord()` entweder auf einem `IBMMQ_REQUEST` oder `IBMMQ_RESPONSE` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

IBMMQ_REQUEST	IBMMQ_RESPONSE
Kanal	Kanal
Kunde ist extern	Kunde ist extern
Kunde ZeroWND	Kunde ZeroWND
Korrelations-ID	Korrelations-ID
Msgid	Fehler
Methode	Msgid
MSG-Format	Methode
Größe der Nachricht	MSG-Format
Warteschlange	Größe der Nachricht
Warteschlange MGR	Warteschlange
Empfänger ist extern	Warteschlange MGR
ReqByte	Empfänger ist extern
REQL 2 Byte	Gelöste Warteschlange
ReqPKTs	Gelöste Warteschlange Mgr
ReqRTO	Zeit der Hin- und Rückfahrt
Gelöste Warteschlange	RSP-Bytes
Gelöste Warteschlange Mgr	RSPL 2 Byte
Absender ist extern	RSPP Kts
Server ist extern	RSP zu
ServerZeroWND	Absender ist extern
	Server ist extern
	ServerZeroWND
	Warnung

`reqBytes:` ***Zahl***
Die Anzahl der Anforderungsbytes auf Anwendungsebene.

`reqL2Bytes:` ***Zahl***
Die Anzahl der L2 Bytes anfordern.

`reqPkts:` ***Zahl***
Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***
Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

`reqZeroWnd:` ***Zahl***
Die Anzahl der Nullfenster in der Anfrage.

`resolvedQueue:` ***Schnur***
Der Name der aufgelösten Warteschlange von `MQGET_REPLY`, `MQPUT_REPLY`, oder `MQPUT1_REPLY` Nachrichten. Wenn es sich bei der Warteschlange um eine entfernte Warteschlange handelt, unterscheidet sich der Wert von `IBMMQ.queue`.

`resolvedQueueMgr:` ***Schnur***

Der aufgelöste Warteschlangenmanager von `MQGET_REPLY`, `MQPUT_REPLY`, oder `MQPUT1_REPLY`. Wenn es sich bei der Warteschlange um eine entfernte Warteschlange handelt, unterscheidet sich der Wert von `IBMMQ.queueMgr`.

`rfh:` ***Reihe von Zeichenketten***

Ein Array von Zeichenketten, die sich im optionalen Rules and Formatting Header (RFH) befinden. Wenn es keinen RFH-Header gibt oder der Header leer ist, ist das Array leer.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`rspBytes:` ***Zahl***

Die Anzahl der Antwortbytes auf Anwendungsebene.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes.

`rspPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`totalMsgLength:` ***Zahl***

Die Gesamtlänge der Nachricht, ausgedrückt in Byte.

`warning:` ***Schnur***

Die Warnzeichenfolge, die der Warnzeichenfolge auf dem Kabel entspricht.

Beispiele für Trigger

- Beispiel: IBM MQ-Metriken sammeln

`ICA`

Die ICA Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `ICA_OPEN`, `ICA_AUTH`, `ICA_TICK`, und `ICA_CLOSE` Ereignisse.

Ereignisse

`ICA_AUTH`

Wird ausgeführt, wenn die ICA-Authentifizierung abgeschlossen ist.

`ICA_CLOSE`

Wird ausgeführt, wenn die ICA-Sitzung geschlossen ist.

`ICA_OPEN`

Wird unmittelbar nach dem ersten Laden der ICA-Anwendung ausgeführt.

`ICA_TICK`

Wird regelmäßig ausgeführt, während der Benutzer mit der ICA-Anwendung interagiert.

Nach dem `ICA_OPEN` Die Ereignis wurde mindestens einmal durchgeführt, die `ICA_TICK` Das Ereignis wird jedes Mal ausgeführt, wenn die Latenz gemeldet und zurückgegeben wird `clientLatency` oder `networkLatency` Eigenschaften, die unten beschrieben werden.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `ICA_OPEN`, `ICA_TICK`, oder `ICA_CLOSE` Ereignis. Commits aufzeichnen für `ICA_AUTH` Ereignisse werden nicht unterstützt.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die Standardeigenschaften, die für jedes Ereignis festgeschrieben wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`application:` ***Schnur***

Der Name der Anwendung, die gestartet wird.

`authDomain:` ***Schnur***

Die Windows-Authentifizierungsdomäne, zu der der Benutzer gehört.

`channels:` ***Reihe***

Eine Reihe von Objekten, die Informationen über virtuelle Kanäle enthalten, die seit dem letzten Mal beobachtet wurden `ICA_TICK` Ereignis.

Zugriff nur auf `ICA_TICK`Ereignisse; andernfalls tritt ein Fehler auf.

Jedes Objekt enthält die folgenden Eigenschaften:

`name:` ***Schnur***

Der Name des virtuellen Kanals.

`description:` ***Schnur***

Die freundliche Beschreibung des Kanalnamens.

`clientBytes:` ***Zahl***

Die Anzahl der Byte, die von der gesendet wurden Client für diesen Kanal.

`serverBytes:` ***Zahl***

Die Anzahl der vom Server für den Kanal gesendeten Byte.

`clientMachine:` ***Schnur***

Der Name des Client Maschine. Der Name wird vom ICA-Client angezeigt und ist in der Regel der Hostname des Client-Computers.

`clientBytes:` ***Zahl***

Nach einem `ICA_CLOSE` Ereignis, die inkrementelle Anzahl von Client-Bytes auf Anwendungsebene, die seit dem letzten Ereignis beobachtet wurden `ICA_TICK` Ereignis. Gibt nicht die Gesamtzahl der Byte für die Sitzung an.

Zugriff nur auf `ICA_CLOSE` oder `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientCGPMsgCount:` ***Zahl***

Die Anzahl der Client-CGP-Nachrichten seit der letzten `ICA_TICK` Ereignis.

Zugriff nur auf `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientLatency:` ***Zahl***

Die Latenz der Client, ausgedrückt in Millisekunden, wie vom End User Experience Management (EUEM) Beacon gemeldet.

Die Clientlatenz wird gemeldet, wenn ein Paket vom Client auf dem EUEM-Kanal das Ergebnis einer einzelnen ICA-Roundtrip-Messung meldet.

Zugriff nur auf `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientL2Bytes:` ***Zahl***

Nach einem `ICA_CLOSE` Ereignis, die inkrementelle Anzahl von L2 Seit dem letzten beobachtete Client-Bytes `ICA_TICK` Ereignis. Gibt nicht die Gesamtzahl der Byte für die Sitzung an.

Zugriff nur auf `ICA_CLOSE` oder `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientMsgCount:` ***Zahl***

Die Anzahl der Client-Nachrichten seit der letzten `ICA_TICK` Ereignis.

Zugriff nur auf `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientPkts:` ***Zahl***

Nach einem `ICA_CLOSE` Ereignis, die inkrementelle Anzahl von Client-Paketen, die seit dem letzten beobachtet wurden `ICA_TICK` Ereignis. Gibt nicht die Gesamtzahl der Pakete für die Sitzung an.

Zugriff nur auf `ICA_CLOSE` oder `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientRTO:` ***Zahl***

Nach einem `ICA_CLOSE` Ereignis, die inkrementelle Anzahl von Client Timeouts bei der erneuten Übertragung (RTOs) beobachtet seit dem letzten `ICA_TICK` Ereignis. Gibt nicht die Gesamtzahl der RTOs für die Sitzung an.

Zugriff nur auf `ICA_CLOSE` oder `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientZeroWnd:` ***Zahl***

Die Anzahl der vom Client gesendeten Nullfenster.

Zugriff nur auf `ICA_CLOSE` oder `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientType:` ***Schnur***

Der Typ des ICA-Clients, bei dem es sich um den Benutzeragenten handelt, der ICA entspricht.

`clipboardData:` ***Puffer***

EIN Puffer Objekt, das Rohdaten aus der Zwischenablage enthält.

Der Wert ist `null` wenn der `ICA_TICK` Das Ereignis ist nicht auf eine Datenübertragung in der Zwischenablage zurückzuführen, oder wenn der von `tickChannel` Die Eigenschaft ist kein Zwischenablagekanal.

Die maximale Anzahl von Bytes im Puffer wird angegeben durch Bytes aus der Zwischenablage in den Puffer Feld, wenn der Auslöser über das ExtraHop-System konfiguriert wurde. Die standardmäßige maximale Objektgröße beträgt 1024 Byte. Weitere Informationen finden Sie in der Erweiterte Trigger-Optionen.

Um die Richtung der Datenübertragung in der Zwischenablage zu bestimmen, greifen Sie auf diese Eigenschaft zu `Flow.sender`, `Flow.receiver`, `Flow.client`, oder `Flow.server`.

Zugriff nur auf `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clipboardDataType:` ***Schnur***

Der Datentyp bei der Übertragung in die Zwischenablage. Die folgenden Typen von Zwischenablagen werden unterstützt:

- `TEXT`
- `BITMAP`
- `METAFILEPICT`
- `SYMLINK`
- `DIF`
- `TIFF`
- `OEMTEXT`
- `DIB`
- `PALLETTE`
- `PENDATA`

- RIFF
- WAVE
- UNICODETEXT
- EHNMETAFILE
- OWNERDISPLAY
- DSPTEXT
- DSPBITMAP
- DSPMETAFILEPICT
- DSPENHMETAFILE

Der Wert ist null wenn der ICA_TICK Das Ereignis ist nicht auf eine Datenübertragung in der Zwischenablage zurückzuführen, oder wenn der von tickChannel Die Eigenschaft ist kein Zwischenablagekanal.

Zugriff nur auf ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

frameCutDuration: ***Zahl***

Die Dauer des Frame-Cuts, wie vom EUEM-Beacon gemeldet.

Zugriff nur auf ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

frameSendDuration: ***Zahl***

Die Dauer des Frame-Sendens, wie vom EUEM-Beacon gemeldet.

Zugriff nur auf ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

host: ***Schnur***

Der Hostname des Citrix-Servers.

isAborted: ***Boolescher Wert***

Der Wert ist true wenn die Anwendung nicht erfolgreich gestartet werden kann.

Zugriff nur auf ICA_CLOSE Ereignisse; andernfalls tritt ein Fehler auf.

isCleanShutdown: ***Boolescher Wert***

Der Wert ist true wenn die Anwendung ordnungsgemäß heruntergefahren wird.

Zugriff nur auf ICA_CLOSE Ereignisse; andernfalls tritt ein Fehler auf.

isClientDiskRead: ***Boolescher Wert***

Der Wert ist true wenn eine Datei von der Clientdiskette auf den Citrix-Server gelesen wurde. Der Wert ist null wenn es sich bei dem Befehl nicht um eine Dateioperation handelt, oder wenn der Kanal, der durch den tickChannel Eigenschaft ist kein Dateikanal.

Zugriff nur auf ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

isClientDiskWrite: ***Boolescher Wert***

Der Wert ist true wenn eine Datei vom Citrix-Server auf die Clientdiskette geschrieben wurde. Der Wert ist null wenn es sich bei dem Befehl nicht um eine Dateioperation handelt, oder wenn der Kanal, der durch den tickChannel Eigenschaft ist kein Dateikanal.

Zugriff nur auf ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

isEncrypted: ***Boolescher Wert***

Der Wert ist true wenn die Anwendung mit RC5-Verschlüsselung verschlüsselt ist.

isSharedSession: ***Boolescher Wert***

Der Wert ist true wenn die Anwendung über eine bestehende Verbindung gestartet wird.

launchParams: ***Schnur***

Die Zeichenfolge, die die Parameter darstellt.

loadTime: ***Zahl***

Die Ladezeit der angegebenen Anwendung, ausgedrückt in Millisekunden.

Hinweis: Die Ladezeit wird nur für das erste Laden der Anwendung aufgezeichnet. Das ExtraHop-System misst nicht die Ladezeit für Anwendungen, die über bestehende Sitzungen gestartet wurden, sondern meldet stattdessen die anfängliche Ladezeit bei nachfolgenden Anwendungsladevorgängen. Wählen `ICA.isSharedSession` um zwischen anfänglichen und nachfolgenden Anwendungsladevorgängen zu unterscheiden.

`loginTime:` ***Zahl***

Die Benutzeranmeldezeit, ausgedrückt in Millisekunden.

Zugriff nur auf `ICA_OPEN`, `ICA_CLOSE`, oder `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

Hinweis: Die Anmeldezeit wird nur für das erste Laden der Anwendung aufgezeichnet. Das ExtraHop-System misst nicht die Anmeldezeit für Anwendungen, die über bestehende Sitzungen gestartet wurden, sondern meldet stattdessen die anfängliche Anmeldezeit bei nachfolgenden Anwendungsladevorgängen. Wähle `ICA.isSharedSession` um zwischen anfänglichen und nachfolgenden Anwendungsladevorgängen zu unterscheiden.

`networkLatency:` ***Zahl***

Die aktuelle Latenz, angekündigt von Client, ausgedrückt in Millisekunden.

Netzwerklatenz wird gemeldet, wenn ein bestimmtes ICA-Paket vom Client Latenzinformationen enthält.

Zugriff nur auf `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`payload:` ***Puffer***

Die Puffer Objekt, das die rohen Payload-Bytes der Datei enthält, die bei dem Ereignis gelesen oder geschrieben wurde.

Der Puffer enthält den *N* erste Byte der Nutzlast, wobei *N* ist die Anzahl der Payload-Bytes, spezifiziert durch Bytes zum Puffer Feld , wenn der Auslöser über die ExtraHop WebUI konfiguriert wurde. Die Standardanzahl von Bytes ist 2048. Weitere Informationen finden Sie unter Erweiterte Trigger-Optionen.

Der Wert ist `null` wenn der Kanal spezifiziert ist durch `tickChannel` Eigenschaft ist kein Dateikanal.

Zugriff nur auf `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`printerName:` ***Schnur***

Der Name des Druckertreiber.

Zugriff nur auf `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`program:` ***Schnur***

Der Name des Programms oder der Anwendung, die gestartet wird.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `ICA.commitRecord()` auf entweder einem `ICA_OPEN`, `ICA_TICK`, oder `ICA_CLOSE` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

ICA_CLOSE	ICA_OPEN	ICA_TICK
authDomain	authDomain	authDomain
clientBytes	clientIsExternal	clientIsExternal
clientIsExternal	clientMachine	clientBytes

ICA_CLOSE	ICA_OPEN	ICA_TICK
clientL2Bytes	clientType	clientCGPMsgCount
clientMachine	clientZeroWnd	clientL2Bytes
clientPkts	host	clientLatency
clientRTO	isEncrypted	clientMachine
clientType	isSharedSession	clientMsgCount
clientZeroWnd	launchParams	clientPkts
host	loadTime	clientRTO
isAborted	loginTime	clientType
isCleanShutdown	program	clientZeroWnd
isEncypted	receiverIsExternal	frameCutDuration
isSharedSession	senderIsExternal	frameSendDuration
launchParams	serverIsExternal	host
loadTime	serverZeroWnd	isClientDiskRead
loginTime	user	isClientDiskWrite
program		isEncrypted
receiverIsExternal		isSharedSession
roundTripTime		launchParams
senderIsExternal		loadTime
serverBytes		loginTime
serverIsExternal		networkLatency
serverL2Bytes		program
serverPkts		receiverIsExternal
serverRTO		resource
serverZeroWnd		roundTripTime
user		senderIsExternal
		serverBytes
		serverCGPMsgCount
		serverIsExternal
		serverL2Bytes
		serverMsgCount
		serverPkts
		serverRTO
		serverZeroWnd
		tickChannel

ICA_CLOSE	ICA_OPEN	ICA_TICK
		user

Greifen Sie nur auf das Datensatzobjekt zu ICA_OPEN, ICA_CLOSE, und ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`resource:` ***Schnur***

Der Pfad der Datei, die bei dem Ereignis gelesen oder geschrieben wurde, falls bekannt. Der Wert ist `null` wenn der Kanal spezifiziert ist durch `tickChannel` Eigenschaft ist kein Dateikanal.

Zugriff nur auf ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`resourceOffset:` ***Zahl***

Der Offset der Datei, die bei dem Ereignis gelesen oder geschrieben wurde, falls bekannt. Der Wert ist `null` wenn der Kanal spezifiziert ist durch `tickChannel` Eigenschaft ist kein Dateikanal.

Zugriff nur auf ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

Zugriff nur auf ICA_CLOSE oder ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`serverBytes:` ***Zahl***

Nach einem ICA_CLOSE Ereignis, die inkrementelle Anzahl von Server-Bytes auf Anwendungsebene, die seit dem letzten Ereignis beobachtet wurden ICA_TICK Ereignis. Gibt nicht die Gesamtzahl der Byte für die Sitzung an.

Zugriff nur auf ICA_CLOSE oder ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`serverCGPMsgCount:` ***Zahl***

Die Anzahl der CGP-Servernachrichten seit der letzten ICA_TICK Ereignis.

Zugriff nur auf ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`serverL2Bytes:` ***Zahl***

Nach einem ICA_CLOSE Ereignis, die inkrementelle Anzahl von L2 Server-Bytes, die seit dem letzten beobachtet wurden ICA_TICK Ereignis. Gibt nicht die Gesamtzahl der Byte für die Sitzung an.

Zugriff nur auf ICA_CLOSE oder ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`serverMsgCount:` ***Zahl***

Die Anzahl der Servernachrichten seit der letzten ICA_TICK Ereignis.

Zugriff nur auf ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`serverPkts:` ***Zahl***

Nach einem ICA_CLOSE Ereignis, die inkrementelle Anzahl von Serverpaketen, die seit dem letzten beobachtet wurden ICA_TICK Ereignis. Gibt nicht die Gesamtzahl der Pakete für die Sitzung an.

Zugriff nur auf ICA_CLOSE oder ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`serverRTO:` ***Zahl***

Nach einem ICA_CLOSE Ereignis, die inkrementelle Anzahl von Server Timeouts bei der erneuten Übertragung (RTOs) beobachtet seit dem letzten ICA_TICK Ereignis. Gibt nicht die Gesamtzahl der RTOs für die Sitzung an.

Zugriff nur auf ICA_CLOSE oder ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`serverZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster, die vom Server gesendet wurden.

Zugriff nur auf ICA_CLOSE oder ICA_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`tickChannel:` ***Schnur***

Der Name des virtuellen Kanals, der zum aktuellen `ICA_TICK` Ereignis. Die folgenden Kanäle werden unterstützt:

- **CTXCLI:** Zwischenablage
- **CTXCDM:** Datei
- **CTXEUE:** Überwachung der Endbenutzererfahrung

Zugriff nur auf `ICA_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`user:` ***Schnur***

Der Name des Benutzers, falls verfügbar.

ICMP

Die ICMP Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `ICMP_MESSAGE` Ereignisse.

Ereignisse

`ICMP_MESSAGE`

Läuft auf jeder ICMP-Meldung, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `ICMP_MESSAGE` Ereignis.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`gwAddr:` ***IP-Addresse***

Gibt bei einer Umleitungsnachricht die Adresse des Gateways zurück, an das der Datenverkehr für das Netzwerk gesendet werden soll, das im Feld Internet-Zielnetzwerk der Daten des ursprünglichen Datagramms angegeben ist. Gibt Null für alle anderen Nachrichten zurück.

Nachricht	ICMPv4-Typ	ICMPv6-Typ
`Redirect Message`	`5`	`n/a`

`hopLimit:` ***Zahl***

Die Live-Zeit oder die Hop-Zählung des ICMP-Pakets.

`isError:` ***Boolescher Wert***

Der Wert ist `true` für Nachrichtentypen in der folgenden Tabelle.

Nachricht	ICMPv4-Typ	ICMPv6-Typ
`Destination Unreachable`	`3`	`1`
`Redirect`	`5`	`n/a`
`Source Quench`	`4`	`n/a`

Nachricht	ICMPv4-Typ	ICMPv6-Typ
Time Exceeded	11	3
Parameter Problem	12	4
Packet Too Big	n/a	2

`isQuery:` ***Boolescher Wert***

Der Wert ist `true` für Nachrichtentypen in der folgenden Tabelle.

Nachricht	ICMPv4-Typ	ICMPv6-Typ
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
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:` ***Boolescher Wert***

Der Wert ist `true` für Nachrichtentypen in der folgenden Tabelle.

Nachricht	ICMPv4-Typ	ICMPv6-Typ
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

Nachricht	ICMPv4-Typ	ICMPv6-Typ
Home Agent Address Discovery Reply Message	n/a	145
Mobile Prefix Advertisement	n/a	147
Certification Path Advertisement	n/a	149

`msg:` ***Puffer***

Ein Pufferobjekt mit bis zu `message_length_max` Byte der ICMP-Meldung. Die `message_length_max` Die Option ist im ICMP-Profil in der laufenden Konfiguration konfiguriert.

Das folgende Beispiel für eine laufende Konfiguration ändert den ICMP `message_length_max` von der Standardeinstellung von 4096 Byte auf 1234 Byte:

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

Hinweis: Sie können das Pufferobjekt mit der Methode String.fromCharcode in eine Zeichenfolge konvertieren. Um die Zeichenfolge im Laufzeitprotokoll anzuzeigen, führen Sie die Methode JSON.stringify aus, wie im folgenden Beispielcode gezeigt:

```
const icmp_msg = String.fromCharCode.apply(String,
 ICMP.msg);
debug('ICMP message text: ' + JSON.stringify(icmp_msg,
 null, 4));
```

Sie können die ICMP-Nachrichtenzeichenfolgen auch mit den Includes- und Testmethoden durchsuchen, wie im folgenden Beispielcode gezeigt:

```
const substring_search = 'search term';
const regex_search = '^search term$';
const icmp_msg = String.fromCharCode.apply(String,
 ICMP.msg);

if (icmp_msg.includes(substring_search){
    debug('ICMP message includes substring');
}
if (regex_search.test(icmp_msg)){
    debug('ICMP message matches regex');
}
```

`msgCode:` ***Zahl***

Der ICMP-Meldungscode.

`msgId:` ***Zahl***

Die ICMP-Nachrichten-ID für Echo Request-, Echo Reply-, Timestamp Request-, Timestamp Reply-, Information Request- und Information Reply-Nachrichten. Der Wert ist `null` für alle anderen Nachrichtentypen.

In der folgenden Tabelle werden Typ-IDs für die ICMP-Meldungen angezeigt:

Nachricht	ICMPv4-Typ	ICMPv6-Typ
Echo Request	8	128

Nachricht	ICMPv4-Typ	ICMPv6-Typ
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:` ***Zahl***

Die Länge der ICMP-Meldung, ausgedrückt in Byte.

`msgText:` ***Schnur***

Der beschreibende Text für die Nachricht (z. B. Echoanforderung oder Port nicht erreichbar).

`msgType:` ***Zahl***

Der ICMP-Nachrichtentyp.

Die folgende Tabelle zeigt die verfügbaren ICMPv4-Nachrichtentypen :

Typ	Nachricht
0	Echo Reply
1 and 2	Unassigned
3	Destination Unreachable
4	Source Quench
5	Redirect Message
6	Alternate Host Address (deprecated)
7	Unassigned
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)
19	Reserved
20-29	Reserved
30	Traceroute (deprecated)
31	Datagram Conversion Error (deprecated)

Typ	Nachricht
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	Extended Echo Request
43	Extended Echo Reply
44-255	Unassigned

Die folgende Tabelle zeigt die verfügbaren ICMPv6-Nachrichtentypen:

Typ	Nachricht
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
138	Router Renumbering

Typ	Nachricht
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
150	ICMP messages utilized by experimental mobility protocols such as Seamoby
151	Multicast Router Advertisement
152	Multicast Router Solicitation
153	Multicast Router Termination
155	RPL Control Message
156	ILNPv6 Locator Update Message
157	Duplicate Address Request
158	Duplicate Address Confirmation
159	MPL Control Message
160	Extended Echo Request - No Error
161	Extended Echo Reply
200	Private Experimentation
201	Private Experimentation
255	Reserved for expansion of ICMPv6 informational messages

`nextHopMTU:` ***Zahl***

Ein ICMPv4 `Ziel nicht erreichbar` oder ein ICMPv6 `Paket zu groß` Nachricht, die maximale Übertragungseinheit der Next-Hop-Verbindung. Der Wert ist `null` für alle anderen Nachrichten.

Nachricht	ICMPv4-Typ	ICMPv6-Typ
Destination Unreachable	3	n/a
Packet Too Big	n/a	2

`original:` ***Objekt***

Ein Objekt, das die folgenden Elemente aus dem IP-Datagramm enthält, das das Senden der ICMP-Nachricht verursacht hat:

`ipproto:` ***Schnur***

Das IP-Protokoll des Datagramms, z. B. TCP, UDP, ICMP oder ICMPv6.

`ipver:` ***Schnur***

Die IP-Version des Datagramms, z. B. IPv4 oder IPv6.

`srcAddr:` ***IP-Addresse***

Die IPAddress des Datagramm-Absenders.

`srcPort:` ***Zahl***

Die Portnummer des Datagrammsenders.

`dstAddr:` ***IP-Addresse***

Die IPAddress des Datagrammempfängers.

`dstPort:` ***Zahl***

Die Portnummer des Datagrammempfängers.

Der Wert ist `null` wenn der Internet-Header und die 64 Bit des Originaldatagramms nicht in der Nachricht vorhanden sind oder wenn das IP-Protokoll nicht TCP oder UDP ist.

Zugriff nur auf `ICMP_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`pointer:` ***Zahl***

Bei einer Parameter-Problem-Meldung das Oktett des Headers des ursprünglichen Datagramms, in dem der Fehler erkannt wurde. Der Wert ist `null` für alle anderen Nachrichten.

Nachricht	ICMPv4-Typ	ICMPv6-Typ
`Parameter Problem`	`12`	`4`

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `ICMP.commitRecord()` auf einem `ICMP_MESSAGE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`
- `gwAddr`
- `hopLimit`
- `msgCode`
- `msgId`
- `msgLength`
- `msgText`
- `msgType`
- `nextHopMTU`
- `pointer`
- `receiverIsExternal`
- `senderIsExternal`
- `serverIsExternal`
- `seqNum`
- `version`

`seqNum:` ***Zahl***

Die ICMP-Sequenznummer für Echo Request-, Echo Reply-, Timestamp Request-, Timestamp Reply-, Information Request- und Information Reply-Nachrichten. Der Wert ist `null` für alle anderen Nachrichten.

`version:` ***Zahl***

Die Version des ICMP-Nachrichtentyps, der ICMPv4 oder ICMPv6 sein kann.

`Kerberos`

Die Kerberos Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen KERBEROS_REQUEST und KERBEROS_RESPONSE Ereignisse.

Ereignisse

KERBEROS_REQUEST

Läuft auf allen Kerberos-AS-REQ- und TGS-REQ-Nachrichtentypen, die vom Gerät verarbeitet werden.

KERBEROS_RESPONSE

Läuft auf allen Kerberos-AS-REP- und TGS-REP-Nachrichtentypen, die vom Gerät verarbeitet werden.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem KERBEROS_REQUEST oder KERBEROS_RESPONSE Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die für jedes Ereignis festgeschriebenen Standardeigenschaften finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`addresses:` ***Reihe von Objekten***

Die Adressen, von denen aus das angeforderte Ticket gültig ist.

Zugriff nur auf KERBEROS_REQUEST Ereignisse; andernfalls tritt ein Fehler auf.

`apOptions:` ***Objekt***

Ein Objekt, das boolesche Werte für jedes Optionsflag in AP_REQ-Nachrichten enthält.

Zugriff nur auf KERBEROS_REQUEST Ereignisse; andernfalls tritt ein Fehler auf.

`clientPrincipalName:` ***Schnur***

Der Prinzipalname des Client.

`cNames:` ***Reihe von Zeichenketten***

Die Namensteile des Prinzipalbezeichners.

`cNameType:` ***Schnur***

Der Typ für das CNAMES-Feld.

`cRealm:` ***Schnur***

Der Kundenbereich.

`eData:` ***Puffer***

Zusätzliche Informationen zu dem in der Antwort zurückgegebenen Fehler.

Zugriff nur auf KERBEROS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`error:` ***Schnur***

Der Fehler ist zurückgekehrt.

Zugriff nur auf KERBEROS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`from:` ***Schnur***

In den Nachrichtentypen AS_REQ und TGS_REQ die Uhrzeit, auf die das angeforderte Ticket vordatiert werden soll.

Zugriff nur auf KERBEROS_REQUEST Ereignisse; andernfalls tritt ein Fehler auf.

`isAccountPrivileged:` ***Boolescher Wert***

Der Wert ist wahr, wenn das in der `clientPrincipalName` Eigentum ist privilegiert.

`kdcOptions:` ***Objekt***

Ein Objekt, das boolesche Werte für jedes Optionsflag in AS_REQ- und TGS_REQ-Nachrichten enthält.

Zugriff nur auf KERBEROS_REQUEST Ereignisse; andernfalls tritt ein Fehler auf.

`msgType:` ***Schnur***

Der Nachrichtentyp. Mögliche Werte sind:

- AP_REP
- AP_REQ
- AS_REP
- AS_REQAUTHENTICATOR
- ENC_AS_REP_PART
- ENC_KRB_CRED_PART
- ENC_KRB_PRIV_PART
- ENC_P_REP_PART
- ENC_TGS_REP_PART
- ENC_TICKET_PART
- KRB_CRED
- KRB_ERROR
- KRB_PRIV
- KRB_SAFE
- TGS_REP
- TGS_REQ
- TICKET

`paData:` ***Reihe von Objekten***

Die Daten vor der Authentifizierung.

`processingTime:` ***Zahl***

Die Verarbeitungszeit, ausgedrückt in Millisekunden.

Zugriff nur auf KERBEROS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`realm:` ***Schnur***

Der Serverbereich. In einem AS_REQ-Nachrichtentyp ist dies der Client-Bereich.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `Kerberos.commitRecord()` entweder auf einem KERBEROS_REQUEST oder KERBEROS_RESPONSE Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

KERBEROS_REQUEST	KERBEROS_RESPONSE
clientIsExternal	clientIsExternal
cNames	cNames
cNameType	cNameType
cRealm	cRealm
clientZeroWnd	clientZeroWnd
encryptedTicketLength	encryptedTicketLength
eType	error
from	msgType
isAccountPrivileged	isAccountPrivileged
msgType	processingTime
realm	realm
receiverIsExternal	receiverIsExternal
reqBytes	roundTripTime
reqL2Bytes	rspBytes
reqPkts	rspL2Bytes
reqRTO	rspPkts
senderIsExternal	rspRTO
serverZeroWnd	senderIsExternal
sNames	serverIsExternal
sNameType	sNames
ticketETypeName	sNameType
till	ticketETypeName
	serverZeroWnd

`reqETypes:` ***Reihe von Zahlen***

Eine Reihe von Zahlen, die bevorzugten Verschlüsselungsmethoden entsprechen.

Verschlüsselungsmethode	Zahl
ntlm-hash	-150
aes256-cts-hmac-sha1-96-plain	-149
aes128-cts-hmac-sha1-96-plain	-148
rc4-plain-exp	-141
rc4-plain	-140
rc4-plain-old-exp	-136
rc4-hmac-old-exp	-135
rc4-plain-old	-134

Verschlüsselungsmethode	Zahl
rcr-hmac-old	-133
des-plain	-132
rc4-sha	-131
rc4-lm	-130
rc4-plain2	-129
rc4-md4	-128
null	0
des-cbc-crc	1
des-cbc-md4	2
des-cbc-md5	3
des3-cbc-md5	5
des3-cbc-sha1	7
dsaWithSHA1-CmsOID	9
md5WithRSAEncryption-CmsOID	10
sha1WithRSAEncryption-CmsOID	11
rc2CBC-EnvOID	12
rsaEncryption-EnvOID	13
rsaES-OAEP-ENV-OID	14
des-ede3-cbc-Env-OID	15
des3-cbc-sha1-kd	16
aes128-cts-hmac-sha1-96	17
aes256-cts-hmac-sha1-96	18
aes128-cts-hmac-sha256-128	19
aes256-cts-hmac-sha384-192	20
rc4-hmac	23
rc4-hmac-exp	24
camellia128-cts-cmac	25
camellia256-cts-cmac	26
subkey-keymaterial	65

`reqETypeNames:` ***Reihe von Zeichenketten***

Eine Reihe der bevorzugten Verschlüsselungsmethoden.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`serverPrincipalName:` ***Schnur***

Der Serverprinzipalname (SPN).

`sNames:` ***Reihe von Zeichenketten***

Die Namensteile der Serverprinzipal-ID.

`sNameType:` ***Schnur***

Der Typ für das SNames-Feld.

`ticket:` ***Objekt***

Ein neu generiertes Ticket in einer AP_REP-Nachricht oder ein Ticket zur Authentifizierung des Client gegenüber dem Server in einer AP_REQ-Nachricht.

`till:` ***Schnur***

Das vom Client in einer Ticketanfrage angeforderte Ablaufdatum.

Zugriff nur auf `KERBEROS_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

LDAP

Die `LDAP` Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `LDAP_REQUEST` und `LDAP_RESPONSE` Ereignisse.

Ereignisse

`LDAP_REQUEST`

Wird bei jeder LDAP-Anfrage ausgeführt, die vom Gerät verarbeitet wird.

`LDAP_RESPONSE`

Läuft auf jeder LDAP-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `LDAP_REQUEST` oder `LDAP_RESPONSE` Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die für jedes Ereignis festgeschriebenen Standardeigenschaften finden Sie unter `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`bindDN:` ***Schnur***

Der Bind-DN der LDAP-Anfrage.

Zugriff nur auf `LDAP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`controls:` ***Reihe von Objekten***

Ein Array von Objekten, das die LDAP-Steuerelemente der LDAP-Anfrage enthält. Jedes Objekt enthält die folgenden Eigenschaften:

`controlType:` ***Schnur***

Die OID des LDAP-Steuerelements.

`criticality:` ***Boolescher Wert***

Zeigt an, ob das Steuerelement erforderlich ist. Wenn `criticality` ist eingestellt auf `true`, sollte der Server die Steuerung verarbeiten oder die Operation fehlschlagen.

`controlValue:` ***Puffer***

Der optionale Kontrollwert, der zusätzliche Informationen darüber angibt, wie das Steuerelement verarbeitet werden soll.

Zugriff nur auf `LDAP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`dn:` ***Schnur***

Der LDAP LDAP-Name (DN). Wenn kein DN gesetzt ist, `<ROOT>` wird stattdessen zurückgegeben.

`encryptionProtocol:` ***Schnur***

Das Protokoll, mit dem die Transaktion verschlüsselt ist.

`error:` ***Schnur***

Die kurze LDAP-Fehlerzeichenfolge, wie sie in der definiert ist Protokoll (zum Beispiel NoSuchObject).

Zugriff nur auf `LDAP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

Ergebniscode	Ergebnis-Zeichenfolge
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
36	aliasDeferencingProblem
48	inappropriateAuthentication
49	invalidCredentials
50	insufficientAccessRights
51	busy
52	unavailable

Ergebniscode	Ergebnis-Zeichenfolge
53	unwillingToPerform
54	loopDetect
64	namingViolation
65	objectClassViolation
66	notAllowedOnNonLeaf
67	notAllowedOnRDN
68	entryAlreadyExists
69	objectClassModsProhibited
71	affectsMultipleDSAs
80	other

`errorDetail:` ***Schnur***

Die LDAP-Fehlerdetails, falls für den Fehlertyp verfügbar. Beispiel: „ProtocolError: Historische Protokollversion angefordert, verwenden Sie stattdessen LDAPv3.“

Zugriff nur auf `LDAP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isEncrypted:` ***Boolescher Wert***

Der Wert ist wahr, wenn die Transaktion mit SSL oder TLS verschlüsselt ist.

`isDecrypted:` ***Boolescher Wert***

Der Wert ist wahr, wenn das ExtraHop-System die Transaktion sicher entschlüsselt und analysiert hat. Durch die Analyse des entschlüsselten Datenverkehrs können komplexe Bedrohungen aufgedeckt werden, die sich im verschlüsselten Verkehr verstecken.

`isPasswordEmpty:` ***Boolescher Wert***

Der Wert ist wahr, wenn die Anfrage kein Passwort für die Authentifizierung angibt.

Zugriff nur auf `LDAP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`isSigned:` ***Boolescher Wert***

Der Wert ist wahr, wenn die LDAP-Transaktion vom Quellcomputer signiert wurde.

`method:` ***Schnur***

Die LDAP-Methode.

`msgSize:` ***Zahl***

Die Größe der LDAP-Nachricht, ausgedrückt in Byte.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei fehlerhaften und abgebrochenen Antworten, wenn das Timing ungültig ist oder wenn das Timing nicht verfügbar ist. Verfügbar für Folgendes:

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

Gilt nur für LDAP_RESPONSE Ereignisse.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `LDAP.commitRecord()` entweder auf einem LDAP_REQUEST oder LDAP_RESPONSE Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

LDAP_REQUEST	LDAP_RESPONSE
bindDN	clientIsExternal
clientIsExternal	clientZeroWnd
clientZeroWnd	dn
dn	error
isSigned	isSigned
method	errorDetail
msgSize	method
receiverIsExternal	msgSize
reqBytes	processingTime
reqL2Bytes	receiverIsExternal
reqPkts	roundTripTime
reqRTO	rspBytes
saslMechanism	rspL2Bytes
searchFilter	rspPkts
searchScope	rspRTO
senderIsExternal	saslMechanism
serverIsExternal	senderIsExternal
serverZeroWnd	serverIsExternal
	serverZeroWnd

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`saslMechanism:` ***Schnur***

Die Zeichenfolge, die den SASL-Mechanismus definiert, der einen Benutzer identifiziert und gegenüber einem Server authentifiziert.

`searchAttributes:` ***Reihe***

Die Attribute, die von Objekten zurückgegeben werden sollen, die den Filterkriterien entsprechen.

Zugriff nur auf `LDAP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`searchFilter:` ***Schnur***

Der Mechanismus, um bestimmte Einträge im Unterbaum zuzulassen und andere auszuschließen.

Zugriff nur auf `LDAP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`searchResults:` ***Reihe von Objekten***

Ein Array von Objekten, das die in einer LDAP-Antwort zurückgegebenen Suchergebnisse enthält. Jedes Objekt enthält die folgenden Eigenschaften:

`type:` ***Schnur***

Die Art des Suchergebnisses.

`values:` ***Reihe von Puffern***

Ein Array von Buffer-Objekten, die die Suchergebniswerte enthalten.

Zugriff nur auf `LDAP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`searchScope:` ***Schnur***

Die Tiefe einer Suche innerhalb der Suchbasis.

Zugriff nur auf `LDAP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`LLDP`

Die LLDP Mit dieser Klasse können Sie auf Eigenschaften zugreifen `LLDP_FRAME` Ereignisse.

Ereignisse

`LLDP_FRAME`

Läuft auf jedem LLDP-Frame, der vom Gerät verarbeitet wird.

Eigenschaften

`chassisId:` ***Puffer***

Die Chassis-ID, die aus dem ChassisID-Datenfeld abgerufen wurde, oder der Typlängenwert (TLV).

`chassisIdSubtype:` ***Zahl***

Der Chassis-ID-Subtyp, der aus dem ChassisID-TLV abgerufen wurde.

`destination:` ***Schnur***

Die Ziel-MAC-Adresse. Die Ziel-MAC-Adresse. Die häufigsten Ziele sind `01-80-C2-00-00-00`, `01-80-C2-00-00-03` und `01-80-C2-00-00-0E`, gibt Multicast-Adressen an.

`optTLVs:` ***Reihe***

Ein Array, das die optionalen TLVs enthält. Jedes TLV ist ein Objekt mit den folgenden Eigenschaften:

`customSubtype:` ***Zahl***

Der Subtyp eines organisationsspezifischen TLV.

`isCustom:` ***Boolescher Wert***

Gibt true zurück, wenn es sich bei dem Objekt um ein organisationsspezifisches TLV handelt.

`oui:` ***Zahl***

Die organisatorisch eindeutige Kennung für organisationsspezifische TLVs.

`type:` ***Zahl***

Der Typ von TLV.

`value:` ***Schnur***

Der Wert des TLV.

`portId:` ***Puffer***

Die Port-ID, die von der Port-ID TLV abgerufen wurde.

`portIdSubtype:` ***Zahl***

Der Port-ID-Subtyp, der aus dem Port-ID-TLV abgerufen wurde.

`source:` ***Gerät***

Das Gerät, das den LLDP-Frame sendet.

`ttl:` ***Zahl***

Die Zeit bis zum Leben, ausgedrückt in Sekunden. Dies ist der Zeitraum, für den die Informationen in diesem Frame gültig sind, beginnend mit dem Zeitpunkt, an dem die Informationen empfangen wurden.

`LLMNR`

Die `LLMNR` Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `LLMNR_REQUEST` und `LLMNR_RESPONSE` Ereignisse.

Ereignisse

`LLMNR_REQUEST`

Läuft bei jeder LLMNR-Anfrage, die vom Gerät verarbeitet wird.

`LLMNR_RESPONSE`

Läuft auf jeder LLMNR-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `LLMNR_REQUEST` oder `LLMNR_RESPONSE` Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`answer:` ***Objekt***

Ein Objekt, das einem Antwortressourceneintrag entspricht.

Zugriff nur auf `LLMNR_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

Die Objekte enthalten die folgenden Eigenschaften:

`data:` ***Schnur | IP-Adresse***

Der Wert der Daten hängt vom Typ ab. Der Wert ist `null` für nicht unterstützte Datensatztypen. Zu den unterstützten Datensatztypen gehören:

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

`name:` ***Schnur***

Der Name des Datensatz.

`ttl:` ***Zahl***

Der Time-to-Live-Wert.

`type:` ***Schnur***

Der LLMNR-Datensatztyp.

`error:` ***Schnur***

Der Name des LLMNR-Fehlercodes gemäß den IANA-LLMNR-Parametern.

Gibt OTHER für Fehlercodes zurück, die vom System nicht erkannt werden; jedoch `errorNum` gibt den numerischen Codewert an.

Zugriff nur auf `LLMNR_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`errorNum:` ***Zahl***

Die numerische Darstellung des LLMNR-Fehlercodes gemäß den IANA-LLMNR-Parametern.

Zugriff nur auf `LLMNR_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`opcode:` ***Schnur***

Der Name des LLMNR-Operationscodes gemäß den IANA-LLMNR-Parametern. Die folgenden Codes werden vom ExtraHop-System erkannt:

OP-Code	Name
0	Query
1	IQuery (Inverse Query - Obsolete)
2	Status

OP-Code	Name
3	Unassigned
4	Notify
5	Update
6-15	Unassigned

Gibt OTHER für Codes zurück, die vom System nicht erkannt werden; der `opcodeNum` Eigenschaft gibt den numerischen Codewert an.

`opcodeNum:` ***Zahl***

Die numerische Darstellung des LLMNR-Operationscodes gemäß den IANA-LLMNR-Parametern.

`qname:` ***Schnur***

Der abgefragte Hostname.

`qtype:` ***Schnur***

Der Name des LLMNR-Anforderungsdatensatztyps gemäß den IANA-LLMNR-Parametern.

Gibt OTHER für Typen zurück, die vom System nicht erkannt werden; der `qtypeNum` property gibt den numerischen Typwert an.

`qtypeNum:` ***Zahl***

Die numerische Darstellung des LLMNR-Anforderungsdatensatztyps gemäß den IANA-LLMNR-Parametern.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `LLMNR.commitRecord()` entweder auf einem `LLMNR_REQUEST` oder `LLMNR_RESPONSE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

LLMNR_REQUEST	LLMNR_RESPONSE
clientIsExternal	answer
opcode	clientIsExternal
qname	error
qtype	opcode
receiverIsExternal	qname
reqBytes	qtype
reqL2Bytes	receiverIsExternal
reqPkts	rspBytes
senderIsExternal	rspL2Bytes
serverIsExternal	rspPkts
	senderIsExternal
	serverIsExternal

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf `LLMNR_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Headern.

Zugriff nur auf `LLMNR_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

Zugriff nur auf `LLMNR_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf `LLMNR_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf `LLMNR_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortbytes auf Anwendungsebene.

Zugriff nur auf `LLMNR_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`Memcache`

Die Memcache Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `MEMCACHE_REQUEST` und `MEMCACHE_RESPONSE` Ereignisse.

Ereignisse

`MEMCACHE_REQUEST`

Läuft bei jeder Memcache-Anfrage, die vom Gerät verarbeitet wird.

`MEMCACHE_RESPONSE`

Läuft auf jeder Memcache-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `MEMCACHE_REQUEST` oder `MEMCACHE_RESPONSE` Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die für jedes Ereignis festgeschriebenen Standardeigenschaften finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`accessTime:` ***Zahl***

Die Zugriffszeit, ausgedrückt in Millisekunden. Nur verfügbar, wenn der erste angeforderte Schlüssel einen Treffer erzeugt hat.

Zugriff nur auf `MEMCACHE_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`error:` ***Schnur***

Die detaillierte Fehlermeldung, die vom ExtraHop-System aufgezeichnet wurde.

Zugriff nur auf `MEMCACHE_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`hits:` ***Reihe***

Ein Array von Objekten, das den Memcache-Schlüssel und die Schlüsselgröße enthält.

Zugriff nur auf `MEMCACHE_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`key:` ***Schnur | null***

Der Memcache-Schlüssel, für den dies ein Treffer war, falls verfügbar.

`size:` ***Zahl***

Die Größe des für den Schlüssel zurückgegebenen Werts, ausgedrückt in Byte.

`isBinaryProtocol:` ***Boolescher Wert***

Der Wert ist `true` wenn die Anfrage/Antwort der Binärversion des Memcache-Protokolls entspricht.

`isNoReply:` ***Boolescher Wert***

Der Wert ist `true` wenn die Anfrage das Schlüsselwort „noreply“ hat und daher niemals eine Antwort erhalten sollte (nur Textprotokoll).

Zugriff nur auf `MEMCACHE_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`isRspImplicit:` ***Boolescher Wert***

Der Wert ist `true` wenn die Antwort durch eine nachfolgende Antwort des Server impliziert wurde (nur Binärprotokoll).

Zugriff nur auf `MEMCACHE_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`method:` ***Schnur***

Die Memcache-Methode, wie sie im Abschnitt Metrics des ExtraHop-Systems aufgezeichnet ist.

`misses:` ***Reihe***

Ein Array von Objekten, das den Memcache-Schlüssel enthält.

Zugriff nur auf `MEMCACHE_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`key:` ***Schnur | null***

Der Memcache-Schlüssel, für den dies ein Fehlschlag war, falls verfügbar.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `Memcache.commitRecord()` entweder auf einem `MEMCACHE_REQUEST` oder `MEMCACHE_RESPONSE` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

MEMCACHE_REQUEST	MEMCACHE_RESPONSE
clientIsExternal	accessTime
clientZeroWnd	clientIsExternal
isBinaryProtocol	clientZeroWnd
isNoReply	error
method	hits
receiverIsExternal	isBinaryProtocol
reqBytes	isRspImplicit
reqL2Bytes	method
reqPkts	misses

MEMCACHE_REQUEST	MEMCACHE_RESPONSE
reqRTO	receiverIsExternal
reqSize	roundTripTime
senderIsExternal	rspBytes
serverIsExternal	rspL2Bytes
serverZeroWnd	rspPkts
vbucket	rspRTO
	senderIsExternal
	serverIsExternal
	serverZeroWnd
	statusCode
	vbucket

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

`reqKeys:` ***Reihe***

Ein Array, das die Memcache-Schlüsselzeichenfolgen enthält, die mit der Anfrage gesendet wurden.

Der Wert des `reqKeys` Die Eigenschaft ist dieselbe, wenn auf eine der folgenden Seiten zugegriffen wird `MEMCACHE_REQUEST` oder der `MEMCACHE_RESPONSE` Ereignis.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf `MEMCACHE_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqSize:` ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne Memcache-Header. Der Wert ist `NaN` für Anfragen ohne Payload, wie GET und DELETE.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf `MEMCACHE_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`statusCode:` ***Schnur***

Der Memcache-Statuscode. Für das Binärprotokoll stellen die ExtraHop-Systemmetriken der Methode andere Statuscodes voran als `NO_ERROR`, die StatusCode-Eigenschaft jedoch nicht. In den Beispielen finden Sie Code, der dem Verhalten der ExtraHop-Systemmetriken entspricht.

Zugriff nur auf `MEMCACHE_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`vbucket:` ***Zahl***

Der Memcache-Bucket, falls verfügbar (nur Binärprotokoll).

Beispiele für Trigger

- Beispiel: Memcache-Treffer und Fehlschläge aufzeichnen
- Beispiel: Memcache-Schlüssel analysieren

`Modbus`

Die `Modbus` Klasse ermöglicht den Zugriff auf Eigenschaften von `MODBUS_REQUEST` und `MODBUS_RESPONSE` Ereignisse. Modbus ist ein serielles Kommunikationsprotokoll, das Verbindungen zwischen mehreren Geräten im selben Netzwerk ermöglicht.

Ereignisse

`MODBUS_REQUEST`

Läuft bei jeder Anfrage, die von einem Modbus-Client gesendet wird. Ein Modbus-Client im ExtraHop-System ist das Modbus-Master-Gerät.

`MODBUS_RESPONSE`

Läuft auf jeder Antwort, die von einem Modbus-Server gesendet wird. Ein Modbus-Server im ExtraHop-System ist das Modbus-Slave-Gerät.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `MODBUS_RESPONSE` Ereignis. Commits aufzeichnen für `MODBUS_REQUEST` Ereignisse werden nicht unterstützt.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`error:` ***Schnur***

Die detaillierte Fehlermeldung, die vom ExtraHop-System aufgezeichnet wurde.

Zugriff nur auf `MODBUS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`functionId:` ***Zahl***

Der Modbus-Funktionscode, der in der Anfrage oder Antwort enthalten ist.

Funktions-ID	Name der Funktion
1	Read Coil
2	Read Discrete Inputs
3	Read Holding Registers
4	Read Input Registers
5	Write Single Coil
6	Write Single Holding Register
15	Write Multiple Coils
16	Write Multiple Holding Registers

`functionName:` ***Schnur***

Der Name des Modbus-Funktionscodes, der in der Anfrage oder Antwort enthalten ist.

`isReqAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wurde, bevor die Anfrage abgeschlossen war.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wurde, bevor die Antwort abgeschlossen war.

Zugriff nur auf `MODBUS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`payload:` ***Puffer***

Die Puffer Objekt, das den Hauptteil der Anfrage oder Antwort enthält.

`payloadOffset:` ***Zahl***

Der Datei-Offset, ausgedrückt in Byte, innerhalb der `resource` Eigentum. Die Payload-Eigenschaft wird von der `resource` Eigentum am Offset.

`processingTime:` ***Zahl***

Die Verarbeitungszeit des Modbus-Servers, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `MODBUS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `Modbus.commitRecord` auf einem `MODBUS_RESPONSE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`
- `error`
- `functionId`
- `functionName`
- `protocolId`
- `reqL2Bytes`
- `rspL2Bytes`
- `receiverIsExternal`
- `reqPkts`
- `rspPkts`
- `reqBytes`
- `rspBytes`
- `reqRTO`
- `rspRTO`

- roundTripTime
- clientZeroWnd
- senderIsExternal
- serverIsExternal
- serverZeroWnd
- statusCode
- txId
- unitId

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqBytes: ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqL2Bytes: ***Zahl***

Die Anzahl der L2-Anforderungsbytes, einschließlich L2 Kopfzeilen.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqPkts: ***Zahl***

Die Anzahl der Pakete in der Anfrage.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqRTO: ***Zahl***

Die Anzahl der Timeouts bei der erneuten Übertragung (RTOs) in der Anfrage.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqSize: ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne Modbus-Header.

reqTransferTime: ***Zahl***

Die Übertragungszeit der Anfrage, ausgedrückt in Millisekunden. Wenn die Anfrage in einem einzigen Paket enthalten ist, ist die Übertragungszeit Null. Wenn sich die Anfrage über mehrere Pakete erstreckt, ist der Wert die Zeitspanne zwischen der Erkennung des ersten Anforderungspakets und der Erkennung des letzten Paket durch das ExtraHop-System. Ein hoher Wert kann auf eine große Anfrage oder eine Netzwerkverzögerung hinweisen. Der Wert ist NaN wenn es keine gültige Messung gibt oder wenn der Zeitpunkt ungültig ist.

reqZeroWnd: ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

roundTripTime: ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist NaN wenn es keine RTT-Proben gibt.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspBytes: ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

rspL2Bytes: ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspPkts: ***Zahl***

Die Anzahl der Pakete in der Antwort.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspRTO: ***Zahl***

Die Anzahl der Timeouts bei der erneuten Übertragung (RTOs) in der Antwort.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspSize: ***Zahl***

Die Anzahl der L7-Antwortbytes, ohne Modbus-Protokoll-Header.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspTransferTime: ***Zahl***

Die Übertragungszeit der Antwort, ausgedrückt in Millisekunden. Wenn die Antwort in einem einzigen Paket enthalten ist, ist die Übertragungszeit Null. Wenn sich die Antwort über mehrere Pakete erstreckt, ist der Wert die Zeitspanne zwischen der Erkennung des ersten Antwortpakets und der Erkennung des letzten Paket durch das ExtraHop-System. Ein hoher Wert kann auf eine starke Reaktion oder eine Netzwerkverzögerung hinweisen. Der Wert ist NaN wenn es keine gültige Messung gibt oder wenn der Zeitpunkt ungültig ist.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspZeroWnd: ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

statusCode: ***Zahl***

Der numerische Statuscode der Antwort.

Statuscodenummer	Beschreibung des Status
1	Illegal Function
2	Illegal Data Address
3	Illegal Data Value
4	Slave Device Failure
5	Acknowledge
6	Slave Device Busy
7	Negative Acknowledge
8	Memory Parity Error
10	Gateway Path Unavailable
11	Gateway Target Device Failed to Respond

Zugriff nur auf MODBUS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

txId: ***Zahl***

Die Transaktions-ID der Anfrage oder Antwort.

unitId: ***Zahl***

Die Einheitenkennung des Modbus-Servers, der auf den Modbus-Client reagiert.

MongoDB

Die MongoDB Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen MONGODB_REQUEST und MONGODB_RESPONSE Ereignisse.

Ereignisse

`MONGODB_REQUEST`

Läuft bei jeder MongoDB-Anfrage, die vom Gerät verarbeitet wird.

`MONGODB_RESPONSE`

Läuft auf jeder MongoDB-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `MONGODB_REQUEST` oder `MONGODB_RESPONSE` Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die Standardeigenschaften, die für jedes Ereignis festgeschrieben wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`collection:` ***Schnur***

Der Name der Datenbanksammlung, die in der aktuellen Anfrage angegeben wurde.

`database:` ***Schnur***

Die MongoDB-Datenbankinstanz. In einigen Fällen, z. B. wenn Anmeldeereignisse verschlüsselt sind, ist der Datenbankname nicht verfügbar.

`error:` ***Schnur***

Die detaillierte Fehlermeldung, die vom ExtraHop-System aufgezeichnet wurde.

Zugriff nur auf `MONGODB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isReqAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die MongoDB-Anfrage abgeschlossen war.

`isReqTruncated:` ***Boolescher Wert***

Der Wert ist `true` wenn die Größe der Anforderungsdokumente größer als die maximale Größe des Payload-Dokuments ist.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die MongoDB-Antwort abgeschlossen war.

Zugriff nur auf `MONGODB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`method:` ***Schnur***

Die MongoDB-Datenbankmethode (erscheint unter **Methoden** in der Benutzeroberfläche).

`opcode:` ***Schnur***

Der MongoDB-Betriebscode im Wire-Protokoll, der sich von der verwendeten MongoDB-Methode unterscheiden kann.

`processingTime:` ***Zahl***

Die Zeit für die Bearbeitung der Anfrage, ausgedrückt in Millisekunden (entspricht `rspTimeToFirstByte - reqTimeToLastByte`). Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `MONGODB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `MongoDB.commitRecord()` entweder auf einem `MONGODB_REQUEST` oder `MONGODB_RESPONSE` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

MONGODB_REQUEST	MONGODB_RESPONSE
clientIsExternal	clientIsExternal
clientZeroWnd	clientZeroWnd
collection	collection
database	database
isReqAborted	error
isReqTruncated	isRspAborted
method	method
opcode	opcode
receiverIsExternal	processingTime
reqBytes	receiverIsExternal
reqL2Bytes	roundTripTime
reqPkts	rspBytes
reqRTO	rspL2Bytes
reqSize	rspPkts
reqTimeToLastByte	rspRTO
senderIsExternal	rspSize
serverIsExternal	rspTimeToFirstByte
serverZeroWnd	rspTimeToLastByte
user	senderIsExternal
	serverIsExternal
	serverZeroWnd
	user

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

`reqSize:` ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne MongoDB-Header.

`reqTimeToLastByte:` ***Zahl***

Die Zeit vom ersten Byte der Anforderung bis zum letzten Byte der Anforderung, ausgedrückt in Millisekunden.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`request:` ***Reihe***

Ein Array von JS-Objekten, die aus MongoDB-Anforderungs-Payload-Dokumenten analysiert wurden. Die Gesamtgröße des Dokuments ist auf 4K begrenzt.

Wenn BSON-Dokumente gekürzt werden, `isReqTruncated` Flagge ist gesetzt. Verkürzte Werte werden wie folgt dargestellt:

- Primitive Zeichenkettenwerte wie Code, Code mit Gültigkeitsbereich und Binärdaten werden teilweise extrahiert.
- Objekte und Arrays werden teilweise extrahiert.
- Alle anderen primitiven Werte wie Numbers, Dates, RegExp usw. werden ersetzt durch `null`.

Wenn keine Dokumente in der Anfrage enthalten sind, wird ein leeres Array zurückgegeben.

Der Wert des `request` Die Eigenschaft ist dieselbe, wenn auf eine der folgenden Seiten zugegriffen wird `MONGODB_REQUEST` oder der `MONGODB_RESPONSE` Ereignis.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes, ohne MongoDB-Header.

Zugriff nur auf `MONGODB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToFirstByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum ersten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `MONGODB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToLastByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum letzten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `MONGODB_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`user:` ***Schnur***

Der Benutzername, falls verfügbar. In einigen Fällen, z. B. wenn Anmeldeereignisse verschlüsselt sind, ist der Benutzername nicht verfügbar.

MSMQ

Die MSMQ Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `MSMQ_MESSAGE` Ereignisse.

Ereignisse

`MSMQ_MESSAGE`

Wird für jede vom Gerät verarbeitete MSMQ-Benutzernachricht ausgeführt.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `MSMQ_MESSAGE` Ereignis.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`adminQueue:` ***Schnur***

Der Name der Administrationswarteschlange der Nachricht.

`correlationId:` ***Puffer***

Die von der Anwendung generierte Korrelations-ID der Nachricht.

`dstQueueMgr:` ***Schnur***

Der Ziel-Nachrichtenbroker der Nachricht.

`isEncrypted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Nutzdaten verschlüsselt sind.

`label:` ***Schnur***

Die Bezeichnung oder Beschreibung der Nachricht.

`msgClass:` ***Schnur***

Die Nachrichtenklasse der Nachricht. Die folgenden Werte sind gültig:

- `MQMSG_CLASS_NORMAL`
- `MQMSG_CLASS_ACK_REACH_QUEUE`
- `MQMSG_CLASS_NACK_ACCESS_DENIED`
- `MQMSG_CLASS_NACK_BAD_DST_Q`
- `MQMSG_CLASS_NACK_BAD_ENCRYPTION`
- `MQMSG_CLASS_NACK_BAD_SIGNATURE`
- `MQMSG_CLASS_NACK_COULD_NOT_ENCRYPT`
- `MQMSG_CLASS_NACK_HOP_COUNT_EXCEEDED`
- `MQMSG_CLASS_NACK_NOT_TRANSACTIONAL_MSG`
- `MQMSG_CLASS_NACK_NOT_TRANSACTIONAL_Q`
- `MQMSG_CLASS_NACK_PURGED`
- `MQMSG_CLASS_NACK_Q_EXCEEDED_QUOTA`
- `MQMSG_CLASS_NACK_REACH_QUEUE_TIMEOUT`

- MQMSG_CLASS_NACK_SOURCE_COMPUTER_GUID_CHANGED
- MQMSG_CLASS_NACK_UNSUPPORTED_CRYPTO_PROVIDER
- MQMSG_CLASS_ACK_RECEIVE
- MQMSG_CLASS_NACK_Q_DELETED
- MQMSG_CLASS_NACK_Q_PURGED
- MQMSG_CLASS_NACK_RECEIVE_TIMEOUT
- MQMSG_CLASS_NACK_RECEIVE_TIMEOUT_AT_SENDER
- MQMSG_CLASS_REPORT

`msgId:` ***Zahl***

Die MSMQ-Meldungs-ID der Nachricht.

`payload:` ***Puffer***

Der Hauptteil der MSMQ-Nachricht.

`priority:` ***Zahl***

Die Priorität der Nachricht. Dies kann eine Zahl zwischen 0 und 7 sein.

`queue:` ***Schnur***

Der Name der Zielwarteschlange der Nachricht.

`receiverBytes:` ***Zahl***

Die Anzahl der L4 Empfänger-Bytes.

`receiverL2Bytes:` ***Zahl***

Die Anzahl der L2 Empfänger-Bytes.

`receiverPkts:` ***Zahl***

Die Anzahl der Empfängerpakete.

`receiverRTO:` ***Zahl***

Die Anzahl der Timeouts bei der erneuten Übertragung (RTOs) vom Empfänger.

`receiverZeroWnd:` ***Zahl***

Die Anzahl der vom Empfänger gesendeten Nullfenster.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `MSMQ.commitRecord()` auf einem `MSMQ_MESSAGE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- adminQueue
- clientIsExternal
- dstQueueMgr
- isEncrypted
- label
- msgClass
- msgId
- priority
- queue
- receiverBytes
- receiverIsExternal
- receiverL2Bytes
- receiverPkts
- receiverRTO
- receiverZeroWnd
- responseQueue
- roundTripTime

- `senderBytes`
- `senderIsExternal`
- `serverIsExternal`
- `senderL2Bytes`
- `senderPkts`
- `senderRTO`
- `serverZeroWnd`
- `srcQueueMgr`

`responseQueue:` ***Schnur***

Der Name der Antwortwarteschlange der Nachricht.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`senderBytes:` ***Zahl***

Die Nummer des Absenders L4 Byte.

`senderL2Bytes:` ***Zahl***

Die Nummer des Absenders L2 Byte.

`senderPkts:` ***Zahl***

Die Anzahl der Absenderpakete.

`senderRTO:` ***Zahl***

Die Anzahl der Timeouts bei der erneuten Übertragung (RTOs) vom Absender.

`senderZeroWnd:` ***Zahl***

Die Anzahl der vom Absender gesendeten Nullfenster.

`srcQueueMgr:` ***Schnur***

Der Quellnachrichtenbroker der Nachricht.

`NetFlow`

Die `NetFlow` Klassenobjekt ermöglicht es Ihnen, Metriken zu speichern und auf Eigenschaften zuzugreifen `NETFLOW_RECORD` Ereignisse.

Hinweis: Die NetFlow-Klasse ist nur auf Reveal (x) 360- und ExtraHop Performance-Systemen verfügbar.

Das ExtraHop-System kann für das NetFlow-Modul lizenziert werden, das die folgenden Flow-Typen unterstützt:

- NetFlow Version 5 (Cisco)
- NetFlow Version 9 (Cisco)
- IPFIX (offener Standard basierend auf RFC 5101)

Ereignisse

`NETFLOW_RECORD`

Wird nach Erhalt eines Flow-Datensatzes von einem Flussnetz ausgeführt.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `NETFLOW_RECORD` Ereignis.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

`findField(field:` ***Zahl*** `, enterpriseId:` ***Zahl*** `):` ***Schnur*** | ***Zahl*** | ***IP-Adresse*** | ***Puffer*** | ***Boolescher Wert***

Durchsucht den NetFlow-Datensatz und gibt das angegebene Feld zurück. Gibt einen Nullwert zurück, wenn das Feld nicht im Datensatz enthalten ist. Wenn das optionale `enterpriseId` Argument ist enthalten, das angegebene Feld wird nur zurückgegeben, wenn die Enterprise-ID übereinstimmt, andernfalls gibt die Methode einen Nullwert zurück.

`hasField(field:` ***Zahl*** `):` ***Boolescher Wert***

Ermittelt, ob das angegebene Feld im NetFlow-Datensatz enthalten ist.

Eigenschaften

`age:` ***Zahl***

Die verstrichene Zeit, ausgedrückt in Sekunden, zwischen `first` und `last` Eigenschaftswerte, die im NetFlow-Datensatz gemeldet werden.

`deltaBytes:` ***Zahl***

Die Anzahl der L3 Bytes im Fluss seit dem letzten `NETFLOW_RECORD` Ereignis.

`deltaPkts:` ***Zahl***

Die Anzahl der Pakete im Fluss seit dem letzten `NETFLOW_RECORD` Ereignis.

`dscp:` ***Zahl***

Die Zahl, die den letzten DSCP-Wert (Differentiated Services Code Point) des Flow-Pakets darstellt.

`dscpName:` ***Schnur***

Der Name, der dem DSCP-Wert des Flow-Pakets zugeordnet ist. In der folgenden Tabelle sind bekannte DSCP-Namen aufgeführt:

Zahl	Name
8	CS1
10	AF11
12	AF12
14	AF13
16	CS2
18	AF21
20	AF22
22	AF23
24	CS3
26	AF31
28	AF32
30	AF33
32	CS4
34	AF41
36	AF 42
38	AF43

Zahl	Name
40	CS5
44	VA
46	EF
48	CS6
56	CS7

Ausgangsschnittstelle: ***Flow-Schnittstelle***

Die FlowInterface Objekt, das das Ausgabegerät identifiziert.

Felder: ***Reihe***

Eine Reihe von Objekten, die Informationsfelder enthalten, die in den Flow-Paketen gefunden wurden. Jedes Objekt kann die folgenden Eigenschaften enthalten:

Feld-ID: ***Zahl***

Die ID-Nummer, die den Feldtyp darstellt.

Unternehmens-ID: ***Zahl***

Die ID-Nummer, die unternehmensspezifische Informationen darstellt.

zuerst: ***Zahl***

Die Zeit, ausgedrückt in Millisekunden, seit der Epoche des ersten Pakets im Datenfluss vergangen ist.

Format: ***Schnur***

Das Format des NetFlow-Datensatzes. Gültige Werte sind `NetFlow v5`, `NetFlow v9`, und `IPFIX`.

Eingangsschnittstelle: ***Flow-Schnittstelle***

Die FlowInterface Objekt, das das Eingabegerät identifiziert.

IP-Priorität: ***Zahl***

Der Wert des IP-Prioritätsfeldes, das dem DSCP des Flow-Pakets zugeordnet ist.

ipproto: ***Schnur***

Das mit dem Fluss verknüpfte IP-Protokoll, z. B. TCP oder UDP.

zuletzt: ***Zahl***

Die Zeit, ausgedrückt in Millisekunden, seit der Epoche des letzten Pakets im Datenfluss vergangen ist.

netzwerk: ***Flow-Netzwerk***

Ein Objekt, das die identifiziert FlowNetwork und enthält die folgenden Eigenschaften:

ID: ***Schnur***

Die Kennung des FlowNetwork.

ipaddr: ***IP-Adresse***

Die IP-Adresse des FlowNetwork.

Nächster Hop: ***IP-Adresse***

Die IP-Adresse des Next-Hop-Routers.

Beobachtungsbereich: ***Zahl***

Die ID der Beobachtungsdomäne für die Vorlage.

empfänger: ***Objekt***

Ein Objekt, das den Empfänger identifiziert und die folgenden Eigenschaften enthält:

als: ***Zahl***

Die autonome Systemnummer (ASN) des Zielgeräts.

ipaddr: ***IP-Adresse***

Die IP-Adresse des Zielgeräts.

Länge des Präfixes: ***Zahl***

Die Anzahl der Bits im Präfix der Zieladresse.

Hafen: ***Zahl***

Die TCP- oder UDP-Portnummer des Zielgeräts.

Datensatz: ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `NetFlow.commitRecord()` auf einem `NETFLOW_RECORD` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- Alter
- Kunde ist extern
- DSCP-Name
- DeltaByte
- Delta PKT
- Ausgangsschnittstelle
- erstmalig
- Format
- Ingress-Schnittstelle
- zuletzt
- Netzwerk
- Netzwerk-Addr
- Nächster Hop
- Proto
- Adresse des Empfängers
- Empfänger ASN
- Empfänger ist extern
- Empfängeranschluss
- Länge des Empfänger-Präfixes
- Adresse des Absenders
- Absender ASN
- Absender ist extern
- Server ist extern
- Absender-Port
- Länge des Absenderpräfixes
- TCP-Flagname
- TCP-Flaggen

Absender: ***Objekt***

Ein Objekt, das den Absender identifiziert und die folgenden Eigenschaften enthält:

als: ***Zahl***

Die autonome Systemnummer (ASN) des Quellgeräts.

ipaddr: ***IP-Adresse***

Die IP-Adresse des Quellgeräts.

Länge des Präfixes: ***Zahl***

Die Anzahl der Bits im Präfix der Quelladresse.

Hafen: ***Zahl***

Die TCP- oder UDP-Portnummer des Quellgeräts.

TCP-Flagnamen: ***Reihe***

Ein Zeichenkettenarray von TCP-Flagnamen, wie SYN oder ACK, die in den Flow-Paketen gefunden wurden.

TCP-Flaggen: ***Zahl***

Das bitweise ODER aller TCP-Flags, die für den Flow gesetzt wurden.

Vorlagen-ID: ***Zahl***

Die ID der Vorlage, auf die der Datensatz verweist. Vorlagen-IDs gelten nur für IPFIX- und NetFlow v9-Datensätze.

tos: ***Zahl***

Die im IP-Header definierte Nummer (Type of Service, ToS).

NFS

Die NFS Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen NFS_REQUEST und NFS_RESPONSE Ereignisse.

Ereignisse

NFS_REQUEST

Läuft bei jeder NFS-Anfrage, die vom Gerät verarbeitet wird.

NFS_RESPONSE

Läuft auf jeder NFS-Antwort, die vom Gerät verarbeitet wird.

Hinweis: Die NFS_RESPONSE Die Ereignis läuft nach jedem NFS_REQUEST Ereignis, auch wenn die entsprechende Reaktion vom ExtraHop-System nie beobachtet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem NFS_RESPONSE Ereignis. Commits aufzeichnen für NFS_REQUEST Ereignisse werden nicht unterstützt.

Die Standardeigenschaften, die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`accessTime:` ***Zahl***

Die Zeit, die der Server für den Zugriff auf eine Datei auf der Festplatte benötigt, ausgedrückt in Millisekunden. Für NFS ist dies die Zeit von jedem READ- und WRITE-Befehl in einem NFS-Flow, der nicht über die Pipeline geleitet wird, bis die Payload, die die Antwort enthält, vom ExtraHop-System aufgezeichnet wird. Der Wert ist `NaN` bei fehlerhaften und abgebrochenen Antworten oder wenn der Zeitpunkt ungültig oder nicht zutreffend ist.

Zugriff nur auf NFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`authMethod:` ***Schnur***

Die Methode zur Authentifizierung von Benutzern.

`error:` ***Schnur***

Die detaillierte Fehlermeldung, die vom ExtraHop-System aufgezeichnet wurde.

Zugriff nur auf NFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`fileHandle:` ***Puffer***

Das vom Server bei LOOKUP-, CREATE-, SYMLINK-, MKNOD-, LINK- oder READDIRPLUS-Operationen zurückgegebene Datei-Handle.

`isCommandFileInfo:` ***Boolescher Wert***

Der Wert ist `true` für Dateiinformationsbefehle.

`isCommandRead:` ***Boolescher Wert***

Der Wert ist `true` für READ-Befehle.

`isCommandWrite:` ***Boolescher Wert***

Der Wert ist `true` für WRITE-Befehle.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist wahr, wenn die Verbindung geschlossen wird, bevor die Antwort abgeschlossen war.

Zugriff nur auf `NFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`method:` ***Schnur***

Die NFS-Methode. Gültige Methoden sind unter der NFS-Metrik im ExtraHop-System aufgeführt.

`offset:` ***Zahl***

Der Dateioffset, der den NFS READ- und WRITE-Befehlen zugeordnet ist.

Zugriff nur auf `NFS_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `NFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `NFS.commitRecord()` auf einem `NFS_RESPONSE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `accessTime`
- `authMethod`
- `clientIsExternal`
- `clientZeroWnd`
- `error`
- `isCommandFileInfo`
- `isCommandRead`
- `isCommandWrite`
- `isRspAborted`
- `method`
- `offset`
- `processingTime`
- `receiverIsExternal`
- `renameDirChanged`
- `reqSize`
- `reqXfer`
- `resource`
- `rspSize`
- `rspXfer`
- `senderIsExternal`
- `serverIsExternal`

- serverZeroWnd
- statusCode
- txID
- user
- version

Greifen Sie nur auf das Datensatzobjekt zu unter NFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

renameDirChanged: ***Boolescher Wert***

Der Wert ist true wenn eine Anfrage zur Umbenennung einer Ressource eine Verzeichnisverschiebung beinhaltet.

Zugriff nur auf NFS_REQUEST Ereignisse; andernfalls tritt ein Fehler auf.

reqBytes: ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf NFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqL2Bytes: ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

Zugriff nur auf NFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqPkts: ***Zahl***

Die Anzahl der Anforderungspakete.

Zugriff nur auf NFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqRTO: ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf NFS_REQUEST Ereignisse; andernfalls tritt ein Fehler auf.

reqSize: ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne NFS-Header.

reqTransferTime: ***Zahl***

Die Übertragungszeit der Anfrage, ausgedrückt in Millisekunden. Wenn die Anfrage in einem einzigen Paket enthalten ist, ist die Übertragungszeit Null. Wenn sich die Anfrage über mehrere Pakete erstreckt, ist der Wert die Zeitspanne zwischen der Erkennung des ersten NFS-Anforderungspakets und der Erkennung des letzten Paket durch das ExtraHop-System. Ein hoher Wert kann auf eine große NFS-Anfrage oder eine Netzwerkverzögerung hinweisen. Der Wert ist NaN wenn es keine gültige Messung gibt oder wenn der Zeitpunkt ungültig ist.

Zugriff nur auf NFS_REQUEST Ereignisse; andernfalls tritt ein Fehler auf.

reqZeroWnd: ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

resource: ***Schnur***

Der Pfad und der Dateiname, miteinander verkettet.

roundTripTime: ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist NaN wenn es keine RTT-Proben gibt.

Zugriff nur auf NFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspBytes: ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen L4-Protokoll-Overhead, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf NFS_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf `NFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

Zugriff nur auf `NFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf `NFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes ohne NFS-Header.

Zugriff nur auf `NFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTransferTime:` ***Zahl***

Die Antwortübertragungszeit, ausgedrückt in Millisekunden. Wenn die Antwort in einem einzigen Paket enthalten ist, ist die Übertragungszeit Null. Wenn sich die Antwort über mehrere Pakete erstreckt, ist der Wert die Zeitspanne zwischen der Erkennung des ersten NFS-Antwortpakets und der Erkennung des letzten Paket durch das ExtraHop-System. Ein hoher Wert kann auf eine große NFS-Antwort oder eine Netzwerkverzögerung hinweisen. Der Wert ist `NaN` wenn es keine gültige Messung gibt oder wenn der Zeitpunkt ungültig ist.

Zugriff nur auf `NFS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`statusCode:` ***Schnur***

Der NFS-Statuscode der Anfrage oder Antwort.

`symlink:` ***Puffer | null***

Das in einer NFS-SYMLINK-Anfrage angegebene Argument.

Der Wert ist Null, wenn auf diese Eigenschaft bei einem anderen Ereignis als NFS_REQUEST zugegriffen wird oder wenn `NFS.method` ist nicht `SYMLINK`.

`txId:` ***Zahl***

Die Transaktions-ID.

`user:` ***Schnur***

Die ID des Linux-Benutzers, formatiert als `uid:xxxx@ip_address`.

`verifierMethod:` ***Schnur***

Die Methode zur Überprüfung des Absenders der Anfrage.

`version:` ***Zahl***

Die NFS-Version.

`NTLM`

Die `NTLM` Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `NTLM_MESSAGE` Ereignisse.

Ereignisse

`NTLM_MESSAGE`

Läuft auf jeder NTLM-Nachricht, die vom Gerät verarbeitet wird.

Methoden

`commitRecord()`: ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `NTLM_MESSAGE` Ereignis.

Die Standardeigenschaften , die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`containsMIC`: ***Boolescher Wert***

Der Wert ist wahr, wenn die Nachricht einen Message Integrity Code (MIC) enthält , der sicherstellt, dass die Nachricht nicht manipuliert wurde.

`challenge`: ***Schnur***

Die hexadezimalkodierte Challenge-Hash-Zeichenfolge.

`domain`: ***Schnur***

Der in der Challenge-Hash-Berechnung enthaltene Client-Domainname.

`flags`: ***Zahl***

Das bitweise ODER der NTLM-Aushandlungsflags. Weitere Informationen finden Sie in der NTLM-Dokumentation auf der Microsoft-Website.

`msgType`: ***Schnur***

Der Typ der NTLM-Nachricht. Die folgenden Nachrichtentypen sind gültig:

- `NTLM_AUTH`
- `NTLM_CHALLENGE`
- `NTLM_NEGOTIATE`

`ntlm2RspAVPairs`: ***Reihe***

Ein Array von Objekten, die NTLM-Attribut-Wert-Paare enthalten. Weitere Informationen finden Sie in der NTLM-Dokumentation auf der Microsoft-Website.

`record`: ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `NTLM.commitRecord()` auf einem `NTLM_MESSAGE` Ereignis.

Das Standard-Datensatzobjekt kann die folgenden Eigenschaften enthalten:

- `challenge`
- `clientIsExternal`
- `domain`
- `flags`
- `l7proto`
- `msgType`
- `proto`
- `receiverAddr`
- `receiverIsExternal`
- `receiverPort`
- `senderAddr`
- `senderIsExternal`
- `senderPort`
- `serverIsExternal`
- `user`
- `windowsVersion`

- `workstation`

`rspVersion:` ***Schnur***

Die Version von NTLM, die in der NTLM_AUTH-Antwort implementiert ist. Der Wert ist `null` für Nachrichten ohne Authentifizierung. Die folgenden Versionen sind gültig:

- `LM`
- `NTLMv1`
- `NTLMv2`

`user:` ***Schnur***

Der Client-Benutzername, der in der Challenge-Hash-Berechnung enthalten ist.

`windowsVersion:` ***Schnur***

Die Version von Windows, die auf dem Client ausgeführt wird, ist in der Challenge-Hash-Berechnung enthalten.

`workstation:` ***Schnur***

Der Name der Client-Workstation, der in der Challenge-Hash-Berechnung enthalten ist.

POP3

Die POP3 Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen POP3_REQUEST und POP3_RESPONSE Ereignisse.

Ereignisse

`POP3_REQUEST`

Wird bei jeder POP3-Anfrage ausgeführt, die vom Gerät verarbeitet wird.

`POP3_RESPONSE`

Läuft bei jeder POP3-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem POP3_RESPONSE Ereignis. Commits aufzeichnen für POP3_REQUEST Ereignisse werden nicht unterstützt.

Die Standardeigenschaften, die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`dataSize:` ***Zahl***

Die Größe der Nachricht, ausgedrückt in Byte.

Zugriff nur auf POP3_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`error:` ***Schnur***

Die detaillierte Fehlermeldung, die vom ExtraHop-System aufgezeichnet wurde.

Zugriff nur auf POP3_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`isEncrypted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Transaktion über einen sicheren POP3-Server Server.

`isReqAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die POP3-Anfrage abgeschlossen war.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die POP3-Antwort abgeschlossen war.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`method:` ***Schnur***

Die POP3-Methode wie RETR oder DELE.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei fehlerhaften und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`recipientList:` ***Reihe***

Ein Array, das eine Liste von Empfängeradressen enthält.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `POP3.commitRecord()` auf einem `POP3_RESPONSE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`
- `clientZeroWnd`
- `dataSize`
- `error`
- `isEncrypted`
- `isReqAborted`
- `isRspAborted`
- `method`
- `processingTime`
- `receiverIsExternal`
- `recipientList`
- `reqSize`
- `reqTimeToLastByte`
- `rspSize`
- `rspTimeToFirstByte`
- `rspTimeToLastByte`
- `sender`
- `senderIsExternal`
- `serverIsExternal`
- `serverZeroWnd`
- `statusCode`

Greifen Sie nur auf das Datensatzobjekt zu `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

`reqSize:` ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne POP3-Header.

`reqTimeToLastByte:` ***Zahl***

Die Zeit vom ersten Byte der Anforderung bis zum letzten Byte der Anforderung, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei abgelaufenen Anfragen und Antworten oder wenn der Zeitpunkt ungültig ist.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***

Die mittlere TCP-Roundtrip-Zeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes, ohne POP3-Header.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToFirstByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum ersten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToLastByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum letzten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`sender:` ***Schnur***

Die Adresse des Absenders der Nachricht.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`status:` ***Schnur***

Die POP3-Statusmeldung der Antwort, die sein kann `OK`, `ERR` oder `NULL`.

Zugriff nur auf `POP3_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`QUIC`

Die `QUIC` Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `QUIC_OPEN` und `QUIC_CLOSE` Ereignisse.

Ereignisse

`QUIC_CLOSE`

Wird ausgeführt, wenn eine QUIC-Verbindung geschlossen ist.

`QUIC_OPEN`

Wird ausgeführt, wenn eine QUIC-Verbindung geöffnet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `QUIC_OPEN` oder `QUIC_CLOSE` Ereignis. Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`cyuFingerprint:` ***Schnur***

Der CYU-Fingerabdruck für die Verbindung. Der CYU-Fingerabdruck wird generiert, indem die Version und die im Client-Hello-Paket angegebenen Tags verkettet werden.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `QUIC.commitRecord()` entweder auf einem `QUIC_OPEN` oder `QUIC_CLOSE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientAddr`
- `clientIsExternal`
- `clientPort`
- `cyuFingerprint`
- `proto`
- `receiverIsExternal`
- `senderIsExternal`
- `serverAddr`
- `serverIsExternal`
- `serverPort`
- `sni`
- `version`

- `vlan`

`sni:` ***Schnur***

Die Server Name Indication (SNI), die den Namen des Servers identifiziert, zu dem der Client eine Verbindung herstellt.

`tags:` ***Reihe von Objekten***

Ein Array von Objekten, die die im Client-Hello-Paket gesetzten Tags angeben. Jedes Objekt hat die folgenden Eigenschaften:

`tag:` ***Schnur***

Der Name des Tags.

`value:` ***Puffer***

Der Wert, auf den das Tag gesetzt ist.

`version:` ***Schnur***

Die Version des QUIC-Protokolls.

RDP

RDP (Remote Desktop Protocol) ist ein von Microsoft entwickeltes proprietäres Protokoll, mit dem ein Windows-Computer eine Verbindung zu einem anderen Windows-Computer im selben Netzwerk oder über das Internet herstellen kann. Die `RDP` Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `RDP_OPEN`, `RDP_CLOSE`, oder `RDP_TICK` Ereignisse.

Ereignisse

`RDP_CLOSE`

Wird ausgeführt, wenn eine RDP-Verbindung geschlossen ist.

`RDP_OPEN`

Wird ausgeführt, wenn eine neue RDP-Verbindung geöffnet wird.

`RDP_TICK`

Wird regelmäßig ausgeführt, während der Benutzer mit der RDP-Anwendung interagiert.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `RDP_OPEN`, `RDP_CLOSE`, oder `RDP_TICK` Ereignis.

Das Ereignis bestimmt , welche Eigenschaften dem Record-Objekt zugewiesen werden. Die Standardeigenschaften , die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`clientBuild:` ***Schnur***

Die Buildnummer des RDP-Clients. Diese Eigenschaft ist nicht verfügbar, wenn die RDP-Verbindung verschlüsselt ist.

`clientName:` ***Schnur***

Der Name des Client-Computers. Diese Eigenschaft ist nicht verfügbar, wenn die RDP-Verbindung verschlüsselt ist.

`cookie:` ***Schnur***

Das vom RDP-Client gespeicherte Auto-Connect-Cookie.

`desktopHeight:` ***Zahl***

Die Höhe des Desktops, ausgedrückt in Pixeln. Diese Eigenschaft ist nicht verfügbar, wenn die RDP-Verbindung verschlüsselt ist.

`desktopWidth:` ***Zahl***

Die Breite des Desktops, ausgedrückt in Pixeln. Diese Eigenschaft ist nicht verfügbar, wenn die RDP-Verbindung verschlüsselt ist.

`encryptionProtocol:` ***Schnur***

Das Protokoll, mit dem die Transaktion verschlüsselt ist.

`error:` ***Schnur***

Die detaillierte Fehlermeldung, die vom ExtraHop-System aufgezeichnet wurde.

`isDecrypted:` ***Boolescher Wert***

Der Wert ist wahr, wenn das ExtraHop-System die Transaktion sicher entschlüsselt und analysiert hat. Die Analyse des entschlüsselten Datenverkehrs kann komplexe Bedrohungen aufdecken, die sich im verschlüsselten Verkehr verstecken.

`isEncrypted:` ***Boolescher Wert***

Der Wert ist `true` wenn die RDP-Verbindung verschlüsselt ist.

`isError:` ***Boolescher Wert***

Der Wert ist `true` wenn bei dem Ereignis ein Fehler aufgetreten ist.

`keyboardLayout:` ***Schnur***

Das Tastaturlayout, das die Anordnung der Tasten und die Eingabesprache angibt. Diese Eigenschaft ist nicht verfügbar, wenn die RDP-Verbindung verschlüsselt ist.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `RDP.commitRecord()` entweder auf einem `RDP_OPEN`, `RDP_CLOSE`, oder `RDP_TICK` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

`RDP_OPEN` **und** `RDP_CLOSE`	`RDP_TICK`
`clientBuild`	`clientBuild`
`clientIsExternal`	`clientBytes`
`clientName`	`clientIsExternal`
`cookie`	`clientL2Bytes`
`desktopHeight`	`clientName`
`desktopWidth`	`clientPkts`
`error`	`clientRTO`
`isEncrypted`	`clientZeroWnd`
`keyboardLayout`	`cookie`
`receiverIsExternal`	`desktopHeight`
`requestedColorDepth`	`desktopWidth`
`requestedProtocols`	`error`
`selectedProtocol`	`isEncrypted`

RDP_OPEN und RDP_CLOSE	RDP_TICK
senderIsExternal	keyboardLayout
serverIsExternal	receiverIsExternal
	requestedColorDepth
	requestedProtocols
	roundTripTime
	selectedProtocol
	senderIsExternal
	serverBytes
	serverIsExternal
	serverL2Bytes
	serverPkts
	serverRTO
	serverZeroWnd

`requestedColorDepth:` ***Schnur***

Die vom RDP-Client angeforderte Farbtiefe. Diese Eigenschaft ist nicht verfügbar, wenn die RDP-Verbindung verschlüsselt ist.

`requestedProtocols:` ***Reihe von Zeichenketten***

Die Liste der unterstützten Sicherheitsprotokolle.

`reqBytes:` ***Zahl***

Die Anzahl der L4 Bytes in der Anfrage.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Bytes in der Anfrage.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`reqPkts:` ***Zahl***

Die Anzahl der Pakete in der Anfrage.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`reqRTO:` ***Zahl***

Die Anzahl der Timeouts bei der erneuten Übertragung (RTOs) in der Anfrage.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT) für die Dauer des Ereignis, ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen L4-Protokoll-Overhead, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Pakete in der Antwort.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rspRTO:` ***Zahl***

Die Anzahl der Timeouts bei der erneuten Übertragung (RTOs) in der Antwort.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

Zugriff nur auf `RDP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`selectedProtocol:` ***Schnur***

Das ausgewählte Sicherheitsprotokoll.

`Redis`

Remote Dictionary Server (Redis) ist ein Open-Source-In-Memory-Datenstrukturserver. Die `Redis` Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `REDIS_REQUEST` und `REDIS_RESPONSE` Ereignisse.

Ereignisse

`REDIS_REQUEST`

Läuft bei jeder Redis-Anfrage, die vom Gerät verarbeitet wird.

`REDIS_RESPONSE`

Läuft auf jeder Redis-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `REDIS_REQUEST` oder `REDIS_RESPONSE` Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die für jedes Ereignis festgeschriebenen Standardeigenschaften finden Sie unter `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`errors:` ***Reihe***

Eine Reihe detaillierter Fehlermeldungen, die vom ExtraHop-System aufgezeichnet wurden.

Zugriff nur auf `REDIS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isReqAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die Redis-Anfrage abgeschlossen war.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die Redis-Antwort abgeschlossen war.

Zugriff nur auf `REDIS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`method:` ***Schnur***

Die Redis-Methode wie GET oder KEYS.

`payload:` ***Puffer***

Der Hauptteil der Antwort oder Anfrage.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `REDIS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `Redis.commitRecord()` entweder auf einem `REDIS_REQUEST` oder `REDIS_RESPONSE` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

REDIS_REQUEST	REDIS_RESPONSE
clientIsExternal	clientIsExternal
clientZeroWnd	clientZeroWnd
method	error
receiverIsExternal	method
reqKey	processingTime
reqSize	receiverIsExternal
reqTransferTime	reqKey
isReqAborted	rspSize
senderIsExternal	rspTransferTime
serverZeroWnd	isRspAborted
	rspTimeToFirstByte
	rspTimeToLastByte
	senderIsExternal
	serverIsExternal
	serverZeroWnd

`reqKey:` ***Reihe***

Ein Array, das die mit der Anfrage gesendeten Redis-Schlüsselzeichenfolgen enthält.

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

`reqSize:` ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne Redis-Header.

`reqTransferTime:` ***Zahl***

Die Übertragungszeit der Anfrage, ausgedrückt in Millisekunden. Wenn die Anfrage in einem einzigen Paket enthalten ist, ist die Übertragungszeit Null. Wenn sich die Anfrage über mehrere Pakete erstreckt, ist der Wert die Zeitspanne zwischen der Erkennung des ersten Redis-Anforderungspakets und der Erkennung des letzten Paket durch das ExtraHop-System. Ein hoher Wert kann auf eine große Redis-Anfrage oder eine Netzwerkverzögerung hinweisen. Der Wert ist `NaN` wenn es keine gültige Messung gibt oder wenn der Zeitpunkt ungültig ist.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***

Die mittlere TCP-Roundtrip-Zeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

`rspTransferTime:` ***Zahl***

Die Antwortübertragungszeit, ausgedrückt in Millisekunden. Wenn die Antwort in einem einzigen Paket enthalten ist, ist die Übertragungszeit Null. Wenn sich die Antwort über mehrere Pakete erstreckt, ist der Wert die Zeitspanne zwischen der Erkennung des ersten Redis-Antwortpakets und der Erkennung des letzten Paket durch das ExtraHop-System. Ein hoher Wert kann auf eine große Redis-Antwort oder eine Netzwerkverzögerung hinweisen. Der Wert ist `NaN` wenn es keine gültige Messung gibt oder wenn der Zeitpunkt ungültig ist.

Zugriff nur auf `REDIS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes, ohne Redis-Header.

Zugriff nur auf `REDIS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToFirstByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum ersten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `REDIS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToLastByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum letzten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `REDIS_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

RFB

Die RFB Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `RFB_OPEN`, `RFB_CLOSE`, und `RFB_TICK` Ereignisse.

Ereignisse

`RFB_CLOSE`

Läuft, wenn eine RFB-Verbindung geschlossen ist.

`RFB_OPEN`

Läuft, wenn eine neue RFB-Verbindung geöffnet wird.

`RFB_TICK`

Läuft regelmäßig auf RFB-Flows.

Methoden

`commitRecord():` ***Leere***

Übergibt ein Datensatzobjekt an den Recordstore. Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`authType:` ***Zahl***

Die Zahl, die dem Sicherheitstyp entspricht, den der Client und der Server ausgehandelt haben.

Zugriff nur auf `RFB_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

Art der Sicherheit	Zahl
`Invalid`	`0`
`None`	`1`
`VNC Authentication`	`2`
`RealVNC`	`3-15`
`Tight`	`16`
`Ultra`	`17`
`TLS`	`18`
`VeNCrypt`	`19`
`GTK-VNC SASL`	`20`
`MD5 hash authentication`	`21`

Art der Sicherheit	Zahl
Colin Dean xvp	22
RealVNC	128-255

`authResult:` ***Zahl***

Zeigt an, ob die Authentifizierung erfolgreich war.

Wert	Beschreibung
0	Succeeded
1	Failed

`duration:` ***Zahl***

Die Dauer der RFB-Sitzung, ausgedrückt in Sekunden.

Zugriff nur auf `RFB_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`error:` ***Schnur***

Die detaillierte Fehlermeldung, die vom ExtraHop-System aufgezeichnet wurde.

Zugriff nur auf `RFB_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Record-Objekt, das durch einen Aufruf von an den Recordstore übergeben wurde `RFB.commitRecord()`.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

RFB_OPEN	RFB_TICK	RFB_CLOSE
authType	clientIsExternal	clientIsExternal
authResult	reqBytes	duration
clientIsExternal	receiverIsExternal	receiverIsExternal
error	reqL2Bytes	senderIsExternal
receiverIsExternal	reqPkts	serverIsExternal
senderIsExternal	reqRTO	
serverIsExternal	reqZeroWnd	
version	roundTripTime	
	rspBytes	
	rspL2Bytes	
	rspPkts	
	rspRTO	
	rspZeroWnd	
	senderIsExternal	
	serverIsExternal	

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Headern.

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`roundTripTime:` ***Zahl***

Die mittlere TCP-Roundtrip-Zeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

Zugriff nur auf `RFB_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`version:` ***Schnur***

Die vom Client und Server ausgehandelte Version des RFB-Protokolls.

Zugriff nur auf `RFB_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

RPC

Das `RPC` Klasse ermöglicht das Speichern von Metriken und den Zugriff auf Eigenschaften aus der MRPC (MSRPC) -Aktivität auf `RPC_REQUEST` und `RPC_RESPONSE` Ereignisse.

Ereignisse

`RPC_REQUEST`

Wird bei jeder RPC-Anfrage ausgeführt, die vom Gerät verarbeitet wird.

`RPC_RESPONSE`

Wird bei jeder RPC-Antwort ausgeführt, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `RPC_REQUEST` oder `RPC_RESPONSE` Ereignis.

Informationen zu den Standardeigenschaften, die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

`encryptionProtocol:` ***Schnur***

Das Protokoll, mit dem die Transaktion verschlüsselt ist.

`interface:` ***Schnur***

Der Name der RPC-Schnittstelle, z. B. `drsuapi` und `epmapper`.

`interfaceGUID:` ***Schnur***

Die GUID der RPC-Schnittstelle. Das Format der GUID enthält Bindestriche, wie im folgenden Beispiel gezeigt:

```
367abb81-9844-35f2-ad32-98f038001004
```

`isEncrypted:` ***Boolesch***

Der Wert ist wahr, wenn die Nutzlast verschlüsselt ist.

`isDecrypted:` ***Boolesch***

Der Wert ist wahr, wenn das ExtraHop-System die Transaktion sicher entschlüsselt und analysiert hat. Die Analyse des entschlüsselten Datenverkehrs kann komplexe Bedrohungen aufdecken, die sich im verschlüsselten Verkehr verstecken.

`isNDR64:` ***Boolesch | null***

Gibt an, ob die Anfrage oder Antwort mit der NDR64-Übertragungssyntax übertragen wurde. Wenn der `pduType` Die Eigenschaft ist keine Anfrage oder Antwort, der Wert ist Null.

`operation:` ***Schnur***

Der Name der RPC-Operation, z. B. `DRSGetNCChanges` und `ept_map`.

`opnum:` ***Zahl***

Das Opnum der RPC-Operation. Das Opnum ist die numerische ID der RPC-Operation .

`payload:` ***Puffer | null***

Das Buffer-Objekt, das den Hauptteil der Anfrage oder Antwort enthält. Wenn der `pduType` Eigenschaft ist keine Anfrage oder Antwort, der Wert ist Null.

`pduType:` ***Schnur***

Der PDU-Typ, der den Zweck der RPC-Nachricht angibt. Die folgenden Werte sind gültig:

- `ack`
- `alter_context`
- `alter_context_resp`
- `auth`
- `bind`
- `bind_ack`

- bind_nak
- cancel_ack
- cl_cancel
- co_cancel
- fack
- fault
- nocall
- orphaned
- ping
- response
- request
- reject
- shutdown
- working

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `RPC.commitRecord()` auf einem `RPC_REQUEST` oder `RPC_RESPONSE` Ereignis.

Das Standard-Datensatzobjekt kann die folgenden Eigenschaften enthalten:

- clientAddr
- clientBytes
- clientIsExternal
- clientL2Bytes
- clientPkts
- clientPort
- clientRTO
- clientZeroWnd
- interface
- operation
- proto
- receiverIsExternal
- roundTripTime
- senderIsExternal
- serverAddr
- serverBytes
- serverIsExternal
- serverL2Bytes
- serverPkts
- serverPort
- serverRTO
- serverZeroWnd

`reqBytes:` ***Zahl***

Die Zahl der L4 Anforderungsbytes, ausgenommen L4-Header.

`reqL2Bytes:` ***Zahl***

Die Zahl der L2 Anforderungsbytes, einschließlich L2-Headern.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Zeitüberschreitungen bei der erneuten Übertragung (RTOs).

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***

Die mittlere TCP-Roundtrip-Zeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Samples gibt.

`authType:` ***Schnur***

Der vom Client und Server ausgehandelte Sicherheitstyp. Die folgenden Typen sind gültig:

- `DIGEST`
- `DPA`
- `GSS_KERBEROS`
- `GSS_SCHANNEL`
- `KRB5`
- `MSN`
- `MQ`
- `NONE`
- `NTLMSSP`
- `SEC_CHAN`
- `SPNEGO`

Zugriff nur auf `RPC_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspBytes:` ***Zahl***

Die Zahl der L4 Antwortbytes, ausgenommen L4-Protokoll-Overhead, wie ACKs, Header und Neuübertragungen.

`rspL2Bytes:` ***Zahl***

Die Zahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie z. B. Header.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Zeitüberschreitungen bei der erneuten Übertragung (RTOs).

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`RTCP`

Das RTCP Klasse ermöglicht es Ihnen, Metriken zu speichern und auf Eigenschaften zuzugreifen `RTCP_MESSAGE` Ereignisse.

Ereignisse

`RTCP_MESSAGE`

Läuft auf jedem RTCP-UDP-Paket, das vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `RTCP_MESSAGE` Ereignis.

Informationen zu den Standardeigenschaften, die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal bestätigt, auch wenn der `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`callId:` ***Zeichenfolge***

Die Anruf-ID für die Verknüpfung mit einem SIP Fluss.

`packets:` ***Reihe***

Ein Array von RTCP-Paketobjekten, wobei jedes Objekt ein Paket darstellt und ein PacketType-Feld enthält. Jedes Objekt hat je nach Nachrichtentyp unterschiedliche Felder, wie unten beschrieben.

`packetType:` ***Zeichenfolge***

Die Art des Paket. Wenn der Pakettyp nicht erkennbar ist, dann `packetType` wird „Unbekanntes N“ sein, wobei N das ist RTP Wert des Steuerpakettyps.

Wert	Typ	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

In der folgenden Liste werden die Felder für jeden Paketobjekttyp beschrieben:

APP

`name:` ***Zeichenfolge***

Der Name, den die Person gewählt hat, die den Satz von APP-Paketen als einzigartig definiert. Wird als vier ASCII-Zeichen interpretiert, bei denen zwischen Groß- und Kleinschreibung unterschieden wird.

`ssrc:` ***Zahl***

Das SSRC des Absenders.

`value:` ***Puffer***

Die optionalen anwendungsabhängigen Daten.

BYE

`packetType:` ***Zahl***

Enthält die Nummer 203, um dies als RTCP-BYE-Paket zu identifizieren.

SR

`ntpTimestamp:` ***Zahl***

Der NTP-Zeitstempel, seit der Epoche (1. Januar 1970) in Millisekunden umgerechnet.

`reportBlocks:` ***Reihe***

Ein Array von Berichtsobjekten, die Folgendes enthalten:

`fractionLost:` ***Zahl***

Die 8-Bit-Zahl, die die Anzahl der verlorenen Pakete geteilt durch die Anzahl der erwarteten Pakete angibt.

`jitter:` ***Zahl***

Eine Schätzung der statistischen Varianz der Interarrival Time von RTP-Datenpaketen, ausgedrückt in Millisekunden.

`lastSR:` ***Zahl***

Die mittleren 32 Bit des NTP_TimeStamp, das als Teil des neuesten RTCP-Senderberichtspakets (SR) Paket Quell-SSRC empfangen wurde. Wenn noch keine SR empfangen wurde, wird dieses Feld auf Null gesetzt.

`lastSRDelay:` ***Zahl***

Die Verzögerung zwischen dem Empfang des letzten SR-Pakets vom Quell-SSRC und dem Senden dieses Empfangsblocks, ausgedrückt in Einheiten von 1/65536 Sekunden. Wenn noch kein SR-Paket empfangen wurde, wird dieses Feld auf Null gesetzt.

`packetsLost:` ***Zahl***

Die Gesamtzahl der RTP-Datenpakete vom Quell-SSRC, die seit Beginn des Empfangs verloren gegangen sind.

`seqNum:` ***Zahl***

Die höchste Sequenznummer, die vom Quell-SSRC empfangen wurde.

`ssrc:` ***Zahl***

Das SSRC des Absenders.

`rtpTimestamp:` ***Zahl***

Der RTP-Zeitstempel, seit der Epoche (1. Januar 1970) in Millisekunden umgerechnet.

`senderOctets:` ***Zahl***

Die Anzahl der Absender-Oktette.

`senderPkts:` ***Zahl***

Die Anzahl der Absenderpakete.

RR

`reportBlocks:` ***Reihe***

Ein Array von Berichtsobjekten, die Folgendes enthalten:

`fractionLost:` ***Zahl***

Die 8-Bit-Zahl, die die Anzahl der letzten Pakete geteilt durch die Anzahl der erwarteten Pakete angibt.

`jitter:` ***Zahl***

Eine Schätzung der statistischen Varianz des RTP-Datenpakets, ausgedrückt in Millisekunden.

`lastSR:` ***Zahl***

Die mittleren 32 Bit des NTP_TimeStamp, das als Teil des neuesten RTCP-Senderberichtspakets (SR) Paket Quell-SSRC empfangen wurde. Wenn noch keine SR empfangen wurde, wird dieses Feld auf Null gesetzt.

`lastSRDelay:` ***Zahl***

Die Verzögerung zwischen dem Empfang des letzten SR-Pakets vom Quell-SSRC und dem Senden dieses Empfangsberichtsblocks, ausgedrückt in Einheiten von 1/65536 Sekunden. Wenn noch kein SR-Paket empfangen wurde, wird dieses Feld auf Null gesetzt.

`packetsLost:` ***Zahl***

Die Gesamtzahl der RTP-Datenpakete vom Quell-SSRC, die seit Beginn des Empfangs verloren gegangen sind.

`seqNum:` ***Zahl***

Die höchste Sequenznummer, die vom Quell-SSRC empfangen wurde.

`ssrc:` ***Zahl***

Das SSRC des Absenders.

`ssrc:` ***Zahl***

Das SSRC des Absenders.

`SDES`

`descriptionBlocks:` ***Reihe***

Ein Array von Objekten, das Folgendes enthält:

`type:` ***Zahl***

Der SDES-Typ.

SDES-Typ	Abkürzung.	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:` ***Puffer***

Ein Puffer, der den Textteil des SDES-Pakets enthält.

`ssrc:` ***Zahl***

Das SSRC des Absenders.

XR

`ssrc:` ***Zahl***

Das SSRC des Absenders.

`xrBlocks:` ***Reihe***

Eine Reihe von Berichtsblöcken, die Folgendes enthalten:

`statSummary:` ***Objekt***

Nur Typ 6. Das `statSummary` Objekt enthält die folgenden Eigenschaften:

`beginSeq:` ***Zahl***

Die erste Sequenznummer für das Intervall.

`devJitter:` ***Zahl***

Die Standardabweichung der relativen Laufzeit zwischen jeweils zwei Paketserien im Sequenzintervall.

`devTTLOrHL:` ***Zahl***

Die Standardabweichung der TTL- oder Hop-Grenzwerte von Datenpaketen im Sequenznummernbereich.

`dupPackets:` ***Zahl***

Die Anzahl der doppelten Pakete im Sequenznummernintervall.

`endSeq:` ***Zahl***

Die letzte Sequenznummer für das Intervall.

`lostPackets:` ***Zahl***

Die Anzahl der verlorenen Pakete im Sequenznummernintervall.

`maxJitter:` ***Zahl***

Die maximale relative Übertragungszeit zwischen zwei Paketen im Sequenzintervall, ausgedrückt in Millisekunden.

`maxTTLOrHL:` ***Zahl***

Der maximale TTL- oder Hop-Limit-Wert von Datenpaketen im Sequenznummernbereich.

`meanJitter:` ***Zahl***

Die mittlere relative Übertragungszeit zwischen zwei Paketserien im Sequenzintervall, gerundet auf den nächsten Wert, der als RTP-Zeitstempel ausgedrückt werden kann, ausgedrückt in Millisekunden.

`meanTTLOrHL:` ***Zahl***

Der mittlere TTL- oder Hop-Limit-Wert von Datenpaketen im Sequenznummernbereich.

`minJitter:` ***Zahl***

Die minimale relative Übertragungszeit zwischen zwei Paketen im Sequenzintervall, ausgedrückt in Millisekunden.

`minTTLOrHL:` ***Zahl***

Der minimale TTL- oder Hop-Limit-Wert von Datenpaketen im Sequenznummernbereich.

`ssrc:` ***Zahl***

Das SSRC des Absenders.

`type:` ***Zahl***

Der XR-Blocktyp.

Blocktyp	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 Acquisition 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)

Blocktyp	Name
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)
31	CSB (Concealed Seconds Metrics Block)
32	MPEG2 Transport Stream PSI Decodability Statistics Block

`typeSpecific:` ***Zahl***

Der Inhalt dieses Feldes hängt vom Blocktyp ab.

`value:` ***Puffer***

Der Inhalt dieses Feldes hängt vom Blocktyp ab.

`voipMetrics:` ***Objekt***

Nur Typ 7. Das `voipMetrics` Objekt enthält die folgenden Eigenschaften:

`burstDensity:` ***Zahl***

Der Anteil der RTP-Datenpakete innerhalb der Burst-Perioden seit Beginn des Empfangs, die entweder verloren gegangen sind oder verworfen wurden.

`burstDuration:` ***Zahl***

Die mittlere Dauer, ausgedrückt in Millisekunden, der Burst-Perioden, die seit Beginn des Empfangs aufgetreten sind.

`discardRate:` ***Zahl***

Der Bruchteil der RTP-Datenpakete von der Quelle, die seit Beginn des Empfangs verworfen wurden, weil der empfangende Jitterpuffer zu spät oder zu früh eingetroffen ist, zu wenig oder zu viel läuft.

`endSystemDelay:` ***Zahl***

Die zuletzt geschätzte Verzögerung des Endsystems, ausgedrückt in Millisekunden.

`extRFactor:` ***Zahl***

Die externe R-Faktor-Qualitätsmetrik. Ein Wert von 127 gibt an, dass dieser Parameter nicht verfügbar ist.

`gapDensity:` ***Zahl***

Der Anteil der RTP-Datenpakete innerhalb der Inter-Burst-Lücken seit Beginn des Empfangs, die entweder verloren gegangen sind oder verworfen wurden.

`gapDuration:` ***Zahl***

Die mittlere Dauer der Lückenperioden, die seit Beginn des Empfangs aufgetreten sind, ausgedrückt in Millisekunden.

`gmin:` ***Zahl***

Die Lückenschwelle.

`jbAbsMax:` ***Zahl***

Die absolute maximale Verzögerung, ausgedrückt in Millisekunden, die der adaptive Jitter-Puffer im schlimmsten Fall erreichen kann.

`jbMaximum:` ***Zahl***

Die aktuelle maximale Jitter-Pufferverzögerung, die dem frühesten eintreffenden Paket entspricht, das nicht verworfen würde, ausgedrückt in Millisekunden.

`jbNominal:` ***Zahl***

Die aktuelle nominale Jitter-Pufferverzögerung, die der nominalen Jitter-Pufferverzögerung für Pakete entspricht , die exakt pünktlich ankommen, ausgedrückt in Millisekunden.

`lossRate:` ***Zahl***

Der Bruchteil der RTP-Datenpakete von der Quelle, die seit Beginn des Empfangs verloren gegangen sind.

`mosCQ:` ***Zahl***

Der geschätzte durchschnittliche Meinungswert für die Gesprächsqualität (MOS-CQ). Ein Wert von 127 gibt an, dass dieser Parameter nicht verfügbar ist.

`mosLQ:` ***Zahl***

Der geschätzte durchschnittliche Meinungswert für die Hörqualität (MOS-LQ). Ein Wert von 127 gibt an, dass dieser Parameter nicht verfügbar ist.

`noiseLevel:` ***Zahl***

Der Geräuschpegel, ausgedrückt in Dezibel.

`rerl:` ***Zahl***

Der Wert für die Restechorückflussdämpfung, ausgedrückt in Dezibel.

`rFactor:` ***Zahl***

Die R-Faktor-Qualitätsmetrik. Ein Wert von 127 gibt an, dass dieser Parameter nicht verfügbar ist.

`roundTripDelay:` ***Zahl***

Die zuletzt berechnete Round-Trip-Zeit (RTT) zwischen RTP-Schnittstellen, ausgedrückt in Millisekunden.

`rxConfig:` ***Zahl***

Das Konfigurationsbyte des Empfängers.

`signalLevel:` ***Zahl***

Der relative Pegel des Sprachsignals, ausgedrückt in Dezibel.

`ssrc:` ***Zahl***

Das SSRC des Absenders.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `RTCP.commitRecord()` auf einem `RTCP_MESSAGE` Ereignis.

Das Standard-Datensatzobjekt kann die folgenden Eigenschaften enthalten:

- `callId`

- clientIsExternal
- cName
- flowId
- receiverIsExternal
- senderIsExternal
- serverIsExternal
- signalingFlowId

  Die ID des entsprechenden SIP- oder SCCP-Flusses , der den vom RTCP-Flow überwachten VoIP-Anruf aushandelt.

RTP

Das RTP Klasse ermöglicht es Ihnen, Metriken zu speichern und auf Eigenschaften zuzugreifen RTP_OPEN, RTP_CLOSE, und RTP_TICK Ereignisse.

Ereignisse

RTP_CLOSE

Wird ausgeführt, wenn eine RTP-Verbindung geschlossen wird.

RTP_OPEN

Wird ausgeführt, wenn eine neue RTP-Verbindung geöffnet wird.

RTP_TICK

Läuft regelmäßig auf RTP-Flows.

Methoden

commitRecord(): ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem RTP_TICK Ereignis. Commits aufzeichnen am RTP_OPEN und RTP_CLOSE Ereignisse werden nicht unterstützt.

Informationen zu den Standardeigenschaften , die dem Datensatzobjekt zugewiesen wurden, finden Sie in der record Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn commitRecord() Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

bytes: ***Zahl***

Die Anzahl der gesendeten Byte.

Zugriff nur auf RTP_TICK Ereignisse; andernfalls tritt ein Fehler auf.

callId: ***Schnur***

Die dem SIP- oder SCCP-Flow zugeordnete Anruf-ID.

drops: ***Zahl***

Die Anzahl der erkannten verworfenen Pakete.

Zugriff nur auf RTP_TICK Ereignisse; andernfalls tritt ein Fehler auf.

dups: ***Zahl***

Die Anzahl der erkannten doppelten Pakete.

Zugriff nur auf RTP_TICK Ereignisse; andernfalls tritt ein Fehler auf.

jitter: ***Zahl***

Eine Schätzung der statistischen Varianz der Zwischenankunftszeit von Datenpaketen.

Zugriff nur auf RTP_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`l2Bytes:` ***Zahl***

Die Zahl der L2 Byte.

Zugriff nur auf RTP_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`mos:` ***Zahl***

Der geschätzte durchschnittliche Meinungswert für Qualität.

Zugriff nur auf RTP_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`outOfOrder:` ***Zahl***

Die Anzahl der erkannten Nachrichten, die nicht in der richtigen Reihenfolge sind.

Zugriff nur auf RTP_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`payloadType:` ***Schnur***

Der Typ der RTP-Nutzlast.

Zugriff nur auf RTP_TICK Ereignisse; andernfalls tritt ein Fehler auf.

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
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

payloadTypeId	payloadType
34	ITU-T H.263-1996 Video

`payloadTypeId:` ***Zahl***

Der numerische Wert des Nutzlasttyps. Siehe Tabelle unter `payloadType`.

Zugriff nur auf `RTP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`pkts:` ***Zahl***

Die Anzahl der gesendeten Pakete.

Zugriff nur auf `RTP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `RTP.commitRecord()` auf einem `RTP_TICK` Ereignis.

Das Standard-Datensatzobjekt kann die folgenden Eigenschaften enthalten:

- `bytes`
- `callId`
- `clientIsExternal`
- `drops`
- `dups`
- `flowId`
- `jitter`
- `l2Bytes`
- `mos`
- `outOfOrder`
- `payloadType`
- `payloadTypeId`
- `pkts`
- `receiverIsExternal`
- `rFactor`
- `senderIsExternal`
- `serverIsExternal`
- `signalingFlowId`

  Die ID des entsprechenden SIP- oder SCCP-Flows , der den vom RTP-Flow gestreamten VoIP-Anruf aushandelt.
- `ssrc`
- `version`

Greifen Sie nur auf Datensatzobjekte zu `RTP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`rFactor:` ***Zahl***

Die R-Faktor-Qualitätsmetrik.

Zugriff nur auf `RTP_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`ssrc:` ***Zahl***

Das SSRC des Absenders.

`version:` ***Zahl***

Die RTP-Versionsnummer.

SCCP

Das Skinny Client Control Protocol (SCCP) ist ein proprietäres Protokoll von Cisco für die Kommunikation mit VoIP-Geräten. Die SCCP Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen SCCP_MESSAGE Ereignisse.

Ereignisse

SCCP_MESSAGE

Läuft auf jeder SCCP-Nachricht, die vom Gerät verarbeitet wird.

Methoden

commitRecord(): ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem SCCP_MESSAGE Ereignis.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in record Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn commitRecord() Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

callId: ***Schnur***

Die Anruf-ID, die dem zugeordnet ist RTP Fluss.

callInfo: ***Objekt***

Ein Objekt, das Informationen über den aktuell aufgerufenen SCCP enthält. Das Objekt enthält die folgenden Felder:

callReference: ***Zahl***

Die eindeutige Kennung des Anrufs.

callType: ***Zahl***

Die ID des Anruftyps.

ID	Art des Anrufs
1	Inbound
2	Outbound
3	Forward

calledPartyName: ***Schnur***

Der Name des Empfängers des Anrufs.

calledPartyNumber: ***Schnur***

Die Telefonnummer des Empfängers des Anrufs.

callingPartyName: ***Schnur***

Der Name des Anrufers.

callingPartyNumber: ***Schnur***

Die Telefonnummer des Anrufers.

lineInstance: ***Zahl***

Die eindeutige Kennung der Leitung.

`callStats:` ***Objekt***

Ein Objekt, das Statistiken für den SCCP-Aufruf enthält, wie vom Client gemeldet und berechnet. Das Objekt enthält die folgenden Felder:

`reportedBytesIn:` ***Zahl***

Die Anzahl der L7 empfangene Bytes.

`reportedBytesOut:` ***Zahl***

Die Anzahl der L7 gesendete Byte.

`reportedJitter:` ***Zahl***

Der Grad des Paket-Jitters oder der Variation der Latenz während des Anrufs.

`reportedLatency:` ***Zahl***

Der Grad der Paketlatenz, ausgedrückt in Millisekunden, während des Anrufs.

`reportedPktsIn:` ***Zahl***

Die Anzahl der empfangenen Pakete.

`reportedPktsLost:` ***Zahl***

Die Anzahl der Pakete, die während des Anrufs verloren gegangen sind.

`reportedPktsOut:` ***Zahl***

Die Anzahl der gesendeten Pakete.

`msgType:` ***Schnur***

Der dekodierte SCCP-Nachrichtentyp.

`receiverBytes:` ***Zahl***

Die Anzahl der L4 Bytes vom Empfänger.

`receiverL2Bytes:` ***Zahl***

Die Anzahl der L2 Bytes vom Empfänger.

`receiverPkts:` ***Zahl***

Die Anzahl der Pakete vom Empfänger.

`receiverRTO:` ***Zahl***

Die Anzahl der Timeouts bei der erneuten Übertragung (RTOs) vom Empfänger.

`receiverZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster vom Empfänger aus.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `SCCP.commitRecord()` auf einem `SCCP_MESSAGE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`
- `msgType`
- `receiverBytes`
- `receiverIsExternal`
- `receiverL2Bytes`
- `receiverPkts`
- `receiverRTO`
- `receiverZeroWnd`
- `roundTripTime`
- `senderBytes`
- `senderIsExternal`
- `senderL2Bytes`
- `senderPkts`

- `senderRTO`
- `senderZeroWnd`
- `serverIsExternal`

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`senderBytes:` ***Zahl***

Die Anzahl der L4 Bytes vom Absender.

`senderL2Bytes:` ***Zahl***

Die Anzahl der L2 Bytes vom Absender.

`senderPkts:` ***Zahl***

Die Anzahl der Pakete vom Absender.

`senderRTO:` ***Zahl***

Die Anzahl der Timeouts bei der erneuten Übertragung (RTOs) vom Absender.

`senderZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster vom Absender.

SDP

Die SDP Mit dieser Klasse können Sie auf Eigenschaften zugreifen `SIP_REQUEST` und `SIP_RESPONSE` Ereignisse.

Die `SIP_REQUEST` und `SIP_RESPONSE` Ereignisse sind definiert in SIP Abschnitt.

Eigenschaften

`mediaDescriptions:` ***Reihe***

Eine Reihe von Objekten, die die folgenden Felder enthalten:

`attributes:` ***Reihe von Zeichenketten***

Die optionalen Sitzungsattribute.

`bandwidth:` ***Reihe von Zeichenketten***

Der optionale vorgeschlagene Bandbreitentyp und die Bandbreite, die von der Sitzung oder den Medien verbraucht werden sollen.

`connectionInfo:` ***Schnur***

Die Verbindungsdaten, einschließlich Netzwerktyp, Adresstyp und Verbindungsadresse. Kann je nach Adresstyp auch optionale Unterfelder enthalten.

`description:` ***Schnur***

Die Sitzungsbeschreibung, die eine oder mehrere Medienbeschreibungen enthalten kann. Jede Medienbeschreibung besteht aus Medien-, Port- und Transportprotokollfeldern.

`encryptionKey:` ***Schnur***

Die optionale Verschlüsselungsmethode und der Schlüssel für die Sitzung.

`mediaTitle:` ***Schnur***

Der Titel des Medienstreams.

`sessionDescription:` ***Objekt***

Ein Objekt, das die folgenden Felder enthält:

`attributes:` ***Reihe von Zeichenketten***

Die optionalen Sitzungsattribute.

`bandwidth:` ***Reihe von Zeichenketten***

Der optionale vorgeschlagene Bandbreitentyp und die Bandbreite, die von der Sitzung oder den Medien verbraucht werden sollen.

`connectionInfo:` ***Schnur***

Die Verbindungsdaten, einschließlich Netzwerktyp, Adresstyp und Verbindungsadresse. Kann je nach Adresstyp auch optionale Unterfelder enthalten.

`email:` ***Schnur***

Die optionale E-Mail-Adresse. Falls vorhanden, kann dies mehrere E-Mail-Adressen enthalten.

`encryptionKey:` ***Schnur***

Die optionale Verschlüsselungsmethode und der Schlüssel für die Sitzung.

`origin:` ***Schnur***

Der Urheber der Sitzung, einschließlich Benutzername, Adresse des Hosts des Benutzers, Sitzungs-ID und Versionsnummer.

`phoneNumber:` ***Schnur***

Die optionale Telefonnummer. Falls vorhanden, kann dies mehrere Telefonnummern enthalten.

`sessionInfo:` ***Schnur***

Die Beschreibung der Sitzung.

`sessionName:` ***Schnur***

Der Name der Sitzung.

`timezoneAdjustments:` ***Schnur***

Die Anpassungszeit und der Offset für eine geplante Sitzung.

`uri:` ***Schnur***

Die optionale URI soll mehr Informationen über die Sitzung bereitstellen.

`version:` ***Schnur***

Die Versionsnummer. Das sollte 0 sein.

`timeDescriptions:` ***Reihe***

Eine Reihe von Objekten, die die folgenden Felder enthalten:

`repeatTime:` ***Schnur***

Die Wiederholungszeit der Sitzung, einschließlich Intervall, aktiver Dauer und Offsets von der Startzeit.

`time:` ***Schnur***

Die Start- und Endzeiten für eine Sitzung.

`SFlow`

Die `SFlow` Klassenobjekt ermöglicht es Ihnen, Metriken zu speichern und auf Eigenschaften zuzugreifen `SFLOW_RECORD` Ereignisse. sFlow ist eine Sampling-Technologie zur Überwachung des Datenverkehrs in Datennetzwerken. sFlow tastet jedes n-te Paket ab und sendet es an den Collector, wohingegen NetFlow Daten von jedem Fluss an den Collector sendet. Der Hauptunterschied zwischen sFlow und NetFlow besteht darin, dass sFlow unabhängig von der Netzwerkschicht ist und alles abtasten kann.

Ereignisse

`SFLOW_RECORD`

Läuft nach Erhalt einer aus einem Flussnetz exportierten sFlow-Probe.

Methoden

`commitRecord()`: ***Leere***

Sendet ein Flow-Datensatzobjekt, das das sFlow-Format angibt, an den konfigurierten Recordstore auf einem `SFLOW_RECORD` Ereignis.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in der Datensatzeigenschaft unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `.commitRecord` wird mehrfach für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`deltaBytes`: ***Zahl***

Die Anzahl der L3-Bytes im Flow-Paket.

`dscp`: ***Zahl***

Die Zahl, die den letzten DSCP-Wert (Differentiated Services Code Point) des Flow-Pakets darstellt.

`dscpName`: ***Schnur***

Der Name, der dem DSCP-Wert zugeordnet ist, der von einem Gerät im Fluss übertragen wird. In der folgenden Tabelle sind bekannte DSCP-Namen aufgeführt:

Zahl	Name
8	CS1
10	AF11
12	AF12
14	AF13
16	CS2
18	AF21
20	AF22
22	AF23
24	CS3
26	AF31
28	AF32
30	AF33
32	CS4
34	AF41
36	AF42
38	AF43
40	CS5
44	VA
46	EF
48	CS6
56	CS7

`egressInterface:` ***Flow-Schnittstelle***

Die FlowInterface Objekt, das die Ausgabeschnittstelle identifiziert.

`format:` ***Schnur***

Das Format des sFlow-Datensatzes. Gültiger Wert ist „sFlow v5“.

`headerData:` ***Puffer***

Die Puffer Objekt, das die Rohbytes des gesamten Flow-Paket-Headers enthält.

`ingressInterface:` ***Flow-Schnittstelle***

Die FlowInterface Objekt, das die Eingabeschnittstelle identifiziert.

`ipPrecedence:` ***Zahl***

Der Wert des IP-Prioritätsfeldes, das dem DSCP des Flow-Pakets zugeordnet ist.

`ipproto:` ***Schnur***

Das dem Fluss zugeordnete IP-Protokoll, z. B. TCP oder UDP.

`network:` ***Flow-Netzwerk***

Gibt a zurück FlowNetwork Objekt, das den Exporteur identifiziert und die folgenden Eigenschaften enthält:

`id:` ***Schnur***

Die Kennung des FlowNetwork.

`ipaddr:` ***IP-Adresse***

Die IP-Adresse des FlowNetwork.

`record:` ***Objekt***

Das Flow-Datensatzobjekt, das über einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `SFlow.commitRecord()` auf einem `SFLOW_RECORD` Ereignis.

Das Standard-Datensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`
- `deltaBytes`
- `dscpName`
- `egressInterface`
- `format`
- `ingressInterface`
- `ipPrecedence`
- `ipproto`
- `network`
- `networkAddr`
- `receiverIsExternal`
- `senderIsExternal`
- `serverIsExternal`
- `tcpFlagName`
- `tcpFlags`

`tcpFlagNames:` ***Reihe***

Ein String-Array von TCP-Flag-Namen, wie `SYN` oder `ACK`, gefunden in den Flow-Paketen.

`tcpFlags:` ***Zahl***

Das bitweise `OR` aller TCP-Flags, die für den Fluss gesetzt sind.

`tos:` ***Zahl***

Die im IP-Header definierte ToS-Nummer (Type of Service).

SIP

Die SIP Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen SIP_REQUEST und SIP_RESPONSE Ereignisse.

Ereignisse

SIP_REQUEST

Läuft bei jeder SIP-Anfrage, die vom Gerät verarbeitet wird.

SIP_RESPONSE

Läuft auf jeder SIP-Antwort, die vom Gerät verarbeitet wird.

Methoden

commitRecord(): ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem SIP_REQUEST oder SIP_RESPONSE Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die für jedes Ereignis festgeschriebenen Standardeigenschaften finden Sie in der record Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn commitRecord() Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

findHeaders(name: ***Schnur***): ***Reihe***

Ermöglicht den Zugriff auf SIP-Header-Werte. Das Ergebnis ist ein Array von Header-Objekten (mit Namens- und Werteigenschaften), wobei die Namen mit dem Präfix der übergebenen Zeichenfolge übereinstimmen findHeaders.

Eigenschaften

callId: ***Schnur***

Die Anruf-ID für diese Nachricht.

from: ***Schnur***

Der Inhalt des From-Headers.

hasSDP: ***Boolescher Wert***

Der Wert ist true wenn diese Ereignis beinhaltet SDP Informationen.

headers: ***Objekt***

Ein Array-ähnliches Objekt, das den Zugriff auf SIP-Header-Namen und -Werte ermöglicht. Greifen Sie mit einer der folgenden Methoden auf einen bestimmten Header zu:

String-Eigenschaft:

Der Name des Headers, auf den wie ein Wörterbuch zugegriffen werden kann. Zum Beispiel:

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

numerische Eigenschaft:

Die Reihenfolge, in der Header auf dem Draht erscheinen. Das zurückgegebene Objekt hat einen Namen und eine Werteigenschaft. Numerische Eigenschaften sind nützlich, um über alle Header zu iterieren und Header mit doppelten Namen eindeutig zu identifizieren. Zum Beispiel:

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

```
    debug("headers[" + i + "].value: " + hdr.value);
}
```

Hinweis: Speichern `SIP.headers` to the Flow Store speichert nicht alle einzelnen Header-Werte. Es hat sich bewährt, die einzelnen Header-Werte im Flow-Speicher zu speichern.

`method:` ***Schnur***

Die SIP-Methode.

Name der Methode	Beschreibung
`ACK`	Bestätigt die Client hat eine endgültige Antwort auf eine INVITE-Anfrage erhalten.
`BYE`	Beendet einen Anruf. Kann entweder vom Anrufer oder vom Angerufenen gesendet werden.
`CANCEL`	Storniert alle ausstehenden Anfragen
`INFO`	Sendet während der Sitzung Informationen, die den Sitzungsstatus nicht ändern.
`INVITE`	Lädt ein Client um an einer Telefonsitzung teilzunehmen.
`MESSAGE`	Transportiert Sofortnachrichten mithilfe von SIP.
`NOTIFY`	Benachrichtige den Abonnenten über ein neues Ereignis.
`OPTIONS`	Fragt die Fähigkeiten von Servern ab.
`PRACK`	Vorläufige Anerkennung.
`PUBLISH`	Veröffentlichen Sie ein Ereignis auf dem Server.
`REFER`	Bitten Sie den Empfänger, eine SIP-Anfrage (Anrufweiterleitung) zu stellen.
`REGISTER`	Registriert die im To-Header-Feld aufgeführte Adresse bei einem SIP-Server.
`SUBSCRIBE`	Abonniert ein Benachrichtigungsereignis vom Notifier.
`UPDATE`	Ändert den Status einer Sitzung, ohne den Status des Dialogs zu ändern.

`payload:` ***Puffer*** | ***null***

Die Puffer Objekt, das die rohen Payload-Bytes der Ereignistransaktion enthält. Wenn die Nutzlast komprimiert wurde, wird der dekomprimierte Inhalt zurückgegeben .

Der Puffer enthält den *N* erste Byte der Nutzlast, wobei *N* ist die Anzahl der Nutzdaten-Bytes, spezifiziert durch Bytes zum Puffer Feld, wenn der Auslöser über die ExtraHop WebUI konfiguriert wurde. Die Standardanzahl von Bytes ist 2048. Weitere Informationen finden Sie unter Erweiterte Trigger-Optionen.

`processingTime:` ***Zahl***

Die Zeit zwischen der Anfrage und der ersten Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `SIP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `SIP.commitRecord()` entweder auf einem `SIP_REQUEST` oder `SIP_RESPONSE` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

SIP_REQUEST	SIP_RESPONSE
callId	callId
clientIsExternal	clientIsExternal
clientZeroWnd	clientZeroWnd
from	from
hasSDP	hasSDP
method	processingTime
receiverIsExternal	receiverIsExternal
reqBytes	roundTripTime
reqL2Bytes	rspBytes
reqPkts	rspL2Bytes
reqRTO	rspPkts
reqSize	rspRTO
senderIsExternal	rspSize
serverIsExternal	senderIsExternal
serverZeroWnd	serverIsExternal
to	serverZeroWnd
uri	statusCode
	to

`reqBytes:` ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

`reqL2Bytes:` ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

`reqPkts:` ***Zahl***

Die Anzahl der Anforderungspakete.

`reqRTO:` ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

`reqSize:` ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne SIP-Header.

Zugriff nur auf `SIP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`rspBytes:` ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

`rspL2Bytes:` ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

`rspPkts:` ***Zahl***

Die Anzahl der Antwortpakete.

`rspRTO:` ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes, ohne SIP-Header.

Zugriff nur auf `SIP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`statusCode:` ***Zahl***

Der Statuscode der SIP-Antwort.

Zugriff nur auf `SIP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

Die folgende Tabelle enthält vorläufige Antworten:

Zahl	Antwort
`100`	`Trying`
`180`	`Ringing`
`181`	`Call is Being Forwarded`
`182`	`Queued`
`183`	`Session In Progress`
`199`	`Early Dialog Terminated`

Die folgende Tabelle zeigt erfolgreiche Antworten:

Zahl	Antwort
`200`	`OK`
`202`	`Accepted`
`204`	`No Notification`

In der folgenden Tabelle werden Umleitungsantworten angezeigt:

Zahl	Antwort
`300`	`Multiple Choice`
`301`	`Moved Permanently`
`302`	`Moved Temporarily`

Zahl	Antwort
305	Use Proxy
380	Alternative Service

Die folgende Tabelle zeigt Client Reaktionen auf Fehler:

Zahl	Antwort
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

Zahl	Antwort
439	First Hop Lacks Outbound Support
470	Consent Needed
480	Temporarily Unavailable
481	Call/Transaction Does Not Exist
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

In der folgenden Tabelle werden Antworten auf Serverausfälle angezeigt:

Zahl	Antwort
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

In der folgenden Tabelle sind globale Fehlerreaktionen aufgeführt:

Name	Antwort
600	Busy Everywhere
603	Decline
604	Does Not Exist Anywhere
606	Not Acceptable

`to:` ***Schnur***

Der Inhalt des To-Headers.

`uri:` ***Schnur***

Die URI für SIP Anfrage oder Antwort.

SLP

Die `SLP` Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `SLP_MESSAGE` Ereignisse.

Ereignisse

`SLP_MESSAGE`

Läuft auf jeder SLP-Nachricht, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `SLP_MESSAGE` Ereignis.

Die festgeschriebenen Standardeigenschaften finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben Datensatz aufgerufen.

Eigenschaften

`attrList:` ***Schnur | null***

Die Attribute für die SLP-Nachricht in einer durch Kommas getrennten Liste.

`functionId:` ***Zahl***

Die numerische Funktions-ID der SLP-Nachricht, die der Nachrichtentyp-Zeichenfolge entspricht.

`msgType:` ***Schnur***

Die Zeichenfolge des SLP-Nachrichtentyps, die der numerischen Funktions-ID entspricht, wie in der folgenden Tabelle dargestellt:

Art der Nachricht	Funktions-ID
`Service Request`	`1`
`Service Reply`	`2`
`Service Registration`	`3`
`Service Deregister`	`4`
`Service Acknowledge`	`5`
`Attribute Request`	`6`
`Attribute Reply`	`7`
`DA Advertisement`	`8`
`Service Type Request`	`9`
`Service Type Reply`	`10`
`SA Advertisement`	`11`

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `SLP.commitRecord()` bei einem SLP_MESSAGE-Ereignis. Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`
- `functionId`
- `msgType`
- `receiverIsExternal`
- `scopeList`
- `senderIsExternal`
- `serverIsExternal`

`scopeList:` ***Schnur*** | ***null***

Der Bereich für die SLP-Nachricht in einer durch Kommas getrennten Liste.

SMPP

Die SMPP Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `SMPP_REQUEST` und `SMPP_RESPONSE` Ereignisse.

Hinweis: Die `mdn`, `shortcode`, und `error` Eigenschaften können sein `null`, je nach Verfügbarkeit und Relevanz.

Ereignisse

`SMPP_REQUEST`

Läuft bei jeder SMPP-Anfrage, die vom Gerät verarbeitet wird.

`SMPP_RESPONSE`

Läuft auf jeder SMPP-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `SMPP_RESPONSE` Ereignis. Commits aufzeichnen für `SMPP_REQUEST` Ereignisse werden nicht unterstützt.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`command:` ***Schnur***

Die SMPP-Befehls-ID.

`destination:` ***Schnur***

Die Zieladresse, wie sie in der `SMPP_REQUEST`. Der Wert ist `null` wenn dies für den aktuellen Befehlstyp nicht verfügbar ist.

`error:` ***Schnur***

Der Fehlercode, der command_status entspricht. Wenn der Befehlsstatus ROK ist, ist der Wert `null`.

Zugriff nur auf `SMPP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`message:` ***Puffer***

Der Inhalt des Feld short_message in den Nachrichten DELIVER_SM und SUBMIT_SM. Der Wert ist `null` falls nicht verfügbar oder nicht zutreffend.

Zugriff nur auf `SMPP_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Entspricht `rspTimeToFirstByte - reqTimeToLastByte`. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `SMPP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `SMPP.commitRecord()` auf einem `SMPP_RESPONSE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`
- `clientZeroWnd`
- `command`
- `destination`
- `error`
- `receiverIsExternal`
- `reqSize`
- `reqTimeToLastByte`
- `rspSize`
- `rspTimeToFirstByte`
- `rspTimeToLastByte`
- `senderIsExternal`
- `serverIsExternal`
- `serverZeroWnd`
- `source`
- `processingTime`

`reqSize:` ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne SMPP-Header.

`reqTimeToLastByte:` ***Zahl***

Die Zeit vom ersten Byte der Anforderung bis zum letzten Byte der Anforderung, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Anfragen oder wenn das Timing ungültig ist.

`rspSize:` ***Zahl***

Die Anzahl der L7-Antwortbytes, ohne SMPP-Header.

Zugriff nur auf `SMPP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToFirstByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum ersten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf `SMPP_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspTimeToLastByte:` ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum letzten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf SMPP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`source:` ***Schnur***

Die Quelladresse, wie sie in der SMPP_REQUEST. Der Wert ist `null` wenn dies für den aktuellen Befehlstyp nicht verfügbar ist.

SMTP

Die SMTP Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen SMTP_REQUEST und SMTP_RESPONSE Ereignisse.

Ereignisse

`SMTP_OPEN`

Läuft auf jeder SMTP-Begrüßung, die vom Gerät verarbeitet wird.

`SMTP_REQUEST`

Wird bei jeder SMTP-Anfrage ausgeführt, die vom Gerät verarbeitet wird.

`SMTP_RESPONSE`

Läuft auf jeder SMTP-Antwort, die vom Gerät verarbeitet wird.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem SMTP_RESPONSE Ereignis. Commits aufzeichnen für SMTP_REQUEST Ereignisse werden nicht unterstützt.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`dataSize:` ***Zahl***

Die Größe des Anhangs, ausgedrückt in Byte.

`domain:` ***Schnur***

Die Domain der Adresse, von der die Nachricht kommt.

`error:` ***Schnur***

Der Fehlercode, der dem Statuscode entspricht.

Zugriff nur auf SMTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`headers:` ***Objekt***

Ein Objekt, das den Zugriff auf SMTP-Header-Namen und -Werte ermöglicht.

Der Wert des `headers` Die Eigenschaft ist dieselbe, wenn auf eine der folgenden Seiten zugegriffen wird SMTP_REQUEST oder der SMTP_RESPONSE Ereignis.

`isEncrypted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Anwendung mit STARTTLS verschlüsselt ist.

`isReqAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die SMTP-Anfrage abgeschlossen ist.

`isRspAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn die Verbindung geschlossen wird, bevor die SMTP-Antwort abgeschlossen ist.

Zugriff nur auf SMTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`method:` ***Schnur***

Die SMTP-Methode.

`processingTime:` ***Zahl***

Die Serververarbeitungszeit, ausgedrückt in Millisekunden. Entspricht `rspTimeToFirstByte - reqTimeToLastByte`. Der Wert ist `NaN` bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf SMTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`recipientList:` ***Reihe von Zeichenketten***

Eine Liste von Empfängeradressen.

Der Wert des `recipientList` Die Eigenschaft ist dieselbe, wenn auf eine der folgenden Seiten zugegriffen wird SMTP_REQUEST oder der SMTP_RESPONSE Ereignis.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `SMTP.commitRecord()` auf einem SMTP_RESPONSE Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`
- `clientZeroWnd`
- `dataSize`
- `domain`
- `error`
- `isEncrypted`
- `isReqAborted`
- `isRspAborted`
- `method`
- `processingTime`
- `receiverIsExternal`
- `recipient`
- `recipientList`
- `reqBytes`
- `reqL2Bytes`
- `reqPkts`
- `reqRTO`
- `reqSize`
- `reqTimeToLastByte`
- `roundTripTime`
- `rspBytes`
- `rspL2Bytes`
- `rspPkts`
- `rspRTO`
- `rspSize`
- `rspTimeToFirstByte`
- `rspTimeToLastByte`
- `sender`
- `senderIsExternal`

- serverIsExternal
- serverZeroWnd
- statusCode
- statusText

Greifen Sie nur auf das Datensatzobjekt zu unter SMTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

reqBytes: ***Zahl***

Die Anzahl der L4 Anforderungsbytes, ausgenommen L4-Header.

reqL2Bytes: ***Zahl***

Die Anzahl der L2 Anforderungsbytes, einschließlich L2-Header.

reqPkts: ***Zahl***

Die Anzahl der Anforderungspakete.

reqRTO: ***Zahl***

Die Anzahl der Anfragen Timeouts bei der erneuten Übertragung (RTOs).

reqSize: ***Zahl***

Die Anzahl der L7-Anforderungsbytes, ohne SMTP-Header.

reqTimeToLastByte: ***Zahl***

Die Zeit vom ersten Byte der Anforderung bis zum letzten Byte der Anforderung, ausgedrückt in Millisekunden. Der Wert ist NaN bei falsch formatierten und abgebrochenen Anfragen oder wenn das Timing ungültig ist.

reqZeroWnd: ***Zahl***

Die Anzahl der Nullfenster in der Anfrage.

roundTripTime: ***Zahl***

Die mittlere TCP-Roundtrip-Zeit (RTT), ausgedrückt in Millisekunden. Der Wert ist NaN wenn es keine RTT-Proben gibt.

rspBytes: ***Zahl***

Die Anzahl der L4 Antwortbytes, ausgenommen Overhead für das L4-Protokoll, wie ACKs, Header und erneute Übertragungen.

rspL2Bytes: ***Zahl***

Die Anzahl der L2 Antwortbytes, einschließlich Protokoll-Overhead, wie Header.

rspPkts: ***Zahl***

Die Anzahl der Antwortpakete.

rspRTO: ***Zahl***

Die Anzahl der Antworten Timeouts bei der erneuten Übertragung (RTOs).

rspSize: ***Zahl***

Die Anzahl der L7-Antwortbytes, ohne SMTP-Header.

Zugriff nur auf SMTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspTimeToFirstByte: ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum ersten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist NaN bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf SMTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

rspTimeToLastByte: ***Zahl***

Die Zeit vom ersten Byte der Anfrage bis zum letzten Byte der Antwort, ausgedrückt in Millisekunden. Der Wert ist NaN bei falsch formatierten und abgebrochenen Antworten oder wenn das Timing ungültig ist.

Zugriff nur auf SMTP_RESPONSE Ereignisse; andernfalls tritt ein Fehler auf.

`rspZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster in der Antwort.

`sender:` ***Schnur***

Der Absender der Nachricht.

`statusCode:` ***Zahl***

Der SMTP-Statuscode der Antwort oder Begrüßung.

Zugriff nur auf SMTP_RESPONSE oder SMTP_OPEN Ereignisse; andernfalls tritt ein Fehler auf.

`statusText:` ***Schnur***

Die mehrzeilige Antwort- oder Begrüßungszeichenfolge.

Zugriff nur auf SMTP_RESPONSE oder SMTP_OPEN Ereignisse; andernfalls tritt ein Fehler auf.

SSH

Secure Socket Shell (SSH) ist ein Netzwerk Protokoll das bietet eine sichere Methode für die Fernanmeldung und andere Netzwerkdienste über ein ungesichertes Netzwerk. Die SSH Klassenobjekt ermöglicht es Ihnen, Metriken zu speichern und auf Eigenschaften zuzugreifen SSH_CLOSE, SSH_OPEN und SSH_TICK Ereignisse.

Ereignisse

SSH_CLOSE

Wird ausgeführt, wenn die SSH-Verbindung beendet wird, indem sie geschlossen, abgelaufen oder abgebrochen wurde.

SSH_OPEN

Wird ausgeführt, wenn die SSH-Verbindung nach dem Aushandeln der Sitzungsinformationen zum ersten Mal vollständig hergestellt ist. Wenn die Verhandlung fehlschlägt, weil der Schlüsselaustausch ungültig ist, SSH_OPEN Das Ereignis wird ausgeführt, wenn es einen ungültigen Austausch gibt, und dann SSH_TICK und SSH_CLOSE Ereignisse laufen unmittelbar hintereinander ab.

Wenn eine Verbindung vorher geschlossen wurde SSH_OPEN läuft, SSH_OPEN, SSH_TICK, und SSH_CLOSE unmittelbar hintereinander laufen.

SSH_TICK

Läuft regelmäßig auf SSH-Flows.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem SSH_OPEN, SSH_CLOSE, oder SSH_TICK Ereignis.

Das Ereignis bestimmt, welche Eigenschaften dem Record-Objekt zugewiesen werden. Die für jedes Ereignis festgeschriebenen Eigenschaften finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `.commitRecord` wird mehrfach für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`clientBytes:` ***Zahl***

Nach einem SSH_CLOSE Ereignis, die inkrementelle Anzahl von Anwendungsebenen Client seit dem letzten beobachtete Byte SSH_TICK Ereignis. Gibt nicht die Gesamtzahl der Byte für die Sitzung an.

`clientCipherAlgorithm:` ***Schnur***

Der Verschlüsselungsalgorithmus auf dem SSH Client.

`clientCompressionAlgorithm:` ***Schnur***

Der Komprimierungsalgorithmus, der auf Daten angewendet wird, die vom SSH-Client über die Verbindung übertragen werden.

`clientCompressionAlgorithmsClientToServer:` ***Schnur***

Die Komprimierungsalgorithmen, die der SSH-Client für die Kommunikation zwischen Client und Server unterstützt.

`clientCompressionAlgorithmsServerToClient:` ***Schnur***

Die Komprimierungsalgorithmen, die der SSH-Client für die Server-zu-Client-Kommunikation unterstützt.

`clientEncryptionAlgorithmsClientToServer:` ***Schnur***

Die Verschlüsselungsalgorithmen, die der SSH-Client für die Kommunikation zwischen Client und Server unterstützt.

`clientEncryptionAlgorithmsServerToClient:` ***Schnur***

Die Verschlüsselungsalgorithmen, die der SSH-Client für die Server-zu-Client-Kommunikation unterstützt.

`clientImplementation:` ***Schnur***

Die auf dem Client installierte SSH-Implementierung, z. B. OpenSSH oder PUTTY.

`clientKexAlgorithms:` ***Schnur***

Die vom Client unterstützten SSH-Schlüsselaustauschalgorithmen.

`clientL2Bytes:` ***Zahl***

Die inkrementelle Anzahl von L2 Seit dem letzten beobachtete Client-Bytes `SSH_TICK` Ereignis. Gibt nicht die Gesamtzahl der Byte für die Sitzung an.

Zugriff nur auf `SSH_CLOSE` und `SSH_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientMacAlgorithm:` ***Schnur***

Der Method Authentication Code (MAC) -Algorithmus auf dem SSH-Client.

`clientMacAlgorithmsClientToServer:` ***Schnur***

Die MAC-Algorithmen (Method Authentication Code), die der SSH-Client für die Kommunikation zwischen Client und Server unterstützt.

`clientMacAlgorithmsServerToClient:` ***Schnur***

Die MAC-Algorithmen (Method Authentication Code), die der SSH-Client für die Server-zu-Client-Kommunikation unterstützt.

`clientPkts:` ***Zahl***

Die inkrementelle Anzahl von Client-Paketen, die seit dem letzten beobachtet wurden `SSH_TICK` Ereignis. Gibt nicht die Gesamtzahl der Pakete für die Sitzung an.

Zugriff nur auf `SSH_CLOSE` und `SSH_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientRTO:` ***Zahl***

Die inkrementelle Anzahl von Client Timeouts bei der erneuten Übertragung (RTOs) beobachtet seit dem letzten `SSH_TICK` Ereignis. Gibt nicht die Gesamtzahl der RTOs für die Sitzung an.

Zugriff nur auf `SSH_CLOSE` und `SSH_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`clientVersion:` ***Schnur***

Die Version von SSH auf dem Client.

`clientZeroWnd:` ***Zahl***

Die Anzahl der vom Client gesendeten Nullfenster.

Zugriff nur auf `SSH_OPEN`, `SSH_CLOSE`, oder `SSH_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`duration:` ***Zahl***

Die Dauer der SSH-Verbindung, ausgedrückt in Millisekunden.

Zugriff nur auf `SSH_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`hasshAlgorithms:` ***Schnur***

Eine Zeichenfolge, die die Algorithmen für den SSH-Schlüsselaustausch, die Verschlüsselung, die Nachrichtenauthentifizierung und die Komprimierung enthält, die der Client für die SSH-Kommunikation unterstützt. Diese Algorithmen werden im Paket SSH_MSG_KEXINIT kommuniziert, das zu Beginn einer SSH-Verbindung gesendet wird.

`hassh:` ***Schnur***

Ein MD5-Hash der HashAlgorithms-Zeichenfolge.

`hasshServerAlgorithms:` ***Schnur***

Eine Zeichenfolge, die die Algorithmen für den SSH-Schlüsselaustausch, die Verschlüsselung, die Nachrichtenauthentifizierung und die Komprimierung enthält, die der Server für die SSH-Kommunikation unterstützt. Diese Algorithmen werden im Paket SSH_MSG_KEXINIT kommuniziert, das zu Beginn einer SSH-Verbindung gesendet wird.

`hasshServer:` ***Schnur***

Ein MD5-Hash der HashServerAlgorithms-Zeichenfolge.

`kexAlgorithm:` ***Schnur***

Der Key Exchange (Kex) -Algorithmus für die SSH-Verbindung.

`messageNumbers:` ***Reihe von Zahlen***

Die numerischen IDs der ausgetauschten SSH-Nachrichten, in chronologischer Reihenfolge aufgeführt. Das Array kann nicht mehr als 50 Einträge enthalten. Wenn mehr als 50 Nachrichten ausgetauscht werden, enthält das Array die 50 neuesten IDs.

Zugriff nur auf `SSH_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `SSH.commitRecord()` entweder auf einem `SSH_OPEN`, `SSH_CLOSE`, oder `SSH_TICK` Ereignis.

Das Ereignis, für das die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

SSH_TICK	SSH_OPEN	SSH_CLOSE
clientCipherAlgorithm	clientCipherAlgorithm	clientCipherAlgorithm
clientCompressionAlgorithm	clientCompressionAlgorithm	clientCompressionAlgorithm
clientImplementation	clientImplementation	clientImplementation
clientIsExternal	clientIsExternal	clientIsExternal
clientMacAlgorithm	clientMacAlgorithm	clientMacAlgorithm
clientVersion	clientVersion	clientVersion
clientZeroWnd	clientZeroWnd	clientZeroWnd
kexAlgorithm	kexAlgorithm	kexAlgorithm
receiverIsExternal	receiverIsExternal	receiverIsExternal
senderIsExternal	senderIsExternal	senderIsExternal
serverCipherAlgorithm	serverCipherAlgorithm	serverCipherAlgorithm

SSH_TICK	SSH_OPEN	SSH_CLOSE
serverCompressionAlgorithm	serverCompressionAlgorithm	serverCompressionAlgorithm
serverImplementation	serverImplementation	serverImplementation
serverIsExternal	serverIsExternal	serverIsExternal
serverMacAlgorithm	serverMacAlgorithm	serverMacAlgorithm
serverVersion	serverVersion	serverVersion
serverZeroWnd	serverZeroWnd	serverZeroWnd
		duration

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`serverBytes:` ***Zahl***

Die inkrementelle Anzahl von Server-Bytes auf Anwendungsebene, die seit dem letzten Mal beobachtet wurde `SSH_TICK` Ereignis. Gibt nicht die Gesamtzahl der Byte für die Sitzung an.

Zugriff nur auf `SSH_CLOSE` und `SSH_TICK` Ereignisse; andernfalls tritt ein Fehler auf.

`serverCipherAlgorithm:` ***Schnur***

Der Verschlüsselungsalgorithmus auf dem SSH-Server.

`serverCompressionAlgorithm:` ***Schnur***

Gibt die Art der Komprimierung zurück, die auf Daten angewendet wird, die vom SSH-Server über die Verbindung übertragen wurden.

`serverCompressionAlgorithmsClientToServer:` ***Schnur***

Die Komprimierungsalgorithmen, die der SSH-Server für die Client-zu-Server-Kommunikation unterstützt.

`serverCompressionAlgorithmsServerToClient:` ***Schnur***

Die Komprimierungsalgorithmen, die der SSH-Server für die Server-zu-Client-Kommunikation unterstützt.

`serverEncryptionAlgorithmsClientToServer:` ***Schnur***

Die Verschlüsselungsalgorithmen, die der SSH-Server für die Client-zu-Server-Kommunikation unterstützt.

`serverEncryptionAlgorithmsServerToClient:` ***Schnur***

Die Verschlüsselungsalgorithmen, die der SSH-Server für die Server-zu-Client-Kommunikation unterstützt.

`serverHostKey:` ***Schnur***

Die Base64-Kodierung des öffentlichen SSH-Schlüssels, der vom Server an den Client gesendet wird.

`serverHostKeyType:` ***Schnur***

Der Typ des öffentlichen SSH-Schlüssels, der vom Server an den Client gesendet wird, z. B. ssh-rsa oder ssh-ed25519.

`serverImplementation:` ***Schnur***

Die auf dem Server installierte SSH-Implementierung, z. B. OpenSSH oder PUTTY.

`serverKexAlgorithms:` ***Schnur***

Die vom Server unterstützten SSH-Schlüsselaustauschalgorithmen.

`serverL2Bytes:` ***Zahl***

Die inkrementelle Anzahl von L2 Server-Bytes, die seit dem letzten beobachtet wurden `SSH_TICK` Ereignis. Gibt nicht die Gesamtzahl der Byte für die Sitzung an.

Zugriff nur auf SSH_CLOSE und SSH_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`serverMacAlgorithm:` ***Schnur***

Der Method Authentication Code (MAC) -Algorithmus auf dem SSH-Server.

`serverMacAlgorithmsClientToServer:` ***Schnur***

Die MAC-Algorithmen (Method Authentication Code), die der SSH-Server für die Kommunikation zwischen Client und Server unterstützt.

`serverMacAlgorithmsServerToClient:` ***Schnur***

Die MAC-Algorithmen (Method Authentication Code), die der SSH-Server für die Server-zu-Client-Kommunikation unterstützt.

`serverPkts:` ***Zahl***

Die inkrementelle Anzahl von Serverpaketen, die seit dem letzten beobachtet wurden SSH_TICK Ereignis. Gibt nicht die Gesamtzahl der Pakete für die Sitzung an.

Zugriff nur auf SSH_CLOSE und SSH_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`serverRTO:` ***Zahl***

Die inkrementelle Anzahl von Server Timeouts bei der erneuten Übertragung (RTOs) beobachtet seit dem letzten SSH_TICK Ereignis. Gibt nicht die Gesamtzahl der RTOs für die Sitzung an.

Zugriff nur auf SSH_CLOSE und SSH_TICK Ereignisse; andernfalls tritt ein Fehler auf.

`serverVersion:` ***Schnur***

Die Version von SSH auf dem Server.

`serverZeroWnd:` ***Zahl***

Die Anzahl der vom Server gesendeten Nullfenster.

Zugriff nur auf SSH_OPEN, SSH_CLOSE, oder SSH_TICK Ereignisse; andernfalls tritt ein Fehler auf.

SSL

Das SSL Klasse ermöglicht es Ihnen, Metriken zu speichern und auf Eigenschaften zuzugreifen SSL_OPEN, SSL_CLOSE, SSL_ALERT, SSL_RECORD, SSL_HEARTBEAT, und SSL_RENEGOTIATE Ereignisse.

Ereignisse

SSL_ALERT

Wird ausgeführt, wenn ein SSL-Alert-Datensatz ausgetauscht wird.

SSL_CLOSE

Wird ausgeführt, wenn die SSL-Verbindung geschlossen wird.

SSL_HEARTBEAT

Wird ausgeführt, wenn ein SSL-Heartbeat-Datensatz ausgetauscht wird.

SSL_OPEN

Wird ausgeführt, wenn die SSL-Verbindung zum ersten Mal hergestellt wird.

SSL_PAYLOAD

Wird ausgeführt, wenn die entschlüsselte SSL-Nutzlast den im zugehörigen Auslöser konfigurierten Kriterien entspricht.

Je nach Fluss befindet sich die Nutzlast in den folgenden Eigenschaften:

- `Flow.payload1`
- `Flow.payload2`
- `Flow.client.payload`
- `Flow.server.payload`
- `Flow.sender.payload`

- `Flow.receiver.payload`

  Zusätzliche Payload-Optionen sind verfügbar, wenn Sie einen Auslöser erstellen, der bei diesem Ereignis ausgeführt wird. siehe Erweiterte Trigger-Optionen für weitere Informationen.

`SSL_RECORD`

Wird ausgeführt, wenn ein SSL-Datensatz ausgetauscht wird.

`SSL_RENEGOTIATE`

Wird bei einer SSL-Neuverhandlung ausgeführt.

Methoden

`addApplication(name:` ***Zeichenfolge*** `):` ***Leere***

Ordnet der benannten Anwendung eine SSL-Sitzung zu, um SSL-Metrikdaten über die Sitzung zu sammeln. Sie könnten zum Beispiel anrufen `SSL.addApplication()` um SSL-Zertifikatsdaten in einer Anwendung zuzuordnen.

Nachdem eine SSL-Sitzung mit einer Anwendung verknüpft wurde, ist diese Kopplung für die gesamte Dauer der Sitzung dauerhaft.

Rufen Sie nur an `SSL_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`commitRecord():` ***Leere***

Sendet einen Datensatz nur an den konfigurierten Recordstore am `SSL_ALERT`, `SSL_CLOSE`, `SSL_HEARTBEAT`, `SSL_OPEN`, oder `SSL_RENEGOTIATE` Ereignisse. Commits aufzeichnen am `SSL_PAYLOAD` und `SSL_RECORD` Ereignisse werden nicht unterstützt.

Informationen zu den Standardeigenschaften, die dem Datensatzobjekt zugewiesen wurden, finden Sie in der `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

`getClientExtensionData(extension_name | extension_id):` ***Puffer | Null***

Gibt die Daten für die angegebene Erweiterung zurück, wenn die Erweiterung als Teil der Hello-Nachricht vom Client übergeben wurde. kehrt zurück `null` wenn die Nachricht keine Daten enthält.

Rufen Sie nur an `SSL_OPEN` und `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`getServerExtensionData(extension_name | extension_id):` ***Puffer | Null***

Gibt Daten für die angegebene Erweiterung zurück, wenn die Erweiterung als Teil von übergeben wurde `Hello` Nachricht vom Server. kehrt zurück `null` wenn die Nachricht keine Daten enthält.

Rufen Sie nur an `SSL_OPEN` und `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`hasClientExtension(extension_name | extension_id):` ***boolesch***

kehrt zurück `true` für die angegebene Erweiterung, wenn die Erweiterung als Teil des übergeben wurde `Hello` Nachricht vom Client.

Rufen Sie nur an unter `SSL_OPEN` und `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`hasServerExtension(extension_name | extension_id):` ***boolesch***

kehrt zurück `true` für die angegebene Erweiterung, wenn die Erweiterung als Teil des übergeben wurde `Hello` Nachricht vom Server.

Rufen Sie nur an unter `SSL_OPEN` und `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

Die folgende Tabelle enthält eine Liste bekannter SSL-Erweiterungen.

ID	Name
`0`	`server_name`

ID	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	supported_groups
11	ec_point_formats
12	srp
13	signature_algorithms
14	use_srtp
15	heartbeat
16	application_layer_protocol_negotiation
17	status_request_v2
18	signed_certificate_timestamp
19	client_certificate_type
20	server_certificate_type
27	compress_certificate
28	record_size_limit
29	pwd_protect
30	pwd_clear
31	password_salt
35	session_ticket
41	pre_shared_key
42	early_data
43	supported_versions
44	cookie
45	psk_key_exchange_modes
47	certificate_authorities
48	oid_filters
49	post_handshake_auth

ID	Name
50	signature_algorithms_cert
51	key_share
65281	renegotiation_info
65486	encrypted_server_name

Die folgenden Erweiterungen werden von Anwendungen gesendet, um zu testen, ob Server unbekannte Erweiterungen verarbeiten können. Weitere Informationen zu diesen Erweiterungen finden Sie unter Anwendung von GREASE auf TLS-Erweiterbarkeit.

- 2570
- 6682
- 10794
- 14906
- 19018
- 23130
- 27242
- 31354
- 35466
- 39578
- 43690
- 47802
- 51914
- 56026
- 60138
- 64250

Eigenschaften

`alertCode:` ***Zahl***

Die numerische Darstellung der SSL-Warnung. Die folgende Tabelle zeigt die möglichen SSL-Warnungen, die in der `AlertDescription` Datenstruktur in RFC 2246:

Warnung	Zahl
close_notify	0
unexpected_message	10
bad_record_mac	20
decryption_failed	21
record_overflow	22
decompression_failure	30
handshake_failure	40
bad_certificate	42
unsupported_certificate	43
certificate_revoked	44
certificate_expired	45

Warnung	Zahl
certificate_unknown	46
illegal_parameter	47
unknown_ca	48
access_denied	49
decode_error	50
decrypt_error	51
export_restriction	60
protocol_version	70
insufficient_security	71
internal_error	80
user_canceled	90
no_renegotiation	100

Wenn die Sitzung undurchsichtig ist, ist der Wert SSL.ALERT_CODE_UNKNOWN (null).

Zugriff nur auf SSL_ALERT Ereignisse; andernfalls tritt ein Fehler auf.

alertCodeName: ***Schnur***

Der Name der SSL-Warnung, die dem Warncode zugeordnet ist. Sehen Sie die alertCode Eigenschaft für Warnungsnamen, die mit Warncodes verknüpft sind. Der Wert ist null wenn kein Name für den zugehörigen Alert-Code verfügbar ist.

Zugriff nur auf SSL_ALERT Ereignisse; andernfalls tritt ein Fehler auf.

alertLevel: ***Zahl***

Die numerische Darstellung der SSL-Warnstufe. Die folgenden möglichen Warnstufen sind in der AlertLevel Datenstruktur in RFC 2246:

- warning (1)
- fatal (2)

Wenn die Sitzung undurchsichtig ist, ist der Wert SSL.ALERT_LEVEL_UNKNOWN (null).

Zugriff nur auf SSL_ALERT Ereignisse; andernfalls tritt ein Fehler auf.

certificate: ***SSL-Zertifikat***

Das der Kommunikation zugeordnete SSL-Serverzertifikatsobjekt. Jeder Objekt enthält die folgenden Eigenschaften:

authorityInfoAccess: ***Objekt***

Ein Objekt, das Informationen von der Behörde enthält Information Access-Erweiterung, die Informationen enthält über die Zertifizierungsstelle (CA). Das Objekt enthält folgende Felder:

location: ***Schnur***

Die URL des Online-Zertifikatsstatusprotokolls (OCSP) Responder, der überprüfen kann, ob Zertifikat ist gültig.

method: ***Schnur***

Die OID der Methode, die der Zertifikatsaussteller kann mit aufgerufen werden.

authorityKeyIdentifier: ***Schnur***

Die Kennung für den öffentlichen Schlüssel der Zertifizierungsstelle (CA), ausgedrückt als Oktett-Zeichenfolge.

Hinweis: Dieses Feld enthält nicht die behördliche Zertifizierung Aussteller- oder Seriennummer.

`basicConstraints:` ***Objekt***

Ein Objekt, das Informationen aus den Basic Constraints enthält Erweiterung, die den Typ des Zertifikatsantragstellers angibt. Das Objekt enthält die folgenden Felder:

`ca:` ***Boolesch***

Gibt an, ob der Betreff des Zertifikats ein CA.

`pathlen:` ***Zahl***

Die maximale Anzahl von Zertifikaten, die angezeigt werden können in der Zertifikatskette danach Zertifikat.

`certificatePolicies:` ***Reihe von Saiten***

Ein Array von OIDs für die im Zertifikat angegebenen Richtlinien Erweiterung der Richtlinien. Qualifizierer sind darin nicht enthalten Reihe.

`crlDistributionPoints:` ***Reihe von Saiten***

Ein Array von Objekten, die Informationen über Server enthalten, Host-Zertifikatssperrlisten (CRLs) für den Server Zertifikat. Die Server sind in der CRL-Distribution angegeben Punkterweiterung (CDP). Jedes Objekt enthält Folgendes Felder:

CRL-Benutzer: Reihe von Saiten

Eine Reihe von Orten, an denen das Zertifikat der Der CRL-Aussteller kann abgerufen werden.

`distPoint:` ***Reihe von Saiten***

Eine Reihe von Orten, an denen die CRL sein kann abgerufen.

`reasons:` ***Reihe von Saiten***

Eine Reihe von Ursachencodes, die die Gründe angeben dass das Zertifikat von der CRL widerrufen werden könnte Verteilungspunkt.

`extensionOIDs:` ***Reihe von Saiten***

Ein Array von OIDs für die X509-Erweiterungen, spezifiziert in Zertifikat.

`extendedKeyUsage:` ***Reihe von Saiten***

Eine Reihe von Verwendungsmöglichkeiten für den öffentlichen Schlüssel des Serverzertifikats angegeben in der Erweiterung Extended Key Usage. Das Array kann enthalten die folgenden Zeichenketten:

- `serverAuth`
- `clientAuth`
- `emailProtection`
- `codeSigning`
- `OCSPSigning`
- `timeStamping`
- `anyExtendedKeyUsage`
- `nsSGC`

`fingerprint:` ***Schnur***

Die hexadezimale Darstellung des SHA-1-Hashs von Zertifikat. Die Zeichenfolge enthält keine Trennzeichen, wie in der folgend Beispiel:

```
55F30E6D49E19145CF680E8B7E3DC8FC7041DC81
```

Das Der SHA-1-Zertifikat-Hash wird im Serverzertifikat angezeigt Dialogfeld der meisten Browser.

`fingerprintSHA256:` ***Schnur***

Die hexadezimale Darstellung des SHA-256-Hashs von Zertifikat. Die Zeichenfolge enthält keine Trennzeichen, wie in der folgend Beispiel:

```
468C6C84DB844821C9CCB0983C78D1CC05327119B894B5CA1C6A1318784D3675
```

Das Der SHA-256-Zertifikat-Hash wird im Serverzertifikat angezeigt Dialogfeld der meisten Browser.

`getExtensionDataByOID(extension_oid):` ***Puffer***

Methode, die ein Pufferobjekt zurückgibt, das den Wert von angegebene Erweiterung, ausgedrückt als Oktett-Zeichenfolge. Gibt Null zurück wenn die OID nicht existiert oder das Serverzertifikat nicht existiert enthalten die Erweiterung.

`inhibitAnyPolicy:` ***Zahl***

Die in der Inhibit AnyPolicy-Erweiterung angegebene Nummer, die begrenzt die Anzahl der Zertifikate, die die AnyPolicy-Erweiterung enthält wird angewendet auf. Die Zahl gibt an, wie viele weitere Nicht selbst ausgestellte Zertifikate in der Kette sind betroffen von AnyPolicy-Erweiterung.

`isSelfSigned:` ***Boolesch***

Der Wert ist `true` wenn das Serverzertifikat selbstsigniert.

`issuer:` ***Schnur***

Der allgemeine Name des Ausstellers des Serverzertifikats. Der Wert ist `null` wenn der Emittent nicht verfügbar ist.

`issuerAlternativeNames:` ***Reihe von Saiten***

Ein Array von Issuer Alternative Names (IANs), spezifiziert in der Serverzertifikat.

`issuerDistinguishedName:` ***Objekt***

Ein Objekt, das Informationen über den definierten Namen enthält des Zertifikatsausstellers. Jedes Objekt enthält Folgendes Eigenschaften:

`commonName:` ***Schnur***

Der gebräuchliche Name (CN).

`country:` ***Reihe von Saiten***

Der Ländername (C).

`emailAddress:` ***Schnur***

Die E-Mail-Adresse.

`organization:` ***Reihe von Saiten***

Der Name der Organisation (O).

`organizationalUnit:` ***Reihe von Saiten***

Der Name der Organisationseinheit (OU).

`locality:` ***Reihe von Saiten***

Der Ortsname (L).

`stateOrProvince:` ***Reihe von Saiten***

Der Name des Bundesstaats oder der Provinz (ST).

`keySize:` ***Zahl***

Die Schlüsselgröße des Serverzertifikats.

`keyUsage:` ***Reihe von Saiten***

Eine Reihe von Verwendungsmöglichkeiten für den öffentlichen Schlüssel des Serverzertifikats in der Key Usage-Erweiterung angegeben. Das Array kann das enthalten folgende Zeichenketten:

- `digitalSignature`

- nonRepudiation
- keyEncipherment
- dataEncipherment
- keyAgreement
- keyCertSign
- cRLSign
- encipherOnly
- decipherOnly

`notAfter:` ***Zahl***

Die Ablaufzeit des Serverzertifikats, ausgedrückt in UTC.

`notBefore:` ***Zahl***

Die Startzeit des Serverzertifikats, ausgedrückt in UTC. Das Das Serverzertifikat ist vor diesem Zeitpunkt nicht gültig.

`nsComment:` ***Schnur***

Der in der Netscape Comment-Erweiterung angegebene Kommentar. Das Ein Kommentar wird manchmal in Browsern angezeigt, wenn Benutzer den Serverzertifikat.

`ocspNoCheck:` ***Boolesch***

Gibt an, ob dem Signaturzertifikat vertraut werden kann, ohne Überprüfung durch den OCSP-Responder.

`policyConstraints:` ***Objekt***

Ein Objekt, das Informationen aus den Policy Constraints enthält Erweiterung, die Validierungseinschränkungen für CA spezifiziert zertifikate.

`requireExplicitPolicy:` ***Zahl***

Gibt die maximale Anzahl von benachbarten Zertifikate in der Kette, die nicht geben Sie eine explizite Richtlinie an.

`inhibitPolicyMapping:` ***Zahl***

Gibt die maximale Anzahl von benachbarten Zertifikate in der Zertifikatskette vor der Richtlinie Zuordnungen werden ignoriert.

`policyMappings:` ***Reihe von Objekte***

Ein Array von Objekten, das Informationen aus der Richtlinie enthält Mappings-Erweiterung, die auf gleichwertige Richtlinien hinweist zueinander. Jedes Objekt enthält die folgenden Felder.

`issuerDomainPolicy:` ***Schnur***

Die OID der Emittentenrichtlinie.

`subjectDomainPolicy:` ***Schnur***

Die OID der Betreffrichtlinie.

`publicKeyCurveName:` ***Schnur***

Der Name der elliptischen Standardkurve, die die Kryptographie von der öffentliche Schlüssel basiert auf. Dieser Wert wird durch die OID bestimmt oder explizite Kurvenparameter, die im Zertifikat angegeben sind.

`publicKeyExponent:` ***Schnur | Null***

Eine hexadezimale String-Darstellung des Exponenten des öffentlichen Schlüssels. Das Die Zeichenfolge wird im Dialogfeld für das Client-Zertifikat der meisten angezeigt Browser, aber ohne Leerzeichen.

`publicKeyHasExplicitCurve:` ***Boolesch | Null***

Gibt an, ob das Zertifikat explizite Parameter angibt für die elliptische Kurve des öffentlichen Schlüssels.

`publicKeyModulus:` ***Schnur | Null***

Eine hexadezimale String-Darstellung des öffentlichen Schlüsselmoduls. Das Die Zeichenfolge wird im Dialogfeld für das Client-Zertifikat der meisten angezeigt Browser, aber ohne Leerzeichen, wie `010001`

`serial:` ***Schnur | Null***

Die Seriennummer, die dem Zertifikat vom Zertifikat zugewiesen wurde Behörde (CA).

`signatureAlgorithm:` ***Schnur | Null***

Der Algorithmus, der zum Signieren des Serverzertifikats angewendet wurde. Das Die folgende Tabelle zeigt einige der möglichen Werte:

RFC	Algorithmus
`RFC 3279`	• `md2WithRSAEncryption` • `md5WithRSAEncryption` • `sha1WithRSAEncryption`
`RFC 4055`	• `sha224WithRSAEncryption` • `sha256WithRSAEncryption \` • `sha384WithRSAEncryption` • `sha512WithRSAEncryption`
`RFC 4491`	• `id-GostR3411-94-with-Gost3410-94` • `id-GostR3411-94-with-Gost3410-2001`

`subject:` ***Schnur***

Der Subject Common Name (CN) des Serverzertifikats.

`subjectAlternativeNames:` ***Reihe***

Ein Array von Zeichenketten, die alternativen Betreffnamen entsprechen (SANs) sind im Serverzertifikat enthalten. Unterstützte SANs sind DNS-Namen, E-Mail-Adressen, URIs und IP-Adressen.

`subjectDistinguishedName:` ***Objekt***

Ein Objekt, das Informationen über den definierten Namen enthält des Zertifikatssubjekts. Jedes Objekt enthält Folgendes Eigenschaften:

`commonName:` ***Schnur***

Der gebräuchliche Name (CN).

`country:` ***Reihe von Saiten***

Der Ländername (C).

`emailAddress:` ***Schnur***

Die E-Mail-Adresse.

`organization:` ***Reihe von Saiten***

Der Name der Organisation (O).

`organizationalUnit:` ***Reihe von Saiten***

Der Name der Organisationseinheit (OU).

`locality:` ***Reihe von Saiten***

Der Ortsname (L).

`stateOrProvince:` ***Reihe von Saiten***

Der Name des Bundesstaats oder der Provinz (ST).

`subjectKeyIdentifier:` ***Schnur***

Die Kennung für den öffentlichen Schlüssel des Zertifikatsinhabers, ausgedrückt als Oktett-Zeichenfolge.

`certificates:` ***Reihe von Objekten***

Ein Array von Zertifikatsobjekten für jedes SSL-Zwischenzertifikat. Das Endentitätszertifikat, auch bekannt als Blattzertifikat, ist das erste Objekt im Array; dieses Objekt wird auch vom zurückgegeben `certificate` Eigentum.

`cipherSuite:` ***Zeichenfolge***

Eine Zeichenfolge, die die kryptografische Verschlüsselungssuite darstellt, die zwischen dem Server und dem Client ausgehandelt wurde.

`cipherSuitesSupported:` ***Reihe von Objekten | Null***

Ein Array von Objekten mit den folgenden Eigenschaften, die die vom SSL-Client unterstützten Verschlüsselungssammlungen angeben:

`name:` ***Zeichenfolge***

Der Name der Verschlüsselungssuite.

`type:` ***Zahl***

Die Nummer der Verschlüsselungssuite.

Zugriff nur auf `SSL_OPEN` oder `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`cipherSuiteType:` ***Zahl***

Der numerische Wert, der die kryptografische Verschlüsselungssuite darstellt, die zwischen dem Server und dem Client ausgehandelt wurde. Mögliche Werte werden von der IANA TLS Cipher Suite Registry definiert.

`clientBytes:` ***Zahl***

Die Anzahl der Byte, die der Client seit dem letzten Mal gesendet hat `SSL_RECORD` Ereignis.

Zugriff nur auf `SSL_RECORD` oder `SSL_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`clientCertificate:` ***SSL-Zertifikat***

Das der Kommunikation zugeordnete SSL-Clientzertifikatsobjekt. Jeder Objekt enthält die folgenden Eigenschaften:

`authorityInfoAccess:` ***Objekt***

Ein Objekt, das Informationen von der Behörde enthält Information Access-Erweiterung, die Informationen enthält über die Zertifizierungsstelle (CA). Das Objekt enthält folgende Felder:

`location:` ***Schnur***

Die URL des Online-Zertifikatsstatusprotokolls (OCSP) Responder, der überprüfen kann, ob Zertifikat ist gültig.

`method:` ***Schnur***

Die OID der Methode, die der Zertifikatsaussteller kann mit aufgerufen werden.

`authorityKeyIdentifier:` ***Schnur***

Die Kennung für den öffentlichen Schlüssel der Zertifizierungsstelle (CA), ausgedrückt als Oktett-Zeichenfolge.

Hinweis: Dieses Feld enthält nicht die behördliche Zertifizierung Aussteller- oder Seriennummer.

`basicConstraints:` ***Objekt***

Ein Objekt, das Informationen aus den Basic Constraints enthält Erweiterung, die den Typ des Zertifikatsantragstellers angibt. Das Objekt enthält die folgenden Felder:

`ca:` ***Boolesch***

Gibt an, ob der Betreff des Zertifikats ein CA.

`pathlen:` ***Zahl***

Die maximale Anzahl von Zertifikaten, die angezeigt werden können in der Zertifikatskette danach Zertifikat.

`certificatePolicies:` ***Reihe von Saiten***

Ein Array von OIDs für die im Zertifikat angegebenen Richtlinien Erweiterung der Richtlinien. Qualifizierer sind darin nicht enthalten Reihe.

`crlDistributionPoints:` ***Reihe von Saiten***

Ein Array von Objekten, die Informationen über Server enthalten, Host-Zertifikatssperrlisten (CRLs) für den Client Zertifikat. Die Server sind in der CRL-Distribution angegeben Punkterweiterung (CDP). Jedes Objekt enthält Folgendes Felder:

CRL-Benutzer: Reihe von Saiten

Eine Reihe von Orten, an denen das Zertifikat der Der CRL-Aussteller kann abgerufen werden.

`distPoint:` ***Reihe von Saiten***

Eine Reihe von Orten, an denen die CRL sein kann abgerufen.

`reasons:` ***Reihe von Saiten***

Eine Reihe von Ursachencodes, die die Gründe angeben dass das Zertifikat von der CRL widerrufen werden könnte Verteilungspunkt.

`extensionOIDs:` ***Reihe von Saiten***

Ein Array von OIDs für die im Client angegebenen X509-Erweiterungen Zertifikat.

`extendedKeyUsage:` ***Reihe von Saiten***

Eine Reihe von Verwendungsmöglichkeiten für den öffentlichen Schlüssel des Client-Zertifikats angegeben in der Erweiterung Extended Key Usage. Das Array kann enthalten die folgenden Zeichenketten:

- `serverAuth`
- `clientAuth`
- `emailProtection`
- `codeSigning`
- `OCSPSigning`
- `timeStamping`
- `anyExtendedKeyUsage`
- `nsSGC`

`fingerprint:` ***Schnur***

Die hexadezimale Darstellung des SHA-1-Hashs des Client Zertifikat. Die Zeichenfolge enthält keine Trennzeichen, wie in der folgend Beispiel:

```
55F30E6D49E19145CF680E8B7E3DC8FC7041DC81
```

`fingerprintSHA256:` ***Schnur***

Die hexadezimale Darstellung des SHA-256-Hashs des Client Zertifikat. Die Zeichenfolge enthält keine Trennzeichen, wie in der folgend Beispiel:

```
468C6C84DB844821C9CCB0983C78D1CC05327119B894B5CA1C6A1318784D3675
```

`getExtensionDataByOID(extension_oid):` ***Puffer***

Methode, die ein Pufferobjekt zurückgibt, das den Wert von angegebene Erweiterung, ausgedrückt als Oktett-Zeichenfolge. Gibt Null zurück wenn die OID nicht existiert oder das Client-Zertifikat nicht existiert enthalten die Erweiterung.

`keySize:` ***Zahl***

Die Schlüsselgröße des Client-Zertifikats.

`keyUsage:` ***Reihe von Saiten***

Eine Reihe von Verwendungsmöglichkeiten für den öffentlichen Schlüssel des Client-Zertifikats in der Key Usage-Erweiterung angegeben. Das Array kann das enthalten folgende Zeichenketten:

- `digitalSignature`
- `nonRepudiation`
- `keyEncipherment`
- `dataEncipherment`
- `keyAgreement`
- `keyCertSign`
- `cRLSign`
- `encipherOnly`
- `decipherOnly`

`inhibitAnyPolicy:` ***Zahl***

Die in der Inhibit AnyPolicy-Erweiterung angegebene Nummer, die begrenzt die Anzahl der Zertifikate, die die AnyPolicy-Erweiterung enthält wird angewendet auf. Die Zahl gibt an, wie viele weitere Nicht selbst ausgestellte Zertifikate in der Kette sind betroffen von AnyPolicy-Erweiterung.

`isSelfSigned:` ***Boolesch***

Der Wert ist `true` wenn das Client-Zertifikat selbstsigniert.

`issuer:` ***Schnur | Null***

Der allgemeine Name des Ausstellers des Client-Zertifikats. Der Wert ist `null` wenn der Emittent nicht verfügbar ist.

`issuerDistinguishedName:` ***Objekt***

Ein Objekt, das Informationen über den definierten Namen enthält des Zertifikatsausstellers. Jedes Objekt enthält Folgendes Eigenschaften:

`commonName:` ***Schnur***

Der gebräuchliche Name (CN).

`country:` ***Reihe von Saiten***

Der Ländername (C).

`emailAddress:` ***Schnur***

Die E-Mail-Adresse.

`organization:` ***Reihe von Saiten***

Der Name der Organisation (O).

`organizationalUnit:` ***Reihe von Saiten***

Der Name der Organisationseinheit (OU).

`locality:` ***Reihe von Saiten***

Der Ortsname (L).

`stateOrProvince:` ***Reihe von Saiten***

Der Name des Bundesstaats oder der Provinz (ST).

`issuerAlternativeNames:` ***Reihe von Saiten***

Ein Array von Issuer Alternative Names (IANs), spezifiziert in der Client-Zertifikat.

`notAfter:` ***Zahl***

Die Ablaufzeit des Client-Zertifikats, ausgedrückt in UTC.

`notBefore:` ***Zahl***

Die Startzeit des Client-Zertifikats, ausgedrückt in UTC. Das Das Client-Zertifikat ist vor diesem Zeitpunkt nicht gültig.

`nsComment:` ***Schnur***

Der in der Netscape Comment-Erweiterung angegebene Kommentar. Das Ein Kommentar wird manchmal in Browsern angezeigt, wenn Benutzer den Client-Zertifikat.

`ocspNoCheck:` ***Boolesch***

Gibt an, ob dem Signaturzertifikat vertraut werden kann, ohne Überprüfung durch den OCSP-Responder.

`policyConstraints:` ***Objekt***

Ein Objekt, das Informationen aus den Policy Constraints enthält Erweiterung, die Validierungseinschränkungen für CA spezifiziert zertifikate.

`requireExplicitPolicy:` ***Zahl***

Gibt die maximale Anzahl von benachbarten Zertifikate in der Kette, die nicht geben Sie eine explizite Richtlinie an.

`inhibitPolicyMapping:` ***Zahl***

Gibt die maximale Anzahl von benachbarten Zertifikate in der Zertifikatskette vor der Richtlinie Zuordnungen werden ignoriert.

`publicKeyCurveName:` ***Schnur***

Der Name der elliptischen Standardkurve, die die Kryptographie von der öffentliche Schlüssel basiert auf. Dieser Wert wird durch die OID bestimmt oder explizite Kurvenparameter, die im Zertifikat angegeben sind.

`publicKeyExponent:` ***Schnur | Null***

Eine hexadezimale String-Darstellung des Exponenten des öffentlichen Schlüssels.

`publicKeyHasExplicitCurve:` ***Boolesch | Null***

Gibt an, ob das Zertifikat explizite Parameter angibt für die elliptische Kurve des öffentlichen Schlüssels.

`publicKeyModulus:` ***Schnur | Null***

Eine hexadezimale String-Darstellung des öffentlichen Schlüsselmoduls, z. B. `010001`.

`policyMappings:` ***Reihe von Objekte***

Ein Array von Objekten, das Informationen aus der Richtlinie enthält Mappings-Erweiterung, die auf gleichwertige Richtlinien hinweist zueinander. Jedes Objekt enthält die folgenden Felder.

`issuerDomainPolicy:` ***Schnur***

Die OID der Emittentenrichtlinie.

`subjectDomainPolicy:` ***Schnur***

Die OID der Betreffrichtlinie.

`signatureAlgorithm:` ***Schnur | Null***

Der Algorithmus, der zum Signieren des Client-Zertifikats angewendet wurde. Das Die folgende Tabelle zeigt einige der möglichen Werte:

RFC	Algorithmus
RFC 3279	• `md2WithRSAEncryption` • `md5WithRSAEncryption` • `sha1WithRSAEncryption`
RFC 4055	• `sha224WithRSAEncryption` • `sha256WithRSAEncryption` • `sha384WithRSAEncryption` • `sha512WithRSAEncryption`

RFC	Algorithmus
RFC 4491	• id-GostR3411-94-with-Gost3410-94 • id-GostR3411-94-with-Gost3410-2001

subject: ***Schnur***

Der allgemeine Name (Subject Common Name, CN) des Client-Zertifikats.

subjectAlternativeNames: ***Reihe***

Ein Array von Zeichenketten, die alternativen Betreffnamen entsprechen (SANs) sind im Client-Zertifikat enthalten. Unterstützte SANs sind DNS-Namen, E-Mail-Adressen, URIs und IP-Adressen.

subjectDistinguishedName: ***Objekt***

Ein Objekt, das Informationen über den definierten Namen enthält des Zertifikatssubjekts. Jedes Objekt enthält Folgendes Eigenschaften:

commonName: ***Schnur***

Der gebräuchliche Name (CN).

country: ***Reihe von Saiten***

Der Ländername (C).

emailAddress: ***Schnur***

Die E-Mail-Adresse.

organization: ***Reihe von Saiten***

Der Name der Organisation (O).

organizationalUnit: ***Reihe von Saiten***

Der Name der Organisationseinheit (OU).

locality: ***Reihe von Saiten***

Der Ortsname (L).

stateOrProvince: ***Reihe von Saiten***

Der Name des Bundesstaats oder der Provinz (ST).

subjectKeyIdentifier: ***Schnur***

Die Kennung für den öffentlichen Schlüssel des Client-Zertifikats Betreff, ausgedrückt als Oktett-Zeichenfolge.

clientCertificates: ***Reihe von Objekten***

Ein Array von Zertifikatsobjekten für jedes SSL-Client-Zwischenzertifikat. Das Endentitätszertifikat, auch bekannt als Blattzertifikat, ist das erste Objekt im Array; dieses Objekt wird auch vom zurückgegeben clientCertificate Eigentum.

clientCertificateRequested: ***Boolesch***

Der Wert ist true wenn der SSL-Server ein Client-Zertifikat angefordert hat.

Zugriff nur auf SSL_OPEN, SSL_ALERT, oder SSL_RENEGOTIATE Ereignisse; andernfalls tritt ein Fehler auf.

clientExtensions: ***Reihe | Null***

Ein Array von Client-Erweiterungsobjekten, die die folgenden Eigenschaften enthalten:

id: ***Zahl***

Die ID-Nummer der SSL-Client-Erweiterung.

length: ***Zahl***

Die volle Länge der SSL-Clienterweiterung, ausgedrückt in Byte.

Hinweis: Eine Erweiterung wird möglicherweise gekürzt, wenn die Länge die maximale Größe überschreitet. Die Standardeinstellung ist 512 Byte. Die Kürzung wurde vorgenommen, wenn der Wert dieser Eigenschaft kleiner ist als der Puffer, der vom `getClientExtensionData()` Methode.

`name:` ***Schnur***

Der Name der SSL-Client-Erweiterung, falls bekannt. Andernfalls gibt der Wert an , dass die Erweiterung unbekannt ist. Die Tabelle der bekannten SSL-Erweiterungen finden Sie in der Abschnitt Methoden.

Zugriff nur auf `SSL_OPEN` oder `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`clientHelloVersion:` ***Zahl***

Die vom Client im Client-Hello-Paket angegebene SSL-Version.

`clientL2Bytes:` ***Zahl***

Die Zahl der L2 Byte, die der Client seit dem letzten Mal gesendet hat `SSL_RECORD` Ereignis.

Zugriff nur auf `SSL_RECORD` oder `SSL_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`clientPkts:` ***Zahl***

Die Anzahl der Pakete, die der Client seit dem letzten gesendet hat `SSL_RECORD` Ereignis.

Zugriff nur auf `SSL_RECORD` oder `SSL_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`clientSessionId:` ***Schnur***

Die Client-Sitzungs-ID als Byte-Array, das als Zeichenfolge kodiert ist.

`clientZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster, die der Client seit dem letzten gesendet hat `SSL_RECORD` Ereignis.

Zugriff nur auf `SSL_RECORD` oder `SSL_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`contentType:` ***Schnur***

Der Inhaltstyp für den aktuellen Datensatz.

Zugriff nur auf `SSL_RECORD` Ereignisse; andernfalls tritt ein Fehler auf.

`encryptionProtocol:` ***Schnur***

Die SSL-Protokollversion, mit der die Transaktion verschlüsselt ist.

`handshakeTime:` ***Zahl***

Die Zeit, die zum Aushandeln der SSL-Verbindung erforderlich ist, ausgedrückt in Millisekunden. Insbesondere die Zeitspanne zwischen dem Client eines `Kunde Hello` Nachricht und der Server sendet `ChangeCipherSpec` Werte wie in RFC 2246 angegeben.

Zugriff nur auf `SSL_OPEN` oder `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`heartbeatPayloadLength:` ***Zahl***

Der Wert des Nutzdatenlängenfeldes der HeartbeatMessage-Datenstruktur, wie in RFC 6520 angegeben.

Zugriff nur auf `SSL_HEARTBEAT` Ereignisse; andernfalls tritt ein Fehler auf.

`heartbeatType:` ***Zahl***

Die numerische Darstellung des HeartbeatMessageType-Felds der HeartBeartMessage-Datenstruktur, wie in RFC 6520 spezifiziert. Gültige Werte sind `SSL.HEARTBEAT_TYPE_REQUEST (1)`, `SSL.HEARTBEAT_TYPE_RESPONSE (2)`, oder `SSL.HEARTBEAT_TYPE_UNKNOWN (255)`.

Zugriff nur auf `SSL_HEARTBEAT` Ereignisse; andernfalls tritt ein Fehler auf.

`host:` ***Schnur | Null***

Die SSL Server Name Indication (SNI), falls verfügbar.

Zugriff nur auf `SSL_OPEN` oder `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`isAborted:` ***Boolesch***

Der Wert ist `true` wenn die SSL-Sitzung abgebrochen wird.

Zugriff nur auf `SSL_CLOSE`, `SSL_OPEN`, und `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`isCompressed:` ***Boolesch***

Der Wert ist `true` wenn der SSL-Datensatz komprimiert ist.

`isDecrypted:` ***Boolesch***

Der Wert ist `true` ob das ExtraHop-System die Transaktion sicher entschlüsselt und analysiert hat. Die Analyse des entschlüsselten Datenverkehrs kann komplexe Bedrohungen aufdecken, die sich im verschlüsselten Verkehr verstecken.

`isEncrypted:` ***Boolesch***

Der Wert ist `true` wenn die SSL-Verbindung verschlüsselt ist.

`isResumed:` ***Boolesch***

Der Wert ist `true` wenn die Verbindung von einer bestehenden SSL-Sitzung wieder aufgenommen wird und es sich nicht um eine neue SSL-Sitzung handelt.

Zugriff nur auf `SSL_OPEN`, `SSL_CLOSE`, `SSL_ALERT`, `SSL_HEARTBEAT`, oder `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`isStartTLS:` ***Boolesch***

Der Wert ist `true` wenn die Aushandlung der SSL-Sitzung durch den STARTTLS-Mechanismus des Protokoll initiiert wurde.

Zugriff nur auf `SSL_OPEN`, `SSL_CLOSE`, `SSL_ALERT`, `SSL_HEARTBEAT`, oder `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`isV2ClientHello:` ***Boolesch***

Der Wert ist `true` wenn der Hello-Datensatz SSLv2 entspricht.

`isWeakCipherSuite:` ***Boolesch***

Der Wert ist `true` wenn die Verschlüsselungssuite, die die SSL-Sitzung verschlüsselt, als schwach angesehen wird . Die Verschlüsselungssammlungen NULL, Anonym und EXPORT gelten als schwach, ebenso wie Suiten, die mit CBC, DES, 3DES, MD5 oder RC4 verschlüsseln.

Zugriff nur auf `SSL_OPEN`, `SSL_CLOSE`, `SSL_ALERT`, `SSL_HEARTBEAT`, oder `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`ja3Text:` ***Schnur | Null***

Die vollständige JA3-Zeichenfolge für den Client, einschließlich der Client-Hello-SSL-Version, akzeptierter Chiffren, SSL-Erweiterungen, elliptischer Kurven und elliptischer Kurvenformate.

`ja3Hash:` ***Schnur | Null***

Der MD5-Hash der JA3-Zeichenfolge für den Client.

`ja3sText:` ***Schnur | Null***

Die vollständige JA3S-Zeichenfolge für den Server, einschließlich der Server-Hello SSL-Version, akzeptierter Chiffren und SSL-Erweiterungen.

`ja3sHash:` ***Schnur***

Der MD5-Hash der JA3S-Zeichenfolge für den Server.

`privateKeyId:` ***Schnur | Null***

Die Zeichenketten-ID, die dem privaten Schlüssel zugeordnet ist, wenn das ExtraHop-System den SSL-Verkehr entschlüsselt. Der Wert ist `null` wenn das ExtraHop-System den SSL-Verkehr nicht entschlüsselt.

Um die private Schlüssel-ID in den Verwaltungseinstellungen zu finden, klicken Sie auf **Erfassen** von der Konfiguration des Systems Abschnitt, klicken Sie **SSL-Entschlüsselung**, und klicken Sie dann auf ein Zertifikat. Das Popup-Fenster zeigt alle Identifikatoren für das Zertifikat an.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `SSL.commitRecord()` auf einem `SSL_OPEN`, `SSL_CLOSE`, `SSL_ALERT`, `SSL_HEARTBEAT`, oder `SSL_RENEGOTIATE` Ereignis.

Das Ereignis, bei dem die Methode aufgerufen wurde, bestimmt, welche Eigenschaften das Standard-Datensatzobjekt enthalten kann, wie in der folgenden Tabelle dargestellt:

Ereignis	Verfügbare Immobilien
`SSL_ALERT`	• `alertCode` • `alertLevel` • `certificateFingerprint` • `certificateIsSelfSigned` • `certificateIssuer` • `certificateKeySize` • `certificateNotAfter` • `certificateNotBefore` • `certificateSignatureAlgorithm` • `certificateSubject` • `cipherSuite` • `clientAddr` • `clientBytes` • `clientCertificateRequested` • `clientIsExternal` • `clientL2Bytes` • `clientPkts` • `clientPort` • `clientRTO` • `clientZeroWnd` • `isCompressed` • `isWeakCipherSuite` • `proto` • `receiverIsExternal` • `reqBytes` • `reqL2Bytes` • `reqPkts` • `reqRTO` • `rspBytes` • `rspL2Bytes` • `rspPkts` • `rspRTO` • `senderIsExternal` • `serverAddr` • `serverBytes` • `serverIsExternal` • `serverL2Bytes` • `serverPkts` • `serverPort` • `serverRTO` • `serverZeroWnd`

Ereignis	Verfügbare Immobilien
	• version
SSL_CLOSE	• certificateIsSelfSigned • certificateIssuer • certificateFingerprint • certificateKeySize • certificateNotAfter • certificateNotBefore • certificateSignatureAlgorithm • certificateSubject • cipherSuite • clientAddr • clientBytes • clientIsExternal • clientL2Bytes • clientPkts • clientPort • clientRTO • clientZeroWnd • isAborted • isCompressed • isWeakCipherSuite • proto • receiverIsExternal • reqBytes • reqPkts • reqL2Bytes • reqRTO • rspBytes • rspL2Bytes • rspPkts • rspRTO • senderIsExternal • serverAddr • serverBytes • serverIsExternal • serverL2Bytes • serverPkts • serverPort • serverRTO • serverZeroWnd • version
SSL_HEARTBEAT	• certificateFingerprint • certificateIssuer • certificateKeySize • certificateNotAfter • certificateNotBefore • certificateSignatureAlgorithm

Ereignis	Verfügbare Immobilien
	• certificateSubject • cipherSuite • clientIsExternal • clientZeroWnd • heartbeatPayloadLength • heartbeatType • isCompressed • receiverIsExternal • senderIsExternal • serverIsExternal • serverZeroWnd • version
SSL_OPEN	• certificateFingerprint • certificateIsSelfSigned • certificateIssuer • certificateKeySize • certificateNotAfter • certificateNotBefore • certificateSignatureAlgorithm • certificateSubject • certificateSubjectAlternativeNames • cipherSuite • clientAddr • clientAlpn • clientBytes • clientCertificateRequested • clientIsExternal • clientL2Bytes • clientPkts • clientPort • clientRTO • clientZeroWnd • handshakeTime • host • isAborted • isCompressed • isRenegotiate • isWeakCipherSuite • ja3Hash • ja3sHash • proto • receiverIsExternal • reqBytes • reqL2Bytes • reqPkts • reqRTO • rspBytes • rspL2Bytes

Ereignis	Verfügbare Immobilien
	• rspPkts • rspRTO • senderIsExternal • serverAddr • serverAlpn • serverBytes • serverIsExternal • serverL2Bytes • serverPkts • serverPort • serverRTO • serverZeroWnd • version
SSL_RENEGOTIATE **Hinweis:** Das SSL_OPEN Das Datensatzformat wird auf Datensätze angewendet, die bei diesem Ereignis übertragen wurden.	• certificateFingerprint • certificateKeySize • certificateNotAfter • certificateNotBefore • certificateSignatureAlgorithm • certificateSubject • cipherSuite • clientAlpn • clientIsExternal • handshakeTime • host • isAborted • isCompressed • receiverIsExternal • senderIsExternal • serverAlpn • serverIsExternal • version

`recordLength:` ***Zahl***

Der Wert des Längenfeldes von `TLSPlaintext`, `TLSCompressed`, und `TLSCiphertext` Datenstrukturen wie in RFC 5246 spezifiziert.

Zugriff nur auf SSL_RECORD, SSL_ALERT, oder SSL_HEARTBEAT Ereignisse; andernfalls tritt ein Fehler auf.

`recordType:` ***Zahl***

Die numerische Darstellung des Typfeldes des `TLSPlaintext`, `TLSCompressed`, und `TLSCiphertext` Datenstrukturen wie in RFC 5246 spezifiziert.

Zugriff nur auf SSL_RECORD, SSL_ALERT, und SSL_HEARTBEAT Ereignisse; andernfalls tritt ein Fehler auf.

`roundTripTime:` ***Zahl***

Die mittlere Roundtrip-Zeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Samples gibt.

Zugriff nur auf SSL_RECORD oder SSL_CLOSE Ereignisse; andernfalls tritt ein Fehler auf.

`serverExtensions:` ***Reihe | Null***

Ein Array von Servererweiterungsobjekten, die die folgenden Eigenschaften enthalten:

`id:` ***Zahl***

Die ID-Nummer der SSL-Servererweiterung.

`length:` ***Zahl***

Die volle Länge der SSL-Servererweiterung, ausgedrückt in Byte.

Hinweis: Eine Erweiterung wird möglicherweise gekürzt, wenn die Länge die maximale Größe überschreitet. Die Standardeinstellung ist 512 Byte. Die Kürzung wurde vorgenommen, wenn der Wert dieser Eigenschaft kleiner ist als der Puffer, der vom `getClientExtensionData()` Methode.

`name:` ***Schnur***

Der Name der SSL-Servererweiterung, falls bekannt. Andernfalls gibt der Wert an , dass die Erweiterung unbekannt ist. Die Tabelle der bekannten SSL-Erweiterungen finden Sie in der Abschnitt Methoden.

Zugriff nur auf `SSL_OPEN` oder `SSL_RENEGOTIATE` Ereignisse; andernfalls tritt ein Fehler auf.

`serverBytes:` ***Zahl***

Die Anzahl der Byte, die der Server seit dem letzten Mal gesendet hat `SSL_RECORD` Ereignis.

Zugriff nur auf `SSL_RECORD` oder `SSL_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`serverHelloVersion:` ***Zahl***

Die vom Server im Server-Hello-Paket angegebene SSL-Version.

`serverL2Bytes:` ***Zahl***

Die Zahl der L2 Byte, die der Server seit dem letzten Mal gesendet hat `SSL_RECORD` Ereignis.

Zugriff nur auf `SSL_RECORD` oder `SSL_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`serverPkts:` ***Zahl***

Die Anzahl der Pakete, die der Server seit dem letzten gesendet hat `SSL_RECORD` Ereignis.

Zugriff nur auf `SSL_RECORD` oder `SSL_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`serverSessionId:` ***Schnur***

Das Bytearray der Serversitz-ID, kodiert als Zeichenfolge.

`serverZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster, die der Server seit dem letzten gesendet hat `SSL_RECORD` Ereignis.

Zugriff nur auf `SSL_RECORD` oder `SSL_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`startTLSProtocol:` ***Schnur | Null***

Das Protokoll, von dem aus der Client einen STARTTLS-Befehl gesendet hat.

`version:` ***Zahl***

Die SSL-Protokollversion mit der hexadezimalen RFC-Versionsnummer, ausgedrückt als Dezimalzahl.

Version	Hex	Dezimal
SSLv2	0x200	2
SSLv3	0x300	768
TLS 1.0	0x301	769
TLS 1.1	0x302	770
TLS 1.2	0x303	771
TLS 1.3	0x304	772

TCP

Die TCP Mit dieser Klasse können Sie auf Eigenschaften zugreifen und Metriken von TCP-Ereignissen abrufen und von FLOW_TICK und FLOW_TURN Ereignisse.

Die FLOW_TICK und FLOW_TURN Ereignisse sind definiert in Flow Abschnitt.

Ereignisse

TCP_CLOSE

Wird ausgeführt, wenn die TCP-Verbindung geschlossen, abgelaufen oder abgebrochen wird.

TCP_OPEN

Wird ausgeführt, wenn die TCP-Verbindung zum ersten Mal vollständig hergestellt ist.

Die FLOW_CLASSIFY Die Ereignis läuft nach dem TCP_OPEN Ereignis zur Bestimmung der L7 Protokoll des TCP-Flusses.

Hinweis: Wenn eine TCP-Verbindung für einen längeren Zeitraum unterbrochen wird, wird das TCP_OPEN-Ereignis erneut ausgeführt, wenn die Verbindung wieder aufgenommen wird. Die folgenden TCP-Eigenschaften und -Methoden sind Null, wenn das Ereignis für eine wiederhergestellte Verbindung ausgeführt wird:

- getOption
- handshakeTime
- hasECNEcho
- hasECNEcho1
- hasECNEcho2
- initRcvWndSize
- initRcvWndSize1
- initRcvWndSize2
- initSeqNum
- initSeqNum1
- initSeqNum2
- options
- options1
- options2

TCP_PAYLOAD

Wird ausgeführt, wenn die Nutzlast den im zugehörigen Auslöser konfigurierten Kriterien entspricht.

Abhängig von Flow , die TCP-Nutzlast kann in den folgenden Eigenschaften gefunden werden:

- Flow.client.payload
- Flow.payload1
- Flow.payload2
- Flow.receiver.payload
- Flow.sender.payload
- Flow.server.payload

Zusätzliche Payload-Optionen sind verfügbar, wenn Sie einen Auslöser erstellen, der bei diesem Ereignis ausgeführt wird. siehe Erweiterte Trigger-Optionen für weitere Informationen.

Methoden

`getOption(kind:` ***Zahl*** `):` ***Objekt*** | ***Null***

Gibt ein TCP-Optionsobjekt zurück, das der angegebenen Optionsart entspricht. Eine Liste der gültigen Optionstypen finden Sie unter TCP-Optionen. Geben Sie den TCP-Client oder den TCP-Server in der Syntax an, z. B. `TCP.client.getOption(1)` oder `TCP.server.getOption(1)`.

Gilt nur für `TCP_OPEN` Ereignisse.

Eigenschaften

`handshakeTime:` ***Zahl***

Die Zeit, die für die Aushandlung der TCP-Verbindung benötigt wird, ausgedrückt in Millisekunden.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`hasECNEcho:` ***Boolescher Wert***

Der Wert ist `true` wenn das ECN-Flag während des Drei-Wege-Handshakes auf einem Gerät gesetzt ist. Geben Sie den TCP-Client oder den TCP-Server in der Syntax an, z. B. `TCP.client.hasECNEcho` oder `TCP.server.hasECNEcho`.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`hasECNEcho1:` ***Boolescher Wert***

Der Wert ist `true` wenn das ECN-Flag während des Drei-Wege-Handshakes gesetzt ist, der einem von zwei Geräten in der Verbindung zugeordnet ist; das andere Gerät wird dargestellt durch `hasECNEcho2`. Das Gerät wird repräsentiert durch `hasECNEcho1` bleibt für die Verbindung konsistent.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`hasECNEcho2:` ***Boolescher Wert***

Der Wert ist `true` wenn das ECN-Flag während des Drei-Wege-Handshakes gesetzt ist, der einem von zwei Geräten in der Verbindung zugeordnet ist; das andere Gerät wird dargestellt durch `hasECNEcho1`. Das Gerät wird repräsentiert durch `hasECNEcho2` bleibt für die Verbindung konsistent.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`initRcvWndSize:` ***Zahl***

Die anfängliche Größe des TCP-Schiebefensters auf einem Gerät, die während des Drei-Wege-Handshakes ausgehandelt wurde. Geben Sie den TCP-Client oder den TCP-Server in der Syntax an, z. B. `TCP.client.initRcvWndSize` oder `TCP.server.initRcvWndSize`.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`initRcvWndSize1:` ***Zahl***

Die anfängliche Größe des TCP-Schiebefensters, das während des Drei-Wege-Handshakes ausgehandelt wurde, der einem von zwei Geräten in der Verbindung zugeordnet ist; das andere Gerät wird dargestellt durch `initRcvWndSize2`. Das Gerät wird repräsentiert durch `initRcvWndSize1` bleibt für die Verbindung konsistent.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`initRcvWndSize2:` ***Zahl***

Die anfängliche Größe des TCP-Schiebefensters, das während des Drei-Wege-Handshakes ausgehandelt wurde, der einem von zwei Geräten in der Verbindung zugeordnet ist; das andere Gerät wird dargestellt durch `initRcvWndSize1`. Das Gerät wird repräsentiert durch `initRcvWndSize2` bleibt für die Verbindung konsistent.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`initSeqNum:` ***Zahl***

Die erste Sequenznummer, die von einem Gerät während des Drei-Wege-Handshakes gesendet wurde. Geben Sie den TCP-Client oder den TCP-Server in der Syntax an, z. B. `TCP.client.initSeqNum` oder `TCP.server.initSeqNum`.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`initSeqNum1:` ***Zahl***

Die erste Sequenznummer während des Dreiwege-Handshakes, die einem von zwei Geräten in der Verbindung zugeordnet ist; das andere Gerät wird dargestellt durch `initSeqNum2`. Das Gerät wird repräsentiert durch `initSeqNum1` bleibt für die Verbindung konsistent.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`initSeqNum2:` ***Zahl***

Die erste Sequenznummer während des Dreiwege-Handshakes, die einem von zwei Geräten in der Verbindung zugeordnet ist; das andere Gerät wird dargestellt durch `initSeqNum1`. Das Gerät wird repräsentiert durch `initSeqNum2` bleibt für die Verbindung konsistent.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`isAborted:` ***Boolescher Wert***

Der Wert ist `true` wenn ein TCP-Fluss durch einen TCP-Reset (RST) abgebrochen wurde, bevor die Verbindung geschlossen wurde. Der Fluss kann durch ein Gerät unterbrochen werden. Geben Sie gegebenenfalls die Geräterolle in der Syntax an, z. B. `TCP.client.isAborted` oder `TCP.server.isAborted`.

Dieser Zustand kann bei jedem TCP-Ereignis und bei allen betroffenen L7-Ereignissen erkannt werden (z. B. `HTTP_REQUEST` oder `DB_RESPONSE`).

Hinweis:
- Ein L4 Ein Abbruch tritt auf, wenn eine TCP-Verbindung mit einem RST statt mit einem ordnungsgemäßen Herunterfahren geschlossen wird.
- Ein L7 Ein Antwortabbruch tritt auf, wenn eine Verbindung während einer Antwort geschlossen wird. Dies kann auf ein RST, ein ordnungsgemäßes FIN-Shutdown oder ein Ablaufdatum zurückzuführen sein.
- Ein L7-Anforderungsabbruch tritt auf, wenn eine Verbindung während einer Anfrage geschlossen wird. Dies kann auch an einem RST, einem ordnungsgemäßen FIN-Shutdown oder einem Ablauf liegen.

`isExpired:` ***Boolescher Wert***

Der Wert ist `true` wenn die TCP-Verbindung zum Zeitpunkt des Ereignisses abgelaufen ist. Geben Sie gegebenenfalls den TCP-Client oder den TCP-Server in der Syntax an, z. B. `TCP.client.isExpired` oder `TCP.server.isExpired`.

Zugriff nur auf `TCP_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isReset:` ***Boolescher Wert***

Der Wert ist `true` wenn ein TCP-Reset (RST) beobachtet wurde, während die Verbindung gerade geschlossen wurde.

`nagleDelay:` ***Zahl***

Die Anzahl der Nagle-Verzögerungen, die einem Gerät im Fluss zugeordnet sind. Geben Sie den TCP-Client oder den TCP-Server in der Syntax an, z. B. `TCP.client.nagleDelay` oder `TCP.server.nagleDelay`.

Zugriff nur auf `FLOW_TICK` und `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`nagleDelay1:` ***Zahl***

Die Anzahl der Nagle-Verzögerungen, die einem von zwei Geräten im Fluss zugeordnet sind; das andere Gerät wird dargestellt durch `nagleDelay1`. Das Gerät wird repräsentiert durch `nagleDelay2` bleibt für die Verbindung konsistent.

Zugriff nur auf `FLOW_TICK` und `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`nagleDelay1:` ***Zahl***

Die Anzahl der Nagle-Verzögerungen, die einem von zwei Geräten im Fluss zugeordnet sind; das andere Gerät wird dargestellt durch `nagleDelay2`. Das Gerät wird repräsentiert durch `nagleDelay1` bleibt für die Verbindung konsistent.

Zugriff nur auf `FLOW_TICK` und `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`options:` ***Reihe***

Ein Array von Objekten, die die TCP-Optionen eines Gerät in den ersten Handshake-Paketen darstellen. Geben Sie das TCP an Client oder der TCP-Server in der Syntax — zum Beispiel `TCP.client.options` oder `TCP.server.options`. Weitere Informationen finden Sie im Abschnitt TCP-Optionen weiter unten.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`options1:` ***Reihe***

Eine Reihe von Optionen, die die TCP-Optionen in den ersten Handshake-Paketen darstellen, die einem von zwei Geräten in der Verbindung zugeordnet sind; das andere Gerät wird dargestellt durch `options2`. Das Gerät wird repräsentiert durch `options1` bleibt für die Verbindung konsistent. Weitere Informationen finden Sie im Abschnitt TCP-Optionen weiter unten.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`options2:` ***Reihe***

Eine Reihe von Optionen, die die TCP-Optionen in den ersten Handshake-Paketen darstellen, die einem von zwei Geräten in der Verbindung zugeordnet sind; das andere Gerät wird dargestellt durch `options1`. Das Gerät wird repräsentiert durch `options2` bleibt für die Verbindung konsistent. Weitere Informationen finden Sie im Abschnitt TCP-Optionen weiter unten.

Zugriff nur auf `TCP_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`overlapSegments:` ***Zahl***

Die Anzahl der nicht identischen TCP-Segmente, die von einem Gerät im Fluss übertragen werden, wobei zwei oder mehr TCP-Segmente Daten für denselben Teil des Fluss enthalten. Geben Sie den TCP-Client oder den TCP-Server in der Syntax an, z. B. `TCP.client.overlapSegments` oder `TCP.server.overlapSegments`.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`overlapSegments1:` ***Zahl***

Die Anzahl der nicht identischen TCP-Segmente, bei denen zwei oder mehr Segmente Daten für denselben Teil des Datenflusses enthalten. Die TCP-Segmente werden von einem von zwei Geräten im Fluss übertragen; das andere Gerät wird dargestellt durch `overlapSegments2`. Das Gerät wird repräsentiert durch `overlapSegments1` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`overlapSegments2:` ***Zahl***

Die Anzahl der nicht identischen TCP-Segmente, bei denen zwei oder mehr Segmente Daten für denselben Teil des Datenflusses enthalten. Die TCP-Segmente werden von einem von zwei Geräten im Fluss übertragen; das andere Gerät wird dargestellt durch `overlapSegments1`. Das Gerät wird repräsentiert durch `overlapSegments2` bleibt für den Fluss konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`rcvWndThrottle:` ***Zahl***

Die Anzahl der Empfangsfensterdrosseln, die von einem Gerät im Fluss gesendet wurden. Geben Sie den TCP-Client oder den TCP-Server in der Syntax an, z. B. `TCP.client.rcvWndThrottle` oder `TCP.server.rcvWndThrottle`.

Zugriff nur auf `FLOW_TICK` und `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`rcvWndThrottle1:` ***Zahl***

Die Anzahl der Empfangsfenster-Drosselungen, die von einem von zwei Geräten im Fluss gesendet wurden; das andere Gerät wird dargestellt durch `rcvWndThrottle2`. Das Gerät wird repräsentiert durch `rcvWndThrottle1` bleibt für die Verbindung konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`rcvWndThrottle2:` ***Zahl***

Die Anzahl der Empfangsfenster-Drosselungen, die von einem von zwei Geräten im Fluss gesendet wurden; das andere Gerät wird dargestellt durch `rcvWndThrottle1`. Das Gerät wird repräsentiert durch `rcvWndThrottle2` bleibt für die Verbindung konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`retransBytes:` ***Zahl***

Die Anzahl der Byte, die von einem Client- oder Servergerät im Fluss erneut über TCP übertragen wurden. Geben Sie den TCP-Client oder den TCP-Server in der Syntax an, z. B. `TCP.client.retransBytes` oder `TCP.server.retransBytes`.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`retransBytes1:` ***Zahl***

Die Anzahl der Byte, die von einem von zwei Geräten im Fluss über TCP erneut übertragen wurden; das andere Gerät wird dargestellt durch `retransBytes2`. Das Gerät wird repräsentiert durch `retransBytes1` bleibt für die Verbindung konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`retransBytes2:` ***Zahl***

Die Anzahl der Byte, die von einem von zwei Geräten im Fluss über TCP erneut übertragen wurden; das andere Gerät wird dargestellt durch `retransBytes1`. Das Gerät wird repräsentiert durch `retransBytes2` bleibt für die Verbindung konsistent.

Zugriff nur auf `FLOW_TICK` oder `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`zeroWnd:` ***Zahl***

Die Anzahl der Nullfenster, die von einem Gerät im Fluss gesendet wurden. Geben Sie den TCP-Client oder den TCP-Server in der Syntax an, z. B. `TCP.client.zeroWnd` oder `TCP.server.zeroWnd`.

Zugriff nur auf `FLOW_TICK` und `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`zeroWnd1:` ***Zahl***

Die Anzahl der Nullfenster, die von einem von zwei Geräten im Fluss gesendet wurden; das andere Gerät wird dargestellt durch `zeroWnd2`. Das Gerät wird repräsentiert durch `zeroWnd1` bleibt für die Verbindung konsistent.

Zugriff nur auf `FLOW_TICK` und `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

`zeroWnd2:` ***Zahl***

Die Anzahl der Nullfenster, die von einem von zwei Geräten im Fluss gesendet wurden; das andere Gerät wird dargestellt durch `zeroWnd1`. Das Gerät wird repräsentiert durch `zeroWnd2` bleibt für die Verbindung konsistent.

Zugriff nur auf `FLOW_TICK` und `FLOW_TURN` Ereignisse; andernfalls tritt ein Fehler auf.

TCP-Optionen

Alle TCP-Options-Objekte haben die folgenden Eigenschaften:

`kind:` ***Zahl***

Die Typnummer der TCP-Option.

Art Nummer	Bedeutung
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)
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 acknowledgments
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)

Art Nummer	Bedeutung
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)
254	RFC3692-style Experiment 2 (also improperly used for shipping products)

`name:` ***Schnur***

Der Name der TCP-Option.

Die folgende Liste enthält die Namen gängiger TCP-Optionen und ihre spezifischen Eigenschaften:

Maximale Segmentgröße (Name 'mss', Optionstyp 2)

`value:` ***Zahl***

Die maximale Segmentgröße.

Fensterwaage (Name 'wscale', Typ 3)

`value:` ***Zahl***

Der Fensterskalierungsfaktor.

Selektive Bestätigung erlaubt (Name 'sack-permitted', Art 4)

Keine zusätzlichen Eigenschaften. Ihr Vorhandensein weist darauf hin, dass die selektive Bestätigungsoption in der SYN enthalten war.

Zeitstempel (Name 'timestamp', Art 8)

`tsval:` ***Zahl***

Das Feld tsVal für die Option.

`tsecr:` ***Zahl***

Das TSecr-Feld für die Option.

Schnellstart-Antwort (Name 'quickstart-rsp', Art 27)

`rate-request:` ***Zahl***

Die angeforderte Transportrate, ausgedrückt in Byte pro Sekunde.

`ttl-diff:` ***Zahl***

Die TTLDIF.

qs-`nonce:` ***Zahl***

Die QS Nonce.

Akamai-Adresse (Name 'akamai-addr', Art 28)

`value:` ***iPadDR***

Die IP-Adresse des Akamai-Servers.

Benutzer-Timeout (Name 'Benutzer-Timeout', Art 28)

`value:` ***Zahl***

Das Benutzer-Timeout.

Authentifizierung (Name 'tcp-ao', Art 29)

`keyId property:` ***Zahl***

Die Schlüssel-ID für den verwendeten Schlüssel.

`rNextKeyId:` ***Zahl***

Die Schlüssel-ID für die Schlüssel-ID „Als Nächstes empfangen“.

`mac:` ***Puffer***

Der Nachrichtenauthentifizierungscode.

Multipath (Name 'mptcp', Art 30)

`value:` ***Puffer***

Der Multipath-Wert.

Hinweis: Die Optionen für Akamai-Adresse und Benutzer-Timeout unterscheiden sich durch die Länge der Option.

Im Folgenden finden Sie ein Beispiel für TCP-Optionen:

```
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);
   }

}
```

`Telnet`

Die `Telnet` Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen `TELNET_MESSAGE` Ereignisse.

Ereignisse

`TELNET_MESSAGE`

Wird mit einem Telnet-Befehl oder einer Datenzeile aus dem Telnet ausgeführt Client oder Server.

Methoden

`commitRecord():` ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem `TELNET_MESSAGE` Ereignis.

Die Standardeigenschaften, die für das Datensatzobjekt übernommen wurden, finden Sie in `record` Eigentum unten.

Bei integrierten Datensätzen wird jeder eindeutige Datensatz nur einmal festgeschrieben, auch wenn `commitRecord()` Methode wird mehrmals für denselben eindeutigen Datensatz aufgerufen.

Eigenschaften

`command:` ***Schnur***

Der Befehlstyp. Der Wert ist `null` wenn das Ereignis aufgrund einer gesendeten Datenzeile ausgeführt wurde.

Die folgenden Werte sind gültig:

- `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:` ***Schnur***

Eine Zeile mit den Daten, die von der gesendet wurden Client oder Server. Terminal-Escape-Sequenzen und Sonderzeichen werden herausgefiltert. Cursorbewegung und Zeilenbearbeitung werden mit Ausnahme von Rücktastenzeichen nicht simuliert.

`option:` ***Schnur***

Die Option wird ausgehandelt. Der Wert ist `null` wenn die Option ungültig ist. Die folgenden Werte sind gültig:

- `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:` ***Puffer***

Für Optionsunterverhandlungen (der SB-Befehl) werden die rohen, optionsspezifischen Daten gesendet. Der Wert ist `null` wenn der Befehl nicht SB ist.

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `Telnet.commitRecord()` auf einem `TELNET_MESSAGE` Ereignis.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientIsExternal`

- command
- option
- receiverBytes
- receiverIsExternal
- receiverL2Bytes
- recieverPkts
- receiverRTO
- receiverZeroWnd
- roundTripTime
- senderBytes
- senderIsExternal
- senderL2Bytes
- senderPkts
- senderRTO
- senderZeroWnd
- serverIsExternal

`receiverBytes:` ***Zahl***

Die Anzahl der Byte auf Anwendungsebene vom Empfänger.

`receiverL2Bytes:` ***Zahl***

Die Anzahl der L2 Bytes vom Empfänger.

`receiverPkts:` ***Zahl***

Die Anzahl der Pakete vom Empfänger.

`receiverRTO:` ***Zahl***

Die Anzahl der Timeouts bei der erneuten Übertragung (RTOs) vom Empfänger.

`receiverZeroWnd:` ***Zahl***

Die Anzahl der vom Empfänger gesendeten Nullfenster.

`roundTripTime:` ***Zahl***

Die mittlere Umlaufzeit (RTT), ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine RTT-Proben gibt.

`senderBytes:` ***Zahl***

Die Anzahl der Byte auf Anwendungsebene vom Absender.

`senderL2Bytes:` ***Zahl***

Die Anzahl der L2 Bytes vom Absender.

`senderPkts:` ***Zahl***

Die Anzahl der Pakete vom Absender.

`senderRTO:` ***Zahl***

Die Anzahl der Timeouts bei der erneuten Übertragung (RTOs) vom Absender.

`senderZeroWnd:` ***Zahl***

Die Anzahl der vom Absender gesendeten Nullfenster.

`Turn`

`Turn` ist eine Klasse, mit der Sie Metriken speichern und auf Eigenschaften zugreifen können, die auf verfügbar sind `FLOW_TURN` Ereignisse.

Die `FLOW_TURN` Ereignis ist definiert in Flow Abschnitt.

Eigenschaften

`clientBytes:` ***Zahl***

Die Größe der Anfrage, die Client übertragen, ausgedrückt in Byte.

`clientTransferTime:` ***Zahl***

Die Client-Übertragungszeit, ausgedrückt in Millisekunden.

`processingTime:` ***Zahl***

Die Zeit, die zwischen der Übertragung der Anfrage durch den Client an den Server und dem Beginn der Übertragung der Antwort durch den Server an den Client verstrichen ist, ausgedrückt in Millisekunden.

`reqSize:` ***Zahl***

Die Größe der Anforderungsnutzlast, ausgedrückt in Byte.

`reqTransferTime:` ***Zahl***

Die Übertragungszeit der Anfrage, ausgedrückt in Millisekunden. Wenn die Anfrage in einem einzigen Paket enthalten ist, ist die Übertragungszeit Null. Wenn sich die Anfrage über mehrere Pakete erstreckt, ist der Wert die Zeitspanne zwischen der Erkennung des ersten Anforderungspakets und der Erkennung des letzten Paket durch das ExtraHop-System. Ein hoher Wert kann auf eine große Anfrage oder eine Netzwerkverzögerung hinweisen. Der Wert ist `NaN` wenn es keine gültige Messung gibt oder wenn der Zeitpunkt ungültig ist.

`rspSize:` ***Zahl***

Die Größe der Antwortnutzlast, ausgedrückt in Byte.

`rspTransferTime:` ***Zahl***

Die Antwortübertragungszeit, ausgedrückt in Millisekunden. Wenn die Antwort in einem einzigen Paket enthalten ist, ist die Übertragungszeit Null. Wenn sich die Antwort über mehrere Pakete erstreckt, ist der Wert die Zeitspanne zwischen der Erkennung des ersten Antwortpakets und der Erkennung des letzten Paket durch das ExtraHop-System. Ein hoher Wert kann auf eine starke Reaktion oder eine Netzwerkverzögerung hinweisen. Der Wert ist `NaN` wenn es keine gültige Messung gibt oder wenn der Zeitpunkt ungültig ist.

`serverBytes:` ***Zahl***

Die Größe der Antwort, die der Server übertragen hat, ausgedrückt in Byte.

`serverTransferTime:` ***Zahl***

Die Serverübertragungszeit, ausgedrückt in Millisekunden.

`sourceDevice:` ***Gerät***

Das Quellgeräteobjekt. Sehen Sie die Device Klasse für weitere Informationen.

`thinkTime:` ***Zahl***

Die Zeit, die zwischen der Übertragung der Antwort durch den Server an den Client und der Client überträgt eine neue Anfrage an den Server, ausgedrückt in Millisekunden. Der Wert ist `NaN` wenn es keine gültige Messung gibt.

UDP

Die `UDP` Klasse ermöglicht den Zugriff auf Eigenschaften und das Abrufen von Metriken aus UDP-Ereignissen und von `FLOW_TICK` und `FLOW_TURN` Ereignisse.

Die `FLOW_TICK` und `FLOW_TURN` Ereignisse sind definiert in der Flow Abschnitt.

Ereignisse

`UDP_PAYLOAD`

Wird ausgeführt, wenn die Nutzlast den im zugehörigen Auslöser konfigurierten Kriterien entspricht.

Abhängig von der Flow , die UDP-Nutzlast kann in den folgenden Eigenschaften gefunden werden:

- `Flow.client.payload`
- `Flow.payload1`
- `Flow.payload2`
- `Flow.receiver.payload`
- `Flow.sender.payload`
- `Flow.server.payload`

Zusätzliche Payload-Optionen sind verfügbar, wenn Sie einen Auslöser erstellen, der bei diesem Ereignis ausgeführt wird. siehe Erweiterte Trigger-Optionen für weitere Informationen.

`WebSocket`

Die `WebSocket` Mit dieser Klasse können Sie auf Eigenschaften zugreifen `WEBSOCKET_OPEN`, `WEBSOCKET_CLOSE`, und `WEBSOCKET_MESSAGE` Ereignisse.

Ereignisse

`WEBSOCKET_OPEN`

Wird ausgeführt, wenn ein erfolgreicher Handschlag beobachtet wurde.

`WEBSOCKET_CLOSE`

Wird ausgeführt, wenn beide geschlossenen Frames beobachtet werden oder wenn die zugrunde liegende TCP-Verbindung geschlossen ist.

`WEBSOCKET_MESSAGE`

Wird ausgeführt, wenn alle Frames einer Text- oder Binärnachricht beobachtet wurden.

Eigenschaften

`clientBytes:` ***Zahl***

Die Gesamtzahl der Byte, die vom Client während der WebSocket-Sitzung gesendet wurden.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`clientL2Bytes:` ***Zahl***

Die Gesamtzahl der L2 Byte, die vom Client während der WebSockets-Sitzung gesendet wurden.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`clientPkts:` ***Zahl***

Die Gesamtzahl der Pakete, die vom Client während der WebSocket-Sitzung gesendet wurden.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`clientRTO:` ***Zahl***

Die Gesamtzahl der Client Timeouts bei der erneuten Übertragung (RTOs), die während der WebSockets-Sitzung beobachtet wurden.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`clientZeroWnd:` ***Zahl***

Die Anzahl der vom Client gesendeten Nullfenster.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`closeReason:` ***Schnur***

Die im ersten beobachteten Abschluss-Frame enthaltene Textnachricht, die den Grund beschreibt, warum die Verbindung geschlossen wurde. Der Wert ist `null` wenn der Frame diese Information nicht enthält.

Zugriff nur auf `WEBSOCKET_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`host:` ***Schnur***

Der in der Handshake-Anfrage vom Client angegebene Host. Der Wert ist `null` wenn kein Host bereitgestellt wird.

Zugriff nur auf `WEBSOCKET_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`isClientClose:` ***Boolescher Wert***

Der Wert ist `true` wenn der erste Close-Frame vom Client gesendet wurde.

Zugriff nur auf `WEBSOCKET_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`isEncrypted:` ***Boolescher Wert***

Der Wert ist `true` wenn die WebSocket-Verbindung SSL-verschlüsselt ist.

`isServerClose:` ***Boolescher Wert***

Der Wert ist `true` wenn der erste Close-Frame vom Server gesendet wurde. Der Wert ist `false` wenn die Verbindung ungewöhnlich beendet wurde.

Zugriff nur auf `WEBSOCKET_CLOSE` Ereignisse; andernfalls tritt ein Fehler auf.

`msg:` ***Puffer***

Die Puffer Inhalt der WebSocket-Nachricht. Der Puffer ist `null` wenn der Inhalt diese maximale Länge überschritten hat.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`msgType:` ***Schnur***

Der Typ des WebSocket-Nachrichtenrahmens. Gültige Werte sind `TEXT` oder `BINARY`.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`origin:` ***Schnur***

Die Quell-URL, die in der vom Client initiierten Handshake-Anfrage angegeben wurde.

Zugriff nur auf `WEBSOCKET_OPEN` Ereignisse; andernfalls tritt ein Fehler auf.

`serverBytes:` ***Zahl***

Die Gesamtzahl der Byte, die der Server während der WebSockets-Sitzung zurückgegeben hat.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`serverL2Bytes:` ***Zahl***

Die Gesamtzahl der L2 Byte, die vom Server während der WebSockets-Sitzung zurückgegeben wurden.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`serverPkts:` ***Zahl***

Die Gesamtzahl der Pakete, die vom Server während der WebSocket-Sitzung zurückgegeben wurden.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`serverRTO:` ***Zahl***

Die Gesamtzahl der Server Timeouts bei der erneuten Übertragung (RTOs), die während der WebSockets-Sitzung beobachtet wurden.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`serverZeroWnd:` ***Zahl***

Die Anzahl der Nullfenster, die vom Server gesendet wurden.

Zugriff nur auf `WEBSOCKET_MESSAGE` Ereignisse; andernfalls tritt ein Fehler auf.

`statusCode:` ***Zahl***

Der Statuscode, der den Grund darstellt, warum die Verbindung geschlossen wurde, wie in RFC 6455 definiert.

Der Wert ist NO_STATUS_RECVD (1005), wenn der anfängliche Schließframe keinen Statuscode enthält. Der Wert ist NaN wenn die Verbindung ungewöhnlich beendet wurde.

Zugriff nur auf WEBSOCKET_CLOSE Ereignisse; andernfalls tritt ein Fehler auf.

uri: ***Schnur***

Die URI, die in der vom Client initiierten Handshake-Anfrage angegeben wurde.

Zugriff nur auf WEBSOCKET_OPEN Ereignisse; andernfalls tritt ein Fehler auf.

WSMAN

Die WSMAN Mit dieser Klasse können Sie Metriken speichern und auf Eigenschaften zugreifen WSMAN_REQUEST und WSMAN_RESPONSE Ereignisse. Web Services-Management (WSMAN) und die Microsoft-Implementierung Windows Remote Management (WinRM) sind Protokolle, die es Geräten ermöglichen, Verwaltungsinformationen in einem Netzwerk auszutauschen.

Ereignisse

WSMAN_REQUEST

Läuft auf jedem WSMAN_REQUEST vom Gerät verarbeitet.

WSMAN_RESPONSE

Läuft auf jedem WSMAN_RESPONSE vom Gerät verarbeitet.

Methoden

commitRecord(): ***Leere***

Sendet einen Datensatz an den konfigurierten Recordstore auf einem WSMAN_REQUEST oder WSMAN_RESPONSE Ereignis. Die Standardeigenschaften, die für jedes Ereignis übernommen wurden, finden Sie in der Datensatzeigenschaft unten.

Wenn der commitRecord() Methode wird auf einem aufgerufen WSMAN_REQUEST Ereignis, der Datensatz wird erst erstellt, wenn WSMAN_RESPONSE Ereignis läuft. Wenn der commitRecord() Methode wird auf beiden aufgerufen WSMAN_REQUEST und die entsprechenden WSMAN_RESPONSE, es wird nur ein Datensatz für Anfrage und Antwort erstellt, auch wenn commitRecord() Methode wird mehrmals bei denselben Triggerereignissen aufgerufen.

Eigenschaften

encryptionProtocol: ***Schnur***

Das Protokoll, mit dem die Transaktion verschlüsselt ist.

isEncrypted: ***Boolescher Wert***

Der Wert ist true wenn die Transaktion über sicheres HTTP erfolgt.

isDecrypted: ***Boolescher Wert***

Der Wert ist true ob das ExtraHop-System die Transaktion sicher entschlüsselt und analysiert hat. Durch die Analyse des entschlüsselten Datenverkehrs können komplexe Bedrohungen aufgedeckt werden, die sich im verschlüsselten Verkehr verstecken.

operationId: ***Schnur***

Die eindeutige Kennung des Vorgangs.

payload: ***Puffer***

Ein Pufferobjekt, das den XML-Nachrichtenumschlag enthält. Nachrichten, die länger als die maximale Größe sind, werden gekürzt. Die maximale Größe wird im WSMAN-Profil in der laufenden Konfiguration konfiguriert. Das folgende Beispiel für eine laufende Konfiguration ändert die maximale Nachrichtengröße von der Standardeinstellung von 1024 Byte auf 4096:

```
"capture": {
```

```
    "app_proto": {
        "wsman": {
            "payload_max_size": 4096
        }
    }
}
```

`record:` ***Objekt***

Das Datensatzobjekt, das durch einen Aufruf von an den konfigurierten Recordstore gesendet werden kann `WSMAN.commitRecord()`.

Das Standarddatensatzobjekt kann die folgenden Eigenschaften enthalten:

- `clientAddr`
- `clientIsExternal`
- `clientPort`
- `serverAddr`
- `serverPort`
- `proto`
- `timestamp`
- `user`
- `vlan`
- `operationId`
- `receiverIsExternal`
- `reqAction`
- `reqResourceURI`
- `rspAction`
- `rspResourceURI`
- `senderIsExternal`
- `sequenceId`
- `serverIsExternal`

Greifen Sie nur auf das Datensatzobjekt zu `WSMAN_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`reqAction:` ***Schnur***

Die vom Client angeforderte Aktion, die von der in der ResourceURI angegebenen Ressource ausgeführt werden soll.

Zugriff nur auf `WSMAN_REQUEST` Ereignisse; andernfalls tritt ein Fehler auf.

`reqCommand:` ***Schnur | null***

Der in der Anfrage angegebene Befehl. Wenn kein Befehl angegeben ist, ist der Wert Null.

`reqResourceURI:` ***Schnur***

Der Uniform Resource Identifier (URI) der Ressource, die eine Aktion ausführt.

`rspAction:` ***Schnur***

Die Serverantwort auf die vom Client angeforderte Aktion.

Zugriff nur auf `WSMAN_RESPONSE` Ereignisse; andernfalls tritt ein Fehler auf.

`rspResourceURI:` ***Schnur***

Der Uniform Resource Identifier (URI) der Ressource, die eine Aktion ausführt.

`sequenceId:` ***Schnur***

Die Zeichenkettendarstellung einer 64-Bit-Ganzzahl, die eine Nachricht in einer Operation identifiziert.

`user:` ***Schnur***

Der Benutzername des Kontos, das die Anfrage gesendet hat.

Offene Datenstromklassen

Die Trigger-API-Klassen in diesem Abschnitt ermöglichen es Ihnen, Daten an ein Syslog, eine Datenbank oder einen Server eines Drittanbieters zu senden, und zwar über Datenstrom öffnen (ODS), das Sie in den Administrationseinstellungen konfiguriert haben.

Klasse	Beschreibung
Remote.HTTP	Ermöglicht es Ihnen, HTTP-Anforderungsdaten über REST-API-Endpunkte an einen Remoteserver zu senden.
Remote.Kafka	Ermöglicht es Ihnen, Nachrichtendaten an einen entfernten Kafka-Server zu senden.
Remote.MongoDB	Ermöglicht das Einfügen, Entfernen und Aktualisieren von Dokumentensammlungen in eine Remote-Datei MongoDB Datenbank.
Remote.Raw	Ermöglicht es Ihnen, Rohdaten über einen TCP- oder UDP-Port an einen Remoteserver zu senden.
Remote.Syslog	Ermöglicht das Senden von Syslog-Daten an einen Remoteserver.

`Remote.HTTP`

Die `Remote.HTTP` Die Klasse ermöglicht das Einreichen HTTP Daten an ein HTTP anfordern Datenstrom öffnen (ODS) zielt darauf ab und bietet Zugriff auf HTTP-REST-API-Endpunkte.

Sie müssen zuerst ein HTTP-ODS-Ziel in den Administrationseinstellungen konfigurieren, wofür System- und Zugriffsadministrationsrechte erforderlich sind. Informationen zur Konfiguration finden Sie in der Offene Datenströme Abschnitt in der ExtraHop Administrationshandbuch.

Methoden

`delete`

Sendet eine HTTP-REST-Löschanforderung an einen konfigurierten offenen HTTP-Datenstrom.

Syntax:

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

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

Parameter:

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`options:` ***Objekt***

Das Optionsobjekt hat die folgenden Eigenschaften:

`path:` ***Schnur***

Die Zeichenfolge, die den Anforderungspfad angibt.

`headers:` ***Objekt***

Das optionale Objekt, das die Anforderungsheader angibt. Die folgenden Header sind eingeschränkt und führen zu einem Fehler, wenn sie angegeben werden:

- `Connection`
- `Authorization`
- `Proxy-Connection`
- `Content-Length`
- `X-Forwarded-For`
- `Transfer-Encoding`

Hinweis: Autorisierungsheader müssen entweder mit einer integrierten Authentifizierungsmethode wie Amazon Web Services oder über die **Zusätzlicher HTTP-Header** Feld in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen.

In einem Auslöser konfigurierte Header haben Vorrang vor einem Eintrag in **Zusätzlicher HTTP-Header** Feld, das sich in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen. Zum Beispiel, wenn **Zusätzlicher HTTP-Header** Feld spezifiziert `Content-Type: text/plain`, aber ein Trigger-Skript auf demselben ODS-Ziel spezifiziert `Content-Type: application/json`, dann `Content-Type: application/json` ist in der HTTP-Anfrage enthalten.

Sie können die ausgehenden HTTP-Anfragen mit dem Content-Encoding-Header komprimieren.

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

Die folgenden Werte werden für diesen Komprimierungsheader unterstützt:

- `gzip`
- `deflate`

`payload:` ***Schnur*** | ***Puffer***

Die optionale Zeichenfolge oder der Puffer, der die Nutzlast der Anfrage angibt.

Rückgabewerte:

Retouren `true` wenn die Anfrage in der Warteschlange steht, andernfalls kehrt sie zurück `false`.

`get`

Sendet eine HTTP-REST-Abrufanforderung an einen konfigurierten offenen HTTP-Datenstrom.

Syntax:

```
Remote.HTTP("name").get({path: "path", headers: {header:
 "header"},
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

```
Remote.HTTP.get({path: "path", headers: {header: "header"},
 payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

Parameter:

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`options:` ***Objekt***

Das Optionsobjekt hat die folgenden Eigenschaften:

`path:` ***Schnur***

Die Zeichenfolge, die den Anforderungspfad angibt.

`headers:` ***Objekt***

Das optionale Objekt, das die Anforderungsheader angibt. Die folgenden Header sind eingeschränkt und führen zu einem Fehler, wenn sie angegeben werden:

- `Connection`
- `Authorization`
- `Proxy-Connection`
- `Content-Length`
- `X-Forwarded-For`
- `Transfer-Encoding`

Hinweis: Autorisierungsheader müssen entweder mit einer integrierten Authentifizierungsmethode wie Amazon Web Services oder über die **Zusätzlicher HTTP-Header** Feld in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen.

In einem Auslöser konfigurierte Header haben Vorrang vor einem Eintrag in **Zusätzlicher HTTP-Header** Feld, das sich in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen. Zum Beispiel, wenn **Zusätzlicher HTTP-Header** Feld spezifiziert `Content-Type: text/plain`, aber ein Trigger-Skript auf demselben ODS-Ziel spezifiziert `Content-Type: application/json`, dann `Content-Type: application/json` ist in der HTTP-Anfrage enthalten.

Sie können die ausgehenden HTTP-Anfragen mit dem Content-Encoding-Header komprimieren.

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

Die folgenden Werte werden für diesen Komprimierungsheader unterstützt:

- `gzip`
- `deflate`

`payload:` ***Schnur*** | ***Puffer***

Die optionale Zeichenfolge oder der Puffer, der die Nutzlast der Anfrage angibt.

`enableResponseEvent:` ***Boolescher Wert***

Ermöglicht die Ausführung eines Auslöser für die HTTP-Antwort, die vom ODS-Ziel gesendet wird, indem ein REMOTE_RESPONSE-Ereignis erstellt wird.

Wichtig: Die Verarbeitung einer großen Anzahl von HTTP-Antworten kann die Leistung und Effizienz von Auslöser beeinträchtigen. Wir empfehlen, diese Option nur bei Bedarf zu aktivieren.

`context:` ***Objekt*** | ***Schnur*** | ***Zahl*** | ***Boolescher Wert*** | ***null***

Ein optionales Objekt, das an den Auslöser gesendet wird, der auf der HTTP-Antwort vom ODS-Ziel ausgeführt wird. Sie können auf die im Objekt gespeicherten Informationen zugreifen, indem Sie die `Remote.response.context` Eigentum.

Rückgabewerte:

Retouren `true` wenn die Anfrage in der Warteschlange steht, andernfalls kehrt sie zurück `false`.

`patch`

Sendet eine HTTP-REST-Patchanforderung an einen konfigurierten offenen HTTP-Datenstrom.

Syntax:

```
Remote.HTTP("name").patch({path: "path", headers: {header:
 "header"},
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

```
Remote.HTTP.patch({path: "path", headers: {header: "header"},
 payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

Parameter:

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`options:` ***Objekt***

Das Optionsobjekt hat die folgenden Eigenschaften:

`path:` ***Schnur***

Die Zeichenfolge, die den Anforderungspfad angibt.

`headers:` ***Objekt***

Das optionale Objekt, das die Anforderungsheader angibt. Die folgenden Header sind eingeschränkt und führen zu einem Fehler, wenn sie angegeben werden:

- `Connection`
- `Authorization`
- `Proxy-Connection`
- `Content-Length`
- `X-Forwarded-For`
- `Transfer-Encoding`

Hinweis: Autorisierungsheader müssen entweder mit einer integrierten Authentifizierungsmethode wie Amazon Web Services oder über die **Zusätzlicher HTTP-Header** Feld in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen.

In einem Auslöser konfigurierte Header haben Vorrang vor einem Eintrag in **Zusätzlicher HTTP-Header** Feld, das sich in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen. Zum Beispiel, wenn **Zusätzlicher HTTP-Header** Feld spezifiziert `Content-Type: text/plain`, aber ein Trigger-Skript auf demselben ODS-Ziel spezifiziert `Content-Type: application/json`, dann `Content-Type: application/json` ist in der HTTP-Anfrage enthalten.

Sie können die ausgehenden HTTP-Anfragen mit dem Content-Encoding-Header komprimieren.

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

Die folgenden Werte werden für diesen Komprimierungsheader unterstützt:

- `gzip`
- `deflate`

`payload:` ***Schnur*** | ***Puffer***

Die optionale Zeichenfolge oder der Puffer, der die Nutzlast der Anfrage angibt.

`enableResponseEvent:` ***Boolescher Wert***

Ermöglicht die Ausführung eines Auslöser für die HTTP-Antwort, die vom ODS-Ziel gesendet wird, indem ein REMOTE_RESPONSE-Ereignis erstellt wird.

Wichtig: Die Verarbeitung einer großen Anzahl von HTTP-Antworten kann die Leistung und Effizienz von Auslöser beeinträchtigen. Wir empfehlen, diese Option nur bei Bedarf zu aktivieren.

`context:` ***Objekt*** | ***Schnur*** | ***Zahl*** | ***Boolescher Wert*** | ***null***

Ein optionales Objekt, das an den Auslöser gesendet wird, der auf der HTTP-Antwort vom ODS-Ziel ausgeführt wird. Sie können auf die im Objekt gespeicherten Informationen zugreifen, indem Sie die `Remote.response.context` Eigentum.

Rückgabewerte:

Retouren `true` wenn die Anfrage in der Warteschlange steht, andernfalls kehrt sie zurück `false`.

`post`

Sendet eine HTTP-REST-Post-Anfrage an einen konfigurierten offenen HTTP-Datenstrom.

Syntax:

```
Remote.HTTP("name").post({path: "path", headers: {header:
 "header"},
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

```
Remote.HTTP.post({path: "path", headers: {header: "header"},
 payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

Parameter:

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`options:` ***Objekt***

Das Optionsobjekt hat die folgenden Eigenschaften:

`path:` ***Schnur***

Die Zeichenfolge, die den Anforderungspfad angibt.

`headers:` ***Objekt***

Das optionale Objekt, das die Anforderungsheader angibt. Die folgenden Header sind eingeschränkt und führen zu einem Fehler, wenn sie angegeben werden:

- `Connection`
- `Authorization`
- `Proxy-Connection`
- `Content-Length`
- `X-Forwarded-For`
- `Transfer-Encoding`

Hinweis: Autorisierungsheader müssen entweder mit einer integrierten Authentifizierungsmethode wie Amazon Web Services oder über die **Zusätzlicher HTTP-Header** Feld in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen.

In einem Auslöser konfigurierte Header haben Vorrang vor einem Eintrag in **Zusätzlicher HTTP-Header** Feld, das sich in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen. Zum Beispiel, wenn **Zusätzlicher HTTP-Header** Feld spezifiziert `Content-Type: text/plain`, aber ein Trigger-Skript auf demselben ODS-Ziel spezifiziert `Content-Type: application/json`, dann `Content-Type: application/json` ist in der HTTP-Anfrage enthalten.

Sie können die ausgehenden HTTP-Anfragen mit dem Content-Encoding-Header komprimieren.

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

Die folgenden Werte werden für diesen Komprimierungsheader unterstützt:

- `gzip`
- `deflate`

`payload:` ***Schnur | Puffer***

Die optionale Zeichenfolge oder der Puffer, der die Nutzlast der Anfrage angibt.

`enableResponseEvent:` ***Boolescher Wert***

Ermöglicht die Ausführung eines Auslöser für die HTTP-Antwort, die vom ODS-Ziel gesendet wird, indem ein REMOTE_RESPONSE-Ereignis erstellt wird.

Wichtig: Die Verarbeitung einer großen Anzahl von HTTP-Antworten kann die Leistung und Effizienz von Auslöser beeinträchtigen. Wir empfehlen, diese Option nur bei Bedarf zu aktivieren.

`context:` ***Objekt*** | ***Schnur*** | ***Zahl*** | ***Boolescher Wert*** | ***null***

Ein optionales Objekt, das an den Auslöser gesendet wird, der auf der HTTP-Antwort vom ODS-Ziel ausgeführt wird. Sie können auf die im Objekt gespeicherten Informationen zugreifen, indem Sie die `Remote.response.context` Eigentum.

Rückgabewerte:

Retouren `true` wenn die Anfrage in der Warteschlange steht, andernfalls kehrt sie zurück `false`.

`put`

Sendet eine HTTP-REST-Put-Anforderung an einen konfigurierten offenen HTTP-Datenstrom.

Syntax:

```
Remote.HTTP("name").put({path: "path", headers: {header:
 "header"},
```

```
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

```
Remote.HTTP.put({path: "path", headers: {header: "header"},
 payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

Parameter:

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`options:` ***Objekt***

Das Optionsobjekt hat die folgenden Eigenschaften:

`path:` ***Schnur***

Die Zeichenfolge, die den Anforderungspfad angibt.

`headers:` ***Objekt***

Das optionale Objekt, das die Anforderungsheader angibt. Die folgenden Header sind eingeschränkt und führen zu einem Fehler, wenn sie angegeben werden:

- `Connection`
- `Authorization`
- `Proxy-Connection`
- `Content-Length`
- `X-Forwarded-For`
- `Transfer-Encoding`

Hinweis: Autorisierungsheader müssen entweder mit einer integrierten Authentifizierungsmethode wie Amazon Web Services oder über die **Zusätzlicher HTTP-Header** Feld in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen.

In einem Auslöser konfigurierte Header haben Vorrang vor einem Eintrag in **Zusätzlicher HTTP-Header** Feld, das sich in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen. Zum Beispiel, wenn **Zusätzlicher HTTP-Header** Feld spezifiziert `Content-Type: text/plain`, aber ein Trigger-Skript auf demselben ODS-Ziel spezifiziert `Content-Type: application/json`, dann `Content-Type: application/json` ist in der HTTP-Anfrage enthalten.

Sie können die ausgehenden HTTP-Anfragen mit dem Content-Encoding-Header komprimieren.

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

Die folgenden Werte werden für diesen Komprimierungsheader unterstützt:

- `gzip`
- `deflate`

`payload:` ***Schnur*** | ***Puffer***

Die optionale Zeichenfolge oder der Puffer, der die Nutzlast der Anfrage angibt.

`enableResponseEvent:` ***Boolescher Wert***

Ermöglicht die Ausführung eines Auslöser für die HTTP-Antwort, die vom ODS-Ziel gesendet wird, indem ein REMOTE_RESPONSE-Ereignis erstellt wird.

Wichtig: Die Verarbeitung einer großen Anzahl von HTTP-Antworten kann die Leistung und Effizienz von Auslöser beeinträchtigen. Wir empfehlen, diese Option nur bei Bedarf zu aktivieren.

`context:` ***Objekt*** | ***Schnur*** | ***Zahl*** | ***Boolescher Wert*** | ***null***

Ein optionales Objekt, das an den Auslöser gesendet wird, der auf der HTTP-Antwort vom ODS-Ziel ausgeführt wird. Sie können auf die im Objekt gespeicherten Informationen zugreifen, indem Sie die `Remote.response.context` Eigentum.

Rückgabewerte:

Retouren `true` wenn die Anfrage in der Warteschlange steht, andernfalls kehrt sie zurück `false`.

`request`

Sendet eine HTTP-REST-Anfrage an einen konfigurierten offenen HTTP-Datenstrom.

Syntax:

```
Remote.HTTP("name").request("method", {path: "path", headers:
 {header: "header"},
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

```
Remote.HTTP.request("method", {path: "path", headers: {header:
 "header"},
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

Parameter:

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`method:` ***Schnur***

Zeichenfolge, die die HTTP-Methode angibt.

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

`options:` ***Objekt***

Das Optionsobjekt hat die folgenden Eigenschaften:

`path:` ***Schnur***

Die Zeichenfolge, die den Anforderungspfad angibt.

`headers:` ***Objekt***

Das optionale Objekt, das die Anforderungsheader angibt. Die folgenden Header sind eingeschränkt und führen zu einem Fehler, wenn sie angegeben werden:

- `Connection`

- `Authorization`
- `Proxy-Connection`
- `Content-Length`
- `X-Forwarded-For`
- `Transfer-Encoding`

Hinweis: Autorisierungsheader müssen entweder mit einer integrierten Authentifizierungsmethode wie Amazon Web Services oder über die **Zusätzlicher HTTP-Header** Feld in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen.

In einem Auslöser konfigurierte Header haben Vorrang vor einem Eintrag in **Zusätzlicher HTTP-Header** Feld, das sich in der Offene Datenströme Konfigurationsfenster in den Administrationseinstellungen. Zum Beispiel, wenn **Zusätzlicher HTTP-Header** Feld spezifiziert `Content-Type: text/plain`, aber ein Trigger-Skript auf demselben ODS-Ziel spezifiziert `Content-Type: application/json`, dann `Content-Type: application/json` ist in der HTTP-Anfrage enthalten.

Sie können die ausgehenden HTTP-Anfragen mit dem Content-Encoding-Header komprimieren.

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

Die folgenden Werte werden für diesen Komprimierungsheader unterstützt:

- `gzip`
- `deflate`

`payload:` ***Schnur*** | ***Puffer***

Die optionale Zeichenfolge oder der Puffer, der die Nutzlast der Anfrage angibt.

`enableResponseEvent:` ***Boolescher Wert***

Ermöglicht die Ausführung eines Auslöser für die HTTP-Antwort, die vom ODS-Ziel gesendet wird, indem ein REMOTE_RESPONSE-Ereignis erstellt wird.

Wichtig: Die Verarbeitung einer großen Anzahl von HTTP-Antworten kann die Leistung und Effizienz von Auslöser beeinträchtigen. Wir empfehlen, diese Option nur bei Bedarf zu aktivieren.

`context:` ***Objekt*** | ***Schnur*** | ***Zahl*** | ***Boolescher Wert*** | ***null***

Ein optionales Objekt, das an den Auslöser gesendet wird, der auf der HTTP-Antwort vom ODS-Ziel ausgeführt wird. Sie können auf die im Objekt gespeicherten Informationen zugreifen, indem Sie die `Remote.response.context` Eigentum.

Rückgabewerte:

Retouren `true` wenn die Anfrage in der Warteschlange steht, andernfalls kehrt sie zurück `false`.

Hilfsmethoden

Die folgenden Hilfsmethoden sind für gängige HTTP-Methoden verfügbar.

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

Syntax:

```
Remote.HTTP("name").delete({path: "path", headers: {header: "header"},
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

```
Remote.HTTP.delete({path: "path", headers: {header: "header"}, payload:
 "payload", enableResponseEvent: "enableResponseEvent", context:
 "context"})
```

```
Remote.HTTP("name").get({path: "path", headers: {header: "header"},
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

```
Remote.HTTP.get({path: "path", headers: {header: "header"}, payload:
 "payload", enableResponseEvent: "enableResponseEvent", context:
 "context"})
```

```
Remote.HTTP("name").patch({path: "path", headers: {header: "header"},
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

```
Remote.HTTP.patch({path: "path", headers: {header: "header"}, payload:
 "payload", enableResponseEvent: "enableResponseEvent", context:
 "context"})
```

```
Remote.HTTP("name").post({path: "path", headers: {header: "header"},
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

```
Remote.HTTP.post({path: "path", headers: {header: "header"}, payload:
 "payload", enableResponseEvent: "enableResponseEvent", context:
 "context"})
```

```
Remote.HTTP("name").put({path: "path", headers: {header: "header"},
payload: "payload", enableResponseEvent: "enableResponseEvent",
 context: "context"})
```

```
Remote.HTTP.put({path: "path", headers: {header: "header"}, payload:
 "payload", enableResponseEvent: "enableResponseEvent", context:
 "context"})
```

Rückgabewerte:

Retouren `true` wenn die Anfrage in der Warteschlange steht, andernfalls kehrt sie zurück `false`.

Beispiele

`HTTP GET`

Im folgenden Beispiel wird eine HTTP-GET-Anfrage an die HTTP-Konfiguration namens „my_destination“ und ein Pfad ausgegeben, der der URI ist, einschließlich Abfragezeichenfolgenvariablen, an den die Anfrage gesendet werden soll.

```
Remote.HTTP("my_destination").get( { path: "/?
example=example1&example2=my_data" } );
```

`HTTP POST`

Im folgenden Beispiel wird eine HTTP-POST-Anfrage an die HTTP-Konfiguration namens „my_destination“ ausgegeben, der Pfad, der die URI ist, an die die Anfrage gesendet werden soll, und eine Nutzlast. Die Nutzlast kann aus Daten bestehen, die denen eines HTTP ähneln Client würde senden, einen JSON-Blob, XML oder was auch immer Sie senden möchten.

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

Benutzerdefinierte HTTP-Header

Das folgende Beispiel definiert ein Javascript-Objekt mit Schlüsseln, um die Header-Namen und ihre entsprechenden Werte darzustellen und diese in einem Aufruf als Wert für den Header-Schlüssel bereitzustellen.

```
var my_json = { example: "my_data", example1: 42, example2: false };
var headers = { "Content-Type": "application/json" };
Remote.HTTP("my_destination").post( { path: "/", headers: headers,
 payload:
JSON.stringify(my_json) });
```

Beispiele für Trigger

- Beispiel: Daten mit Remote.http an Elasticsearch senden
- Beispiel: Senden Sie Daten mit Remote.Http an Azure

`Remote.Kafka`

Die `Remote.Kafka` Klasse ermöglicht es Ihnen, Nachrichtendaten über einen Kafka an einen Kafka-Server zu senden Datenstrom öffnen (ODS).

Sie müssen zuerst ein Kafka-ODS-Ziel in den Administrationseinstellungen konfigurieren, wofür System- und Zugriffsadministrationsrechte erforderlich sind. Informationen zur Konfiguration finden Sie in der Offene Datenströme Abschnitt in der ExtraHop Admin-UI-Leitfaden.

Methoden

`send`

Sendet eine Reihe von Nachrichten an ein einzelnes Thema mit einer Option, die angibt, an welche Kafka-Partition die Nachrichten gesendet werden.

Syntax:

```
Remote.Kafka.send({"topic": "topic", "messages":[messages],
"partition": partition})
```

```
Remote.Kafka("name").send({"topic": "topic", "messages":
[messages],
"partition": partition})
```

Parameter:

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`topic:` ***Schnur***

Eine Zeichenfolge, die dem Thema entspricht, das mit dem Kafka verknüpft ist `send` Methode. Für die Themenzeichenfolge gelten die folgenden Einschränkungen:

- Die Länge der Zeichenfolge muss zwischen 1 und 249 Zeichen liegen.
- Die Zeichenfolge unterstützt nur alphanumerische Zeichen und die folgenden Symbole: „-“, „_“ oder „.“.
- Die Zeichenfolge darf nicht „.“ oder „..“ sein.

`messages:` ***Reihe***

Ein optionales Array von Nachrichten, die gesendet werden sollen. Ein Element in diesem Array kann selbst kein Array sein.

`partition:` ***Zahl***

Eine optionale nicht negative Ganzzahl, die der Kafka-Partition entspricht, an die die Nachrichten gesendet werden. Die `send` Die Aktion schlägt unbemerkt fehl, wenn die angegebene Anzahl die Anzahl der Partitionen auf dem Kafka-Cluster übersteigt, die dem angegebenen Ziel zugeordnet sind. Dieser Wert wird ignoriert, es sei denn **Manuelles Partitionieren** wurde als Partitionierungsstrategie ausgewählt, als Sie den offenen Datenstrom in den Administrationseinstellungen konfiguriert haben.

Rückgabewerte:

Keine

Beispiele:

```
Remote.Kafka.send({"topic": "my_topic", "messages": ["hello
 world", 42,
DHCP.msgType], "partition": 2});
```

```
Remote.Kafka("my-target").send({"topic": "my_topic", "messages":
 [HTTP.query,
HTTP.uri]});
```

`send`

Sendet Nachrichten zu einem einzelnen Thema.

Syntax:

```
Remote.Kafka.send("topic", message1, message2, etc...)
```

```
Remote.Kafka("my-target").send("topic", message1, message2,
 etc...)
```

Parameter:

Wenn `Remote.Kafka.send` wird mit mehreren Argumenten aufgerufen, die folgenden Felder sind erforderlich:

`topic:` ***Schnur***

Eine Zeichenfolge, die dem Thema entspricht, das mit dem Kafka verknüpft ist `send` Methode. Für die Themenzeichenfolge gelten die folgenden Einschränkungen:

- Die Länge der Zeichenfolge muss zwischen 1 und 249 Zeichen liegen.
- Die Zeichenfolge unterstützt nur alphanumerische Zeichen und die folgenden Symbole: „-“, „_“ oder „.“.
- Die Zeichenfolge darf nicht „.“ oder „..“ sein.

`messages:` ***Schnur | Zahl***

Die zu sendenden Nachrichten. Das kann kein Array sein.

Rückgabewerte:

Keine.

Beispiele:

```
Remote.Kafka.send("my_topic", HTTP.query, HTTP.uri);
```

```
Remote.Kafka("my-target").send("my_topic", HTTP.query, HTTP.uri);
```

Remote.MongoDB

Die Remote.MongoDB Klasse ermöglicht das Einfügen, Entfernen und Aktualisieren MongoDB Dokumentensammlungen über eine MongoDB Datenstrom öffnen (ODS).

Sie müssen zuerst ein MongoDB-ODS-Ziel in den Administrationseinstellungen konfigurieren, wofür System- und Zugriffsadministrationsrechte erforderlich sind. Informationen zur Konfiguration finden Sie in der Offene Datenströme Abschnitt in der ExtraHop Admin-UI-Leitfaden.

Methoden

insert

Fügt ein Dokument oder eine Reihe von Dokumenten in eine Sammlung ein und verarbeitet sowohl das Hinzufügen als auch das Ändern.

Syntax:

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

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

Parameter:

name: ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt default.

collection: ***Schnur***

Der Name einer Gruppe von MongoDB-Dokumenten.

document: ***Objekt***

Das Dokument im JSON-Format, das in die Sammlung eingefügt werden soll.

Rückgabewerte:

Retouren true wenn die Anfrage in der Warteschlange steht, andernfalls kehrt sie zurück false.

Beispiele:

```
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', {example: 1});
if (x) {
   Network.metricAddCount('perf_trigger_success', 1);
```

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

Beziehen Sie sich auf http://docs.mongodb.org/manual/reference/method/db.collection.insert/#db.collection.insert für weitere Informationen.

`remove`

Entfernt Dokumente aus einer Sammlung.

Syntax:

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

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

Parameter:

`name:` ***Schnur***

Der optionale Name des Hosts, der bei der Konfiguration des offenen Datenstroms in den Administrationseinstellungen angegeben wurde. Wenn kein Host angegeben ist, ist der Wert der Standardhost.

`collection:` ***Schnur***

Der Name einer Gruppe von MongoDB-Dokumenten.

`document:` ***Objekt***

Das Dokument im JSON-Format, das aus der Sammlung entfernt werden soll.

`justOnce:` ***Boolescher Wert***

Ein optionaler boolescher Parameter, der das Entfernen auf nur ein Dokument beschränkt. Eingestellt auf `true` um das Löschen einzuschränken. Der Standardwert ist `false`.

Rückgabewerte:

Retouren `true` wenn die Anfrage in der Warteschlange steht, andernfalls kehrt sie zurück `false`.

Beispiele:

```
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);
}
```

Beziehen Sie sich auf http://docs.mongodb.org/manual/reference/method/db.collection.remove/#db.collection.remove für weitere Informationen.

`update`

Ändert ein vorhandenes Dokument oder Dokumente in einer Sammlung.

Syntax:

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

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

Parameter:

`collection:` ***Schnur***

Der Name einer Gruppe von MongoDB-Dokumenten.

`document:` ***Objekt***

Das Dokument im JSON-Format, das angibt, welche Dokumente aktualisiert oder eingefügt werden sollen, wenn die Option upsert auf true gesetzt ist.

`update:` ***Objekt***

Das Dokument im JSON-Format, das angibt, wie die angegebenen Dokumente aktualisiert werden sollen.

`name:` ***Schnur***

Der Name des Hosts, der bei der Konfiguration des offenen Datenstroms in den Administrationseinstellungen angegeben wurde. Wenn kein Host angegeben wurde, ist der Wert der Standardhost.

`options:`

Optionale Flags, die auf die folgenden zusätzlichen Aktualisierungsoptionen hinweisen:

`upsert:` ***Boolescher Wert***

Ein optionaler boolescher Parameter, der ein neues Dokument erstellt, wenn kein Dokument mit den Abfragedaten übereinstimmt. Eingestellt auf `true` um ein neues Dokument zu erstellen. Der Standardwert ist `false`.

`multi:` ***Boolescher Wert***

Ein optionaler boolescher Parameter, der alle Dokumente aktualisiert, die den Abfragedaten entsprechen. Eingestellt auf `true` um mehrere Dokumente zu aktualisieren. Der Standardwert ist `false`, wodurch nur das erste zurückgegebene Dokument aktualisiert wird.

Rückgabewerte:

Der Wert ist `true` wenn die Anfrage in der Warteschlange steht, andernfalls kehrt sie zurück FALSE.

Beispiele:

```
var x = Remote.MongoDB.update('test.tbc', {_id: 1}, {$set:
 {example:2}},
{'upsert':true, 'multi':false} );
if (x) {
   Network.metricAddCount('perf_trigger_success', 1);
}
else {
   Network.metricAddCount('perf_trigger_error', 1);
}
```

Beziehen Sie sich auf http://docs.mongodb.org/manual/reference/method/db.collection.update/#db.collection.update für weitere Informationen.

Beispiele für Trigger

- Beispiel: Syslog über TCP mit universeller Nutzlastanalyse analysieren

`Remote.Raw`

Die `Remote.Raw` Klasse ermöglicht es Ihnen, Rohdaten an ein Raw zu senden Datenstrom öffnen (ODS) - Ziel über einen TCP- oder UDP-Port.

Sie müssen zuerst ein ODS-Rohziel in den Administrationseinstellungen konfigurieren, wofür System- und Zugriffsadministrationsrechte erforderlich sind. Informationen zur Konfiguration finden Sie in der Offene Datenströme Abschnitt in der ExtraHop Admin-UI-Leitfaden.

Hinweis: Wenn die Gzip-Funktion für den Rohdatenstrom in den Administrationseinstellungen aktiviert ist, komprimiert die Remote.Raw-Klasse die Daten automatisch mit Gzip.

Methoden

`send`

Sendet Rohdaten über einen TCP- oder UDP-Port an ein ODS-Ziel (Raw Open Data Stream).

Syntax:

```
Remote.Raw.send("data")
```

```
Remote.Raw("name").send("data")
```

Parameter:

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`data:` ***Schnur***

Die JavaScript-Zeichenfolge, die die zu sendenden Bytes darstellt.

Rückgabewerte:

Keine

Beispiele

```
Remote.Raw.send("data over the wire");
```

```
Remote.Raw("my-target").send("extra data for my-target");
```

`Remote.Syslog`

Die `Remote.Syslog` Klasse ermöglicht es Ihnen, Remote-Syslog-Nachrichten zu erstellen und Nachrichtendaten an ein Syslog zu senden Datenstrom öffnen (ODS).

Sie müssen zuerst ein Syslog-ODS-Ziel in den Administrationseinstellungen konfigurieren, wofür System- und Zugriffsadministrationsrechte erforderlich sind. Informationen zur Konfiguration finden Sie in der Offene Datenströme Abschnitt in der ExtraHop Admin-UI-Leitfaden.

Hinweis: Wenn das Senden einer Rsyslog-Nachricht erfolgreich ist, geben die APIs true zurück. Bei Erfolg oder Misserfolg wird der Auslöser weiterhin ausgeführt, da es sich bei einem Fehler beim Senden einer Rsyslog-Meldung um einen „weichen“ Fehler handelt. Eine falsche Verwendung der APIs, d. h. das Aufrufen mit der falschen Anzahl oder Art von Argumenten, führt immer noch dazu, dass die Ausführung des Auslöser gestoppt wird.

Methoden

`emerg(message:` ***Schnur*** `):` ***Leere***

Sendet eine Nachricht mit einem Notfallschweregrad an den Remote-Syslog-Server.

Syntax:

```
Remote.Syslog.emerg("eh_event=web uri=" + HTTP.uri + " req_size="
 + HTTP.reqSize + "
```

```
rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

```
Remote.Syslog("name").emerg("eh_event=web uri=" + HTTP.uri + "
 req_size=" +
HTTP.reqSize + " rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

Parameter

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`alert(message:` ***Schnur*** `):` ***Leere***

Sendet eine Nachricht mit einem Warnschweregrad an den Remote-Syslog-Server.

Syntax:

```
Remote.Syslog.alert("eh_event=web uri=" + HTTP.uri + " req_size="
 + HTTP.reqSize + "
rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

```
Remote.Syslog("name").alert("eh_event=web uri=" + HTTP.uri + "
 req_size=" +
HTTP.reqSize + " rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

Parameter

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`crit(message:` ***Schnur*** `):` ***Leere***

Sendet eine Nachricht mit einem kritischen Schweregrad an den Remote-Syslog-Server.

Syntax:

```
Remote.Syslog.crit("eh_event=web uri=" + HTTP.uri + " req_size=" +
 HTTP.reqSize + "
rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

```
Remote.Syslog("name").crit("eh_event=web uri=" + HTTP.uri + "
 req_size=" +
HTTP.reqSize + " rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

Parameter

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`error(message:` ***Schnur*** `):` ***Leere***

Sendet eine Nachricht mit einem Schweregrad des Fehlers an den Remote-Syslog-Server.

Syntax:

```
Remote.Syslog.error("eh_event=web uri=" + HTTP.uri + " req_size="
 + HTTP.reqSize + "
```

```
rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

```
Remote.Syslog("name").error("eh_event=web uri=" + HTTP.uri + "
 req_size=" +
HTTP.reqSize + " rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

Parameter

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`warn(message:` ***Schnur*** `):` ***Leere***

Sendet eine Nachricht mit einem Warnschweregrad an den Remote-Syslog-Server.

Syntax:

```
Remote.Syslog.warn("eh_event=web uri=" + HTTP.uri + " req_size=" +
 HTTP.reqSize + "
rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

```
Remote.Syslog("name").warn("eh_event=web uri=" + HTTP.uri + "
 req_size=" +
HTTP.reqSize + " rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

Parameter

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`notice(message:` ***Schnur*** `):` ***Leere***

Sendet eine Nachricht mit dem Schweregrad „Hinweis“ an den Remote-Syslog-Server.

Syntax:

```
Remote.Syslog.notice("eh_event=web uri=" + HTTP.uri + " req_size="
 + HTTP.reqSize + "
rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

```
Remote.Syslog("name").notice("eh_event=web uri=" + HTTP.uri + "
 req_size=" +
HTTP.reqSize + " rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

Parameter

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`info(message:` ***Schnur*** `):` ***Leere***

Sendet eine Nachricht mit einem Informationsschweregrad an den Remote-Syslog-Server.

Syntax:

```
Remote.Syslog.info("eh_event=web uri=" + HTTP.uri + " req_size=" +
 HTTP.reqSize + "
```

```
rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

```
Remote.Syslog("name").info("eh_event=web uri=" + HTTP.uri + "
 req_size=" +
HTTP.reqSize + " rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

Parameter

`name:` ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

`debug(message:` ***Schnur*** `):` ***Leere***

Sendet eine Nachricht mit einem Debug-Schweregrad an den Remote-Syslog-Server.

Syntax:

```
Remote.Syslog.debug("eh_event=web uri=" + HTTP.uri + " req_size="
 + HTTP.reqSize + "
rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

```
Remote.Syslog("name").debug("eh_event=web uri=" + HTTP.uri + "
 req_size=" +
HTTP.reqSize + " rsp_size=" + HTTP.rspSize + " processingTime=" +
 HTTP.processingTime);
```

Parameter

name: ***Schnur***

Der Name des ODS-Ziels, an das Anfragen gesendet werden. Wenn dieses Feld nicht angegeben ist, wird der Name auf gesetzt `default`.

Größe der Nachricht

Standardmäßig ist die an den Remoteserver gesendete Nachricht auf 1024 Byte begrenzt, einschließlich Nachrichtenkopf und Trailer (falls erforderlich). Der Nachrichtenkopf enthält immer die Priorität und den Zeitstempel, die zusammen bis zu 30 Byte betragen.

Wenn Sie über System- und Zugriffsadministrationsrechte verfügen, können Sie die Standardnachrichtengröße in den Administrationseinstellungen erhöhen. klicken **Konfiguration ausführen** klicken Sie im Bereich Appliance-Einstellungen auf **Konfiguration bearbeiten**. Gehen Sie zum Abschnitt „remote“ und fügen Sie unter dem ODS-Zielnamen, z. B. „rsyslog“, „message_length_max“ hinzu, wie im Beispiel unten gezeigt. Die Einstellung „message_length_max“ gilt nur für die Nachricht, die an die Remote.Syslog-APIs übergeben wird; der Nachrichtenheader wird nicht auf das Maximum angerechnet.

```
"remote": {
   "rsyslog": {
   "host": "hostname",
   "port": 54322,
   "ipproto": "tcp",
   "message_length_max": 4000
   }
}
```

Zeitstempel

Das Standard-Zeitstempelformat für Rsyslog-Nachrichten ist UTC. Sie können den Zeitstempel auf Ortszeit ändern, wenn Sie den offenen Datenstrom in den Administrationseinstellungen konfigurieren.

Beispiele für Trigger

- Beispiel: Erkannte Gerätedaten an einen Remote-Syslog-Server senden
- Beispiel: Syslog über TCP mit universeller Nutzlastanalyse analysieren
- Beispiel: Passende Topnset-Schlüssel

Remote

Die Remote Mit dieser Klasse können Sie Daten über einen Open Data Stream (ODS) an ein Syslog, eine Datenbank oder einen Server eines Drittanbieters senden und auf Antworten zugreifen, die von HTTP-ODS-Zielen zurückgegeben werden.

Ereignisse

REMOTE_RESPONSE

Wird ausgeführt, wenn das ExtraHop-System eine Antwort von einem HTTP-ODS-Ziel erhält.

Hinweis: Ein Auslöser wird nur dann für das REMOTE_RESPONSE-Ereignis ausgeführt, wenn der Auslöser die ODS-Anfrage erstellt hat, die die Antwort ausgelöst hat.

Eigenschaften

response: ***Objekt***

Ein Objekt, das Informationen aus der vom ODS-Ziel zurückgegebenen HTTP-Antwort enthält. Das Antwortobjekt hat die folgenden Eigenschaften:

statusCode: ***Zahl***

Der vom ODS-Ziel zurückgegebene Statuscode.

body: ***Puffer***

Der Hauptteil der HTTP-Antwort, die vom ODS-Ziel gesendet wurde.

headers: ***Objekt***

Ein Objekt, das die Header der vom ODS-Ziel gesendeten HTTP-Antwort enthält. Wenn die Antwort mehrere Header mit demselben Namen enthält, ist der Wert für den Header ein Array. Zum Beispiel, wenn Set-Cookie ist in der Antwort mehrfach angegeben, Sie können auf das erste Cookie zugreifen, indem Sie angeben Remote.response.headers["Set-Cookie"][0].

context: ***Objekt*** | ***Schnur*** | ***Zahl*** | ***Boolescher Wert*** | ***null***

Die in der remote.Http angegebenen Kontextinformationen context Parameter, als die ODS-Anfrage gesendet wurde. Weitere Informationen finden Sie unter Remote.HTTP .

Datastore-Klassen

Mit den Trigger-API-Klassen in diesem Abschnitt können Sie auf Datenspeicher- oder Bridge-Metriken zugreifen.

Klasse	Beschreibung
AlertRecord	Ermöglicht Ihnen den Zugriff Alarm Informationen über `ALERT_RECORD_COMMIT` Ereignisse.
Dataset	Ermöglicht Ihnen den Zugriff auf RAW Datensatz Werte und bietet eine Schnittstelle für die Berechnung von Perzentilen.
MetricCycle	Ermöglicht das Abrufen von Metriken, die während eines Metrikzyklusintervalls veröffentlicht wurden, dargestellt durch `METRIC_CYCLE_BEGIN`, `METRIC_CYCLE_END`, und `METRIC_RECORD_COMMIT` Ereignisse.
MetricRecord	Ermöglicht den Zugriff auf die aktuellen Metriken auf `METRIC_RECORD_COMMIT` Ereignisse.
Sampleset	Ermöglicht das Abrufen von Übersichtsdaten zu Metriken.
Topnset	Ermöglicht Ihnen den Zugriff auf Daten aus einer Sammlung von Metriken, gruppiert nach einem Schlüssel wie einem URI oder einem Client IP-Adresse.

`AlertRecord`

Die AlertRecord-Klasse ermöglicht Ihnen den Zugriff Alarm Informationen über `ALERT_RECORD_COMMIT` Ereignisse.

Ereignisse

`ALERT_RECORD_COMMIT`

Wird ausgeführt, wenn eine Alarm auftritt. Ermöglicht den Zugriff auf Informationen zur Alarm.

Zusätzliche Datenspeicheroptionen sind verfügbar, wenn Sie einen Auslöser erstellen, der bei diesem Ereignis ausgeführt wird. siehe Erweiterte Trigger-Optionen für weitere Informationen.

Hinweis: Sie können Trigger, die nur bei diesem Ereignis ausgeführt werden, nicht bestimmten Geräten oder Gerätegruppen zuweisen. Trigger, die bei diesem Ereignis ausgeführt werden, werden immer dann ausgeführt, wenn dieses Ereignis eintritt.

Wichtig: Dieses Ereignis wird nur ausgeführt, wenn das NPM-Modul auf dem ExtraHop-System aktiviert ist. Wenn Ihrem Benutzerkonto kein NPM-Modulzugriff gewährt wurde, können Sie keinen Auslöser so konfigurieren, dass er bei diesem Ereignis ausgeführt wird.

Eigenschaften

`description:` ***Schnur***

Die Beschreibung der Alarm, wie sie im ExtraHop-System erscheint.

`id:` ***Schnur***

Die ID des Alert-Datensatzes. Alert-Datensatz-IDs werden nach dem folgenden Format benannt:

```
extrahop.<object>.<alert_type>
```

`<object>` ist der Objekttyp, für den sich die Alarm bezieht. Für Netzwerkobjekte ist der `<object>` Wert ist Erfassung. Wenn sich die Alarm auf eine detaillierte Topnset-Metrik Metrik, wird `<alert_type>` ist `alert_detail`; andernfalls `<alert_type>` ist `alert`. Die folgenden Alert-Datensatz-IDs sind gültig:

- `extrahop.capture.alert`
- `extrahop.capture.alert_detail`
- `extrahop.device.alert`
- `extrahop.device.alert_detail`
- `extrahop.application.alert`
- `extrahop.application.alert_detail`
- `extrahop.flow_network.alert`
- `extrahop.flow_network.alert_detail`
- `extrahop.flow_interface.alert`
- `extrahop.flow_interface.alert_detail`

Hinweis: Sie können den Auslöser so einschränken, dass er nur für bestimmte Alert-Datensatztypen ausgeführt wird. Geben Sie eine kommagetrennte Liste von Alert-Datensatz-IDs in das **Metrische Typen** Feld der erweiterten Triggeroptionen.

`name:` ***Schnur***

Der Name der Alarm.

`object:` ***Objekt***

Das Objekt, für das sich die Alarm bezieht. Für Gerät-, Anwendung-, Erfassung-, Flussschnittstelle- oder Flow-Netzwerkwarnungen enthält diese Eigenschaft einen Device , Application , Network , FlowInterface , oder FlowNetwork jeweils ein Objekt.

`time:` ***Zahl***

Die Uhrzeit, mit der der Warnungsdatensatz veröffentlicht wird.

`severityName:` ***Schnur***

Der Name des Schweregrad Alarm. Die folgenden Schweregrad werden unterstützt:

Wert	Beschreibung
`emerg`	Notfall
`alert`	Warnung
`crit`	Kritisch
`err`	Fehler
`warn`	Warnung
`notice`	Hinweis
`info`	Informationen
`debug`	Debuggen

`severityLevel:` ***Zahl***

Der numerische Schweregrad der Alarm. Die folgenden Schweregrad werden unterstützt:

Wert	Beschreibung
`0`	Notfall
`1`	Warnung
`2`	Kritisch
`3`	Fehler
`4`	Warnung
`5`	Hinweis
`6`	Informationen
`7`	Debuggen

`Dataset`

Die Dataset-Klasse ermöglicht Ihnen den Zugriff auf rohe Datensatzwerte und bietet eine Schnittstelle für die Berechnung von Perzentilen.

Instanzmethoden

`percentile(...):` ***Reihe*** | ***Zahl***

Akzeptiert eine Liste von Perzentilen (entweder als Array oder als mehrere Argumente) zur Berechnung und gibt die berechneten Perzentilwerte für den Datensatz zurück. Wenn ein einzelnes numerisches Argument übergeben wird, wird eine Zahl zurückgegeben. Andernfalls wird ein Array zurückgegeben. Die Argumente müssen in aufsteigender Reihenfolge ohne Duplikate angegeben werden. Fließkommawerte wie 99,99 sind zulässig.

Eigenschaften der Instanz

`entries:` ***Reihe***

Eine Reihe von Objekten mit Frequenz- und Wertattributen. Dies entspricht einer Häufigkeitstabelle, in der eine Reihe von Werten angegeben ist und angegeben ist, wie oft jeder Wert beobachtet wurde.

`MetricCycle`

Die `MetricCycle` class steht für ein Intervall, in dem Metriken veröffentlicht werden. Die MetricCycle-Klasse ist gültig für `METRIC_CYCLE_BEGIN`, `METRIC_CYCLE_END`, und `METRIC_RECORD_COMMIT` Ereignisse.

Die `METRIC_RECORD_COMMIT` Ereignis ist definiert in MetricRecord Abschnitt.

Ereignisse

`METRIC_CYCLE_BEGIN`

Wird ausgeführt, wenn ein Metrik Intervall beginnt.

Hinweis: Sie können Trigger, die nur bei diesem Ereignis ausgeführt werden, nicht bestimmten Geräten oder Gerätegruppen zuweisen. Trigger, die bei diesem

Ereignis ausgeführt werden, werden immer dann ausgeführt, wenn dieses Ereignis eintritt.

`METRIC_CYCLE_END`

Wird ausgeführt, wenn ein Metrik Intervall endet.

Hinweis Sie können Trigger, die nur bei diesem Ereignis ausgeführt werden, nicht bestimmten Geräten oder Gerätegruppen zuweisen. Trigger, die bei diesem Ereignis ausgeführt werden, werden immer dann ausgeführt, wenn dieses Ereignis eintritt.

Zusätzliche Datenspeicheroptionen sind verfügbar, wenn Sie einen Auslöser erstellen, der bei einem dieser Ereignisse ausgeführt wird. siehe Erweiterte Trigger-Optionen für weitere Informationen.

Eigenschaften

`id:` ***Schnur***

Eine Zeichenfolge, die den Metrik Zyklus darstellt. Mögliche Werte sind:

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

`interval:` ***Objekt***

Ein Objekt, das die Eigenschaften „Von“ und „Bis“ enthält, ausgedrückt in Millisekunden seit der Epoche.

`store:` ***Objekt***

Ein Objekt, das Informationen über alle `METRIC_RECORD_COMMIT` Ereignisse, die während eines Metrik Zyklus auftreten, d. h. aus dem `METRIC_CYCLE_BEGIN` Ereignis zum `METRIC_CYCLE_END` Ereignis. Dieses Objekt ist analog zu `Flow.store` Objekt. Die `store` Objekt wird von Triggern gemeinsam genutzt für `METRIC_* events`. Es wird am Ende eines Metrik Zyklus gelöscht.

Beispiele für Trigger

- Beispiel: Metriken zum Metric Cycle Store hinzufügen

`MetricRecord`

Die `MetricRecord` Diese Klasse ermöglicht Ihnen den Zugriff auf die aktuellen Metriken von `METRIC_RECORD_COMMIT` Ereignisse.

Ereignisse

`METRIC_RECORD_COMMIT`

Wird ausgeführt, wenn ein Metrikdatensatz in den Datenspeicher übernommen wird, und bietet Zugriff auf verschiedene Metrikeigenschaften.

Zusätzliche Datenspeicheroptionen sind verfügbar, wenn Sie einen Auslöser erstellen, der bei diesem Ereignis ausgeführt wird. siehe Erweiterte Trigger-Optionen für weitere Informationen.

Hinweis Sie können Trigger, die nur bei diesem Ereignis ausgeführt werden, nicht bestimmten Geräten oder Gerätegruppen zuweisen. Trigger, die bei diesem Ereignis ausgeführt werden, werden immer dann ausgeführt, wenn dieses Ereignis eintritt.

Eigenschaften

`fields:` ***Objekt***

Ein Objekt, das Metrik Werte enthält. Die Eigenschaften sind die Feldnamen und die Werte können Zahlen sein, Oberster Satz, Datensatz oder Probenset.

`id:` ***Schnur***

Der Metrik Typ, wie `extrahop.device.http_server`.

`object:` ***Objekt***

Das Objekt, auf das sich die Metrik bezieht. Für Gerät-, Anwendung- oder VLAN-Warnungen enthält diese Eigenschaft einen Device Objekt, ein Application Objekt oder ein VLAN jeweils Instanz. Für Erfassungsmetriken wie `extrahop.capture.net`, das Anwesen enthält eine Network Objekt. Der folgende Beispielcode speichert die ID einer Anwendung in einer Variablen:

```
var app_id = MetricRecord.object.id;
```

Hinweis: Der obige Beispielcode generiert im Trigger-Editor immer die folgende Warnung:

```
Property 'id' does not exist on type 'Device | Application
 | VLAN | Network'. ts(2339) [2, 33]
Property 'id' does not exist on type 'Network'.
```

Die Warnung weist darauf hin, dass die Zuweisung des Auslöser zu einem Netzwerk nicht unterstützt wird. Sie können diese Warnung ignorieren, wenn der Auslöser einer Anwendung zugewiesen ist.

`time:` ***Zahl***

Die Veröffentlichungszeit des Metrikdatensatzes.

Beispiele für Trigger

- Beispiel: Passende Topnset-Schlüssel
- Beispiel: Metriken zum Metric Cycle Store hinzufügen

`Sampleset`

Mit der Sampleset-Klasse können Sie Übersichtsdaten zu Metriken abrufen.

Eigenschaften

`count:` ***Zahl***

Die Anzahl der Samples im Sampleset.

`mean:` ***Zahl***

Der Durchschnittswert der Proben.

`sigma:` ***Zahl***

Die Standardabweichung.

`sum:` ***Zahl***

Die Summe der Stichproben.

`sum2:` ***Zahl***

Die Summe der Quadrate der Stichproben.

Topnset

Die Topnset class steht für eine Sammlung von Metriken, gruppiert nach einem Schlüssel wie einem URI oder einem Client IP-Adresse.

Für benutzerdefinierte Metriken geben Sie die Topnset entspricht den übergebenen Schlüsseln metricAddDetail*() Methoden. Schlüsselwerte können eine Zahl, eine Zeichenfolge sein, Dataset , Sampleset oder ein anderes Topnset.

Methods

findEntries(key: ***IP-Address*** | ***Schnur*** | ***Objekt***): ***Reihe***

Gibt alle Einträge mit passenden Schlüsseln zurück.

findKeys(key: ***IP-Address*** | ***Schnur*** | ***Objekt***): ***Reihe***

Gibt alle passenden Schlüssel zurück.

lookup(key: ***IP-Address*** | ***Schnur*** | ***Objekt***): ***

Sucht im Topnset nach einem Element und ruft den ersten passenden Eintrag ab.

Eigenschaften

entries: ***Reihe***

Ein Array der Topnset-Einträge. Das Array enthält höchstens N Objekte mit Schlüssel- und Werteigenschaften, wobei N ist derzeit auf 1000 eingestellt.

Schlüssel in der entries Arrays folgen der folgenden Struktur oder dem folgenden Schlüsselmuster:

type: ***Schnur***

Der Typ des obersten Schlüssels. Die folgenden Schlüsseltypen werden unterstützt:

- int
- string
- device_id
- ipaddr
- addr_pair
- ether

value: *

Der Schlüsselwert, der je nach Schlüsseltyp variiert.

- Für int, string, und device_id Schlüssel, der Wert ist eine Zahl, eine Zeichenfolge bzw. eine Geräte-ID.
- Für ipaddr keys, der Wert ist ein Objekt mit den folgenden Eigenschaften:
  - addr
  - proto
  - port
  - device_id
  - origin
- Für addr_pair keys, der Wert ist ein Objekt mit den folgenden Eigenschaften:
  - addr1
  - addr2
  - port1
  - port2
  - proto
- Für ether keys, der Wert ist ein Objekt mit den folgenden Eigenschaften:

- `ethertype`
- `hwaddr`

Veraltete API-Elemente

Die in diesem Abschnitt aufgeführten API-Elemente sind veraltet. Jedes Element enthält eine Alternative und die Version, in der das Element veraltet war.

Wenn Ihr Triggerskript ein veraltetes Element enthält, teilt Ihnen der Syntaxvalidator im Trigger-Editor mit, welches Element veraltet ist, und schlägt ein Ersatzelement vor, falls verfügbar. Sie können den Auslöser erst speichern, wenn Sie Ihren Code korrigiert oder die Syntaxüberprüfung deaktiviert haben. Ersetzen Sie veraltete Elemente, um eine bessere Trigger-Leistung zu erzielen.

Veraltete globale Funktionen

Funktion	Ersatz	Ausführung
`exit(): Leer`	Die Rücksendeerklärung	4,0
`getTimestampMSec(): Zahl`	`getTimestamp(): Zahl`	4,0

Veraltete globale Funktionsparameter

Funktion	Eigentum	Ersatz	Ausführung
`commitDetection()`	`categories`	Du kannst Erkennungskategorien im Erkennungskatalog angeben.	9.3

Veraltete Ereignisse

Ereignis	Ersatz	Ausführung
`NEW_VLAN`	Kein Ersatz	6.1

Veraltete Klassen

Klasse	Ersatz	Ausführung
`RemoteSyslog`	Remote.Syslog	4,0
`XML`	Reguläre Ausdrücke	6,0
`TroubleGroup`	Kein Ersatz	6,0

Veraltete Methoden nach Klassen

Klasse	Methode	Ersatz	Ausführung
Flow	`getApplication(): Schnur`	`getApplications(): Schnur`	5.3
	`setApplication(name: Schnur, turnTiming: Boolescher Wert): Leere`	`addApplication(name: Schnur, turnTiming: Boolescher Wert): Leere`	5.3
Session	`update(key: Schnur, value: *, options: Objekt)*`	`replace(key: Schnur, value: *, options: Objekt): *`	3.9
SSL	`setApplication(name: Schnur): Leere`	`addApplication(name: Schnur): Leere`	5.3

Veraltete Eigenschaften nach Klassen

Klasse	Eigentum	Ersatz	Ausführung
AAA	`error:` *Schnur*	`isError:` *Boolescher Wert*	5,0
	`tprocess:` *Zahl*	`processingTime:` *Zahl*	5.2
DB	`tprocess:` *Zahl*	`processingTime:` *Zahl*	5.2
Detection	`participants.object_type:` *Schnur*	Instanz des Operators	7.8
Discover	`vlan:` *VLAN*	Kein Ersatz	6.1
DNS	`tprocess:` *Zahl*	`processingTime:` *Zahl*	5.2
Flow	`isClientAborted:` *Boolescher Wert*	`isAborted:` *Boolescher Wert*	3,10
	`isServerAborted:` *Boolescher Wert*	`isAborted:` *Boolescher Wert*	3,10
	`turnInfo:` *Schnur*	Oberste Ebene Turn Objekt mit Attributen für den Zug	3.9
FTP	`tprocess:` *Zahl*	`processingTime:` *Zahl*	5.2
HL7	tprozess: *Zahl*	Bearbeitungszeit: *Zahl*	5.2
HTTP	`payloadText:` *Schnur*	`payload:` *Puffer*	4,0
	`tprocess:` *Zahl*	`processingTime:` *Zahl*	5.2
IBMMQ	`messageID:` *Schnur*	`msgID:` *Puffer*	5.2
	`msgSize:` *Zahl*	`totalMsgLength:` *Zahl*	5.2
	`objectHandle:` *Schnur*	Kein Ersatz	5,0
	`payload:` *Puffer*	`msg:` *Puffer*	5.2
ICA	`authTicket:` *Schnur*	`user:` *Schnur*	3.7
	`application:` *Schnur*	`program:` *Schnur*	5.2
	`client:` *Schnur*	`clientMachine:` *Schnur*	6,0
LDAP	`tprocess:` *Zahl*	`processingTime:` *Zahl*	5.2
MongoDB	`tprocess:` *Zahl*	`processingTime:` *Zahl*	5.2
NetFlow	`tos:` *Zahl*	`dscp:` *Zahl* `dscp:` *Schnur*	6.1
NTLM	`ntlmRspVersion:` *Schnur*	`rspVersion:` *Schnur*	8.2
SMPP	`tprocess:` *Zahl*	`processingTime:` *Zahl*	5.2
SMTP	`recipient:` *Schnur*	`recipientList:` *Reihe von Zeichenketten*	7,5
	`tprocess:` *Zahl*	`processingTime:` *Zahl*	5.2
SSL	`reqBytes:` *Zahl*	`clientBytes:` *Zahl*	6.1
	`reqL2Bytes:` *Zahl*	`clientL2Bytes:` *Zahl*	6.1

Klasse	Eigentum	Ersatz	Ausführung
	`reqPkts:` *Zahl*	`clientPkts:` *Zahl*	6.1
	`rspBytes:` *Zahl*	`serverBytes:` *Zahl*	6.1
	`rspL2Bytes:` *Zahl*	`serverL2Bytes:` *Zahl*	6.1
	`rspPkts:` *Zahl*	`serverPkts:` *Zahl*	6.1
TCP	`wndSize:` *Zahl*	`initRcvWndSize:` *Zahl*	6.2
	`wndSize1:` *Zahl*	`initRcvWndSize1:` *Zahl*	6.2
	`wndSize2:` *Zahl*	`initRcvWndSize2:` *Zahl*	6.2
Turn	`reqSize:` *Zahl*	`clientBytes:` *Zahl*	4,0
	`reqXfer:` *Zahl*	`clientTransferTime:` *Zahl*	4,0
	`respSize:` *Zahl*	`serverBytes:` *Zahl*	4,0
	`rspXfer:` *Zahl*	`serverTransferTime:` *Zahl*	4,0
	`tprocess:` *Zahl*	`processingTime:` *Zahl*	4,0

Erweiterte Trigger-Optionen

Sie können erweiterte Optionen für einige Ereignisse konfigurieren, wenn Sie einen Auslöser erstellen.

In der folgenden Tabelle werden die verfügbaren erweiterten Optionen und die entsprechenden Ereignisse beschrieben.

Option	Beschreibung	Unterstützte Ereignisse
Zu erfassende Byte pro Paket	Gibt die Anzahl der Byte an, die pro Paket erfasst werden sollen. Die Gefangennahme beginnt mit dem ersten Byte im Paket. Nur diese Option angeben wenn das Trigger-Skript die PCAP durchführt. Ein Wert von 0 gibt an, dass das Capture alle Bytes in jedem sammeln soll Paket.	Alle Ereignisse mit Ausnahme der folgenden Liste werden unterstützt: • `ALERT_RECORD_COMMIT` • `METRIC_CYCLE_BEGIN` • `METRIC_CYCLE_END` • `FLOW_REPORT` • `NEW_APPLICATION` • `NEW_DEVICE` • `SESSION_EXPIRE`
L7-Nutzdatenbyte zum Puffer	Gibt die maximale Anzahl von Nutzdaten-Bytes an, die gepuffert werden sollen. **Hinweis:** Wenn mehrere Trigger laufen auf demselben Ereignis, der Auslöser mit dem Der höchste Wert fü… L7-Payload (Byte to Buffe… bestimmt das Maximum Payload für dieses Ereigni… für jeden Auslöser.	• `CIFS_REQUEST` • `CIFS_RESPONSE` • `HTTP_REQUEST` • `HTTP_RESPONSE` • `ICA_TICK` • `LDAP_RESPONSE`
Bytes in der Zwischenablage	Gibt die Anzahl der Byte an, die in einer Citrix-Zwischenablage	• `ICA_TICK`

Option	Beschreibung	Unterstützte Ereignisse
	gepuffert werden sollen Übertragung.	
Metrischer Zyklus	Gibt die Länge des Metrik Zyklus an, ausgedrückt in Sekunden. Die folgenden Werte sind gültig: • `30sec` • `5min` • `1hr` • `24hr`	• `METRIC_CYCLE_BEGIN` • `METRIC_CYCLE_END` • `METRIC_RECORD_COMMIT`
Metrische Typen	Gibt den Metriktyp anhand des Rohmetriknamens an, z. B. `extrahop.device.http_server`. Geben Sie mehrere an Metrik Typen in einer kommagetrennten Liste.	• `ALERT_RECORD_COMMIT` • `METRIC_RECORD_COMMIT`
Auslöser bei jeder Flow-Turn ausführen	Aktiviert die PCAP auf jedem Fluss abbiegen. Die Analyse pro Spielzug analysiert kontinuierlich die Kommunikation zwischen zwei Endpunkten, um einen einzelnen Nutzdatenpunkt zu extrahieren aus dem Fluss. Wenn diese Option aktiviert ist, alle Werte spezifiziert für **Übereinstimmende Zeichenfolge für den Client** und **Übereinstimmende Zeichenfolge auf dem Server** Optionen sind ignoriert.	• `SSL_PAYLOAD` • `TCP_PAYLOAD`
Portbereich des Clients	Gibt den Client-Portbereich an. Gültige Werte liegen zwischen 0 und 65535.	• `SSL_PAYLOAD` • `TCP_PAYLOAD` • `UDP_PAYLOAD`
Zu puffernde Client-Bytes	Gibt die Anzahl der Client-Bytes an, die gepuffert werden sollen. Der Wert von Diese Option kann nicht auf 0 gesetzt werden, wenn der Wert von **Server-Bytes zum**	• `SSL_PAYLOAD` • `TCP_PAYLOAD`

Option	Beschreibung	Unterstützte Ereignisse
	Puffern Option ist auch auf 0 setzen.	
Suchzeichenfolge für den Client-Puffer	Gibt die Formatzeichenfolge an, die angibt, wann begonnen werden soll Pufferung von Client-Daten. Gibt das gesamte Paket auf einer Zeichenfolge zurück Spiel. Sie können die Zeichenfolge als Text oder Hexadezimalzahlen angeben. Zum Beispiel beide `ExtraHop` und `\x45\x78\x74\x72\x61\x48\x6F \x70` sind gleichwertig. Hexadezimalzahlen sind keine Groß-/Kleinschreibung empfindlich. Jeder für diese Option angegebene Wert wird ignoriert wenn der **Pro Spielzug** oder **Auslöser ausführen auf allen UDP** Paketoption ist aktiviert.	• `SSL_PAYLOAD` • `TCP_PAYLOAD` • `UDP_PAYLOAD`
Portbereich des Servers	Gibt den Serverportbereich an. Gültige Werte sind 0 bis 65535.	• `SSL_PAYLOAD` • `TCP_PAYLOAD` • `UDP_PAYLOAD`
Server-Bytes zum Puffer	Gibt die Anzahl der Server-Bytes an, die gepuffert werden sollen. Der Wert von Diese Option kann nicht auf 0 gesetzt werden, wenn der Wert von **Zu puffernde Client-Bytes** Option ist auch auf 0 setzen.	• `SSL_PAYLOAD` • `TCP_PAYLOAD`
Suchzeichenfolge für den Serverpuffer	Gibt die Formatzeichenfolge an, die angibt, wann begonnen werden soll Pufferung von Serverdaten. Sie können die Zeichenfolge als Text angeben oder Hexadezimalzahlen. Zum Beispiel beide `ExtraHop` und `\x45\x78\x74\x72\x61\x48\x6F \x70` sind gleichwertig. Hexadezimalzahlen sind keine Groß-/Kleinschreibung empfindlich.	• `SSL_PAYLOAD` • `TCP_PAYLOAD` • `UDP_PAYLOAD`

Option	Beschreibung	Unterstützte Ereignisse
	Jeder für diese Option angegebene Wert wird ignoriert wenn der **Pro Spielzug** oder **Auslöser ausführen auf allen UDP** Option ist aktiviert.	
Auslöser für alle UDP-Pakete ausführen	Ermöglicht die Erfassung aller UDP-Datagramme.	• `UDP_PAYLOAD`
Führen Sie FLOW_CLASSIFY für ablaufende, nicht klassifizierte Flows aus	Ermöglicht die Ausführung des Ereignis nach Ablauf, um Metriken zu sammeln zum Flüsse die vorher nicht klassifiziert wurden ablaufend.	• `FLOW_CLASSIFY`
Externe Typen	Gibt die Typen der externen Daten an, die der Auslöser verarbeitet. Das Der Auslöser wird nur ausgeführt, wenn die Nutzlast ein Typfeld mit einem von enthält die angegebenen Werte. Geben Sie mehrere Typen durch Kommas getrennt an auflisten.	1. `EXTERNAL_DATA`

Beispiele

Die folgenden Beispiele sind verfügbar:

Beispiel: ActiveMQ-Metriken sammeln

Der Auslöser in diesem Beispiel zeichnet Zielinformationen vom Java Messaging Service (JMS) auf. Der Auslöser erstellt eine Anwendung und sammelt benutzerdefinierte Metriken, die beinhalten, ob der Broker eines Ereignisses der Sender oder Empfänger ist, und das für dieses Ereignis angegebene JMS-Zielfeld.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `ACTIVEMQ_MESSAGE`

```
var app = Application("ActiveMQ Sample");
    if (ActiveMQ.senderIsBroker) {
       if (ActiveMQ.receiverIsBroker) {
          app.metricAddCount("amq_broker", 1);
          app.metricAddDetailCount("amq_broker", ActiveMQ.queue, 1);
       }
       else {
          app.metricAddCount("amq_msg_out", 1);
          app.metricAddDetailCount("amq_msg_out", ActiveMQ.queue, 1);
       }
}
else {
    app.metricAddCount("amq_msg_in", 1);
    app.metricAddDetailCount("amq_msg_in", ActiveMQ.queue, 1);
}
```

Verwandte Klassen

- ActiveMQ
- Application

Beispiel: Senden Sie Daten mit Remote.Http an Azure

Der Auslöser in diesem Beispiel sendet Daten über einen offenen HTTP-Datenstrom (ODS) an den Microsoft Azure Table-Speicherdienst.

Sie müssen zuerst in den Administrationseinstellungen einen offenen HTTP-Datenstrom konfigurieren, bevor Sie den Auslöser erstellen. Die ODS-Konfiguration enthält die Authentifizierungsinformationen, die für die Anmeldung bei Ihrem Microsoft Azure-Dienst erforderlich sind. Informationen zur Konfiguration finden Sie unter Konfigurieren Sie ein HTTP-Ziel für einen offenen Datenstrom in der ExtraHop Admin-UI-Leitfaden.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: HTTP_RESPONSE

```
// 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";

/* If the header is not set to this value, Azure expects to receive XML;
 * however, it is easier for a trigger to send JSON.
 * The ODS config enables you to specify the datatype of fields; in this
 case
 * the timestamp (TS) field is a datetime even though it is 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":  "my_key", // 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 describe format of TS
 field
    "TS":            now.toISOString(),
    "ServerTime":    HTTP.processingTime,
    "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) } );
```

Verwandte Klassen

- Remote.HTTP
- Flow
- HTTP

Beispiel: Überwachen Sie CIFS-Aktionen auf Geräten

Der Auslöser in diesem Beispiel überwacht die CIFS-Aktionen, die auf Geräten ausgeführt werden, und erstellt dann benutzerdefinierte Gerätemetriken, die die Gesamtzahl der gelesenen und geschriebenen Byte sowie die Anzahl der Byte erfassen, die von CIFS-Benutzern geschrieben wurden, die nicht berechtigt sind, auf eine vertrauliche Ressource zuzugreifen.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `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\\maria" : {read: true, write: true}
};

// Check if this is an action on your 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);
      }
   }
}
```

Verwandte Klassen

- CIFS
- Device
- Flow

Beispiel: HTTP-Antworten auf 500-Ebene nach Kunden-ID und URI verfolgen

Der Auslöser in diesem Beispiel verfolgt HTTP-Serverantworten, die zu dem Fehlercode 500 führen. Der Auslöser erstellt auch benutzerdefinierte Gerätemetriken, die die Kunden-ID und den URI im Header jeder 500-Antwort erfassen.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: HTTP_REQUEST und HTTP_RESPONSE

```
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 save it to
       * 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") {
   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);
   }
}
```

Verwandte Klassen

- HTTP
- Flow
- Device

Beispiel: Antwortmetriken für Datenbankabfragen sammeln

Der Auslöser in diesem Beispiel erstellt benutzerdefinierte Gerätemetriken, die die Anzahl der Antworten und die Verarbeitungszeiten bei Datenbankabfragen erfassen.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `DB_RESPONSE`

```
let stmt = DB.statement;
if (stmt === null) {
    return;
}

// Remove leading whitespace and truncate
stmt = stmt.trimLeft().substr(0, 1023);

// 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.processingTime);
Device.metricAddDetailSampleset("db_proc_time_detail",
                                stmt, DB.processingTime);
```

Verwandte Klassen

- DB
- Device

Beispiel: Erkannte Gerätedaten an einen Remote-Syslog-Server senden

Der Auslöser in diesem Beispiel erkennt, wenn ein neues Gerät auf dem ExtraHop-System erkannt wird, und erstellt Remote-Syslog-Meldungen, die Geräteattribute enthalten.

Sie müssen zunächst in den Administrationseinstellungen einen offenen Remote-Datenstrom konfigurieren, bevor Sie den Auslöser erstellen. Die ODS-Konfiguration spezifiziert den Standort des Remote-Syslog-Servers. Informationen zur Konfiguration finden Sie unter Konfigurieren Sie ein Syslog-Ziel für einen offenen Datenstrom in der ExtraHop Admin-UI-Leitfaden.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `NEW_DEVICE`

```
var dev = Discover.device;
Remote.Syslog.info('Discovered device ' + dev.id + ' (hwaddr: ' + dev.hwaddr
 + ')
');
```

Verwandte Klassen

- Remote.Syslog
- Discover
- Device

Beispiel: Daten mit Remote.http an Elasticsearch senden

Der Auslöser in diesem Beispiel sendet Daten über einen offenen HTTP-Datenstrom (ODS) an einen Elasticsearch-Server.

Sie müssen zuerst in den Administrationseinstellungen einen offenen HTTP-Datenstrom konfigurieren, bevor Sie den Auslöser erstellen. Die ODS-Konfiguration spezifiziert das Elasticsearch-Ziel und alle erforderlichen Authentifizierungsdaten. Informationen zur Konfiguration finden Sie unter Konfigurieren Sie ein HTTP-Ziel für einen offenen Datenstrom in der ExtraHop Admin-UI-Leitfaden.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `HTTP_REQUEST` und `HTTP_RESPONSE`

```
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);
```

Verwandte Klassen

- Remote.HTTP

Beispiel: Zugriff auf HTTP-Header-Attribute

Der Auslöser in diesem Beispiel greift auf HTTP-Ereignisattribute aus dem Header-Objekt zu und erstellt benutzerdefinierte Gerätemetriken, die Header-Anfragen und -Attribute zählen.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `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 is not present). Header values are always strings.
 */

// This syntax also works 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
 *
 * The "length" property is case-sensitive and is not
 * treated as a header lookup. Instead, it 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-");

/* The "results" property 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);
}
```

Verwandte Klassen

- HTTP
- Device

Beispiel: IBM MQ-Metriken sammeln

Die Trigger in diesem Beispiel arbeiten zusammen, um einen Überblick über den Fluss von Nachrichten auf Warteschlangenebene durch die IBMMQ protokoll. Die Trigger erstellen benutzerdefinierte Anwendungsmetriken, die die Anzahl der eingehenden, ausgehenden und zwischen Brokern durch verschiedene Nachrichtenwarteschlangen ausgetauschten Nachrichten zählen.

Führen Sie den folgenden Auslöser auf dem `IBMMQ_REQUEST` Ereignis.

```
if (IBMMQ.method == "MESSAGE_DATA") {
    var app = Application("IBMMQ Sample");
    app.metricAddCount("broker", 1);
    if (IBMMQ.queue !== null) {
        var ret = IBMMQ.queue.split(":");
        var queue = ret.length > 1 ? ret[1] : ret[0];
        app.metricAddDetailCount("broker", queue, 1);
    }
    else {
        app.metricAddCount("queueless_broker", 1);
```

```
    }
    if (IBMMQ.queue !== null && IBMMQ.queue.indexOf("QUEUE2") > -1) {
        app.metricAddCount("queue2_broker", 1);
    }
    app.commit();
}
elseif (IBMMQ.method == "MQPUT" || IBMMQ.method == "MQPUT1") {
    var app = Application("IBMMQ Sample");
    app.metricAddCount("msg_in", 1);
    if (IBMMQ.queue !== null) {
        var ret = IBMMQ.queue.split(":");
        var queue = ret.length > 1 ? ret[1] : ret[0];
        app.metricAddDetailCount("msg_in", queue, 1);
    }
    else {
        app.metricAddCount("queueless_msg_in", 1);
    }
    if (IBMMQ.queue !== null && IBMMQ.queue.indexOf("QUEUE2") > -1) {
        app.metricAddCount("queue2_msg_in", 1);
    }
    app.commit();
}
```

Führen Sie den folgenden Auslöser auf dem IBMMQ_RESPONSE Ereignis.

```
if (IBMMQ.method == "ASYNC_MSG_V7" || IBMMQ.method == "MQGET_REPLY") {
    var app = Application("IBMMQ Sample");
    if (IBMMQ.payload === null) {
        app.metricAddCount("payloadless_msg_out", 1);
    }
    else {
        app.metricAddCount("msg_out", 1);
        if (IBMMQ.queue !== null) {
            var ret = IBMMQ.queue.split(":");
            var queue = ret.length > 1 ? ret[1] : ret[0];
            app.metricAddDetailCount("msg_out", queue, 1);
        }
        else {
            app.metricAddCount("queueless_msg_out", 1);
        }
        if (IBMMQ.queue !== null && IBMMQ.queue.indexOf("QUEUE2") > -1) {
            app.metricAddCount("queue2_msg_out", 1);
        }
    }
    app.commit();
}
```

Verwandte Klassen

- IBMMQ
- Application

Beispiel: Memcache-Treffer und Fehlschläge aufzeichnen

Der Auslöser in diesem Beispiel erstellt benutzerdefinierte Gerätemetriken, die jedes Gerät Datensatz memcache Hit or Miss und die Zugriffszeit jedes Treffers.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: 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);
   }
}
```

Verwandte Klassen

- Memcache
- Device

Beispiel: Memcache-Schlüssel analysieren

Analysiert die memcache Schlüssel zum Extrahieren detaillierter Aufschlüsselungen, z. B. nach ID-Modul und Klassenname, und zum Erstellen benutzerdefinierter Gerätemetriken zur Erfassung wichtiger Details.

Schlüssel sind formatiert als `"com.extrahop.<module>.<class>_<id>"`—zum Beispiel: `"com.extrahop.widgets.sprocket_12345"`.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `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);
         }
      }
   }
}

// 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 keys 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);
```

```
}
```

Verwandte Klassen

- Memcache
- Device

Beispiel: Metriken zum Metric Cycle Store hinzufügen

Der Auslöser in diesem Beispiel veranschaulicht, wie Daten aus allen Commits für Metrikdatensätze, die während eines Metrikzyklus auftreten, vorübergehend gespeichert werden.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `METRIC_CYCLE_BEGIN`, `METRIC_CYCLE_END`, `METRIC_RECORD_COMMIT`

konfigurieren erweiterte Trigger-Optionen wie in der folgenden Tabelle dargestellt:

Wahl	Wert
Metrischer Zyklus	30 Sekunden
Metrischer Typ	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;
}
```

Verwandte Klassen

- MetricCycle
- MetricRecord
- Device

Beispiel: Analysieren von benutzerdefinierten PoS-Nachrichten mit universeller Nutzlastanalyse

Der Auslöser in diesem Beispiel analysiert TCP-Nachrichten von einem Point-of-Sale (PoS) -System und erstellt benutzerdefinierte Gerätemetriken, die spezifische Werte im 4. bis 7. Byte von Antwort- und Anforderungsnachrichten erfassen.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `TCP_PAYLOAD`

```
// Define variables; store client or server payload into a Buffer object

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 is a protocol of no interest or there is no payload
if (protocol !== 'tcp:4015' || (buf_client === null && buf_server === null))
 {
    // debug('Protocol of no interest: ' + 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;
}
```

Verwandte Klassen

- Puffer
- Device
- Flow

Beispiel: Syslog über TCP mit universeller Nutzlastanalyse analysieren

Der Auslöser in diesem Beispiel analysiert das Syslog über TCP und zählt die Syslog-Aktivität im Laufe der Zeit, sowohl netzwerkweit als auch pro Gerät.

Hinweis: Möglicherweise müssen Sie das Trigger-Beispiel bearbeiten, um sicherzustellen, dass die Netzwerkports für Ihren Syslog-Server mit den Ports in Ihrer Umgebung übereinstimmen.

Dieses Trigger-Beispiel steht als Download über ein Lösungspaket von der ExtraHop-Gemeinschaft.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: TCP_PAYLOAD, UDP_PAYLOAD

```
// Global variables
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
 * treat 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));
```

Verwandte Klassen

- Flow
- Network
- Puffer
- Remote.MongoDB
- Remote.Syslog

Beispiel: NTP mit universeller Nutzlastanalyse analysieren

Der Auslöser im folgenden Beispiel analysiert das Netzwerk Time Protokoll durch Universal Payload Analysis (UPA).

Führen Sie den Auslöser bei den folgenden Ereignissen aus: UDP_PAYLOAD

```
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 enables you to continue parsing where you left off
values = buf.unpack(fmt, offset);
ntpData.referenceId = values[0];

// Only the integral parts of the timestamp are referenced 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));
```

Verwandte Klassen

- Puffer
- Flow
- UDP

Beispiel: Daten in einer Sitzungstabelle aufzeichnen

Der Auslöser in diesem Beispiel zeichnet bestimmte HTTP-Transaktionen in der Sitzungstabelle auf und erstellt benutzerdefinierte Netzwerkmetriken, die Sitzungsablaufdaten erfassen.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `HTTP_REQUEST`, `SESSION_EXPIRE`

```
// HTTP_REQUEST
if (event == "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;
   }
   // Specify the matched string as the key for session table entry
   var os_name = os[0];

   var opts =
    {
      // Expire added entries after 30 seconds
      expire: 30,
      // Retain entries with normal priority if 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_name, 0, opts);
   // Increase the count for this entry
   var count = Session.increment(os_name);
   debug(os_name + ": " + count);
}

/* After 30 seconds, the accumulated per-OS counts appear in the
 Session.expiredKeys
 * list, accessible in the SESSION_EXPIRE event:
 */
   //SESSION_EXPIRE
if (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);
      }
   }
}
```

Verwandte Klassen

- HTTP
- Network
- Session

Beispiel: SOAP-Anfragen verfolgen

Der Auslöser in diesem Beispiel verfolgt SOAP-Anfragen über den SoapAction-Header, speichert sie im Flow-Speicher und erstellt benutzerdefinierte Netzwerkmetriken, die Daten über die Transaktionen sammeln.

Hinweis: Bevor Sie beginnen, stellen Sie sicher, dass Ihre SOAP-Implementierung die erforderlichen Informationen über den Header weiterleitet.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `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(detailMethod, Flow.client.ipaddr, 1);
       Network.metricAddSampleset("soap_proc", HTTP.processingTime);
       Network.metricAddDetailSampleset("soap_proc_detail", method,
                                        HTTP.processingTime);
    }
}
```

Verwandte Klassen

- Flow
- HTTP
- Network

Beispiel: Passende Topnset-Schlüssel

Die Trigger in diesem Beispiel veranschaulichen den Topnset-Schlüsselabgleich anhand von Zeichenfolge und IP-Adresse und beinhalten erweiterte Tastenzuordnungen.

Topnset-Schlüsselabgleich nach Zeichenfolge

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `METRIC_RECORD_COMMIT`

konfigurieren erweiterte Trigger-Optionen wie in der folgenden Tabelle dargestellt:

Wahl	Wert
Metrischer Zyklus	30 Sekunden
Metrischer Typ	extrahop.device.app

```
var stat = MetricRecord.fields['bytes_out'],
    id = MetricRecord.object.id,
    proto = 'HTTP2-SSL',
    entry;

entry = stat.lookup(proto);
if (entry !==null) {
```

```
    debug('Device ' + id + ' sent ' + entry.value + ' bytes over ' + proto);
}
```

Topnset-Schlüsselabgleich nach IP-Adresse

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `METRIC_RECORD_COMMIT`

konfigurieren erweiterte Trigger-Optionen wie in der folgenden Tabelle dargestellt:

Wahl	Wert
Metrischer Zyklus	30 Sekunden
Metrischer Typ	extrahop.device.net_detail

```
var stat = MetricRecord.fields['bytes_out'],
    total = 0,
    entry,
    entries,
    i,
    ip = new IPAddress('192.168.112.1');

entries = stat.findEntries(ip);
for (i = 0; i < entries.length; i++) {
    entry = entries[i];
    total += entry.value;
}
Remote.Syslog.alert('IP ' + ip + ' sent ' + total + ' bytes.');
```

Erweiterter Topnet-Tastenabgleich

Führen Sie den Auslöser bei den folgenden Ereignissen aus: `METRIC_RECORD_COMMIT`

konfigurieren erweiterte Trigger-Optionen wie in der folgenden Tabelle dargestellt:

Wahl	Wert
Metrischer Zyklus	30 Sekunden
Metrischer Typ	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));
}
```

Verwandte Klassen

- MetricRecord
- IPAddress
- Remote.Syslog

Beispiel: Erstellen Sie einen Anwendungscontainer

Der Auslöser in diesem Beispiel erstellt einen Anwendungscontainer auf der Grundlage des Datenverkehrs, der mit einer zweistufigen Anwendung verknüpft ist, und erstellt benutzerdefinierte Anwendungsmetriken, die für HTTP - und Datenbankereignisse gesammelt werden.

Führen Sie den Auslöser bei den folgenden Ereignissen aus: HTTP_RESPONSE und DB_RESPONSE

```
/* Initialize the application object against which you will
 * commit specific HTTP and DB transactions. After traffic is
 * committed, an application container called "My App" will appear
 * in the Applications tab in the ExtraHop system.
 */

var myApp = Application("My App");

/* These configurable properties describe features that define
 * your application traffic.
 */

var myAppHTTPHost = "myapp.internal.example.com";
var myAppDatabaseName = "myappdb";
if (event == "HTTP_RESPONSE") {

   /* HTTP transactions can be committed to the application on
    * HTTP_RESPONSE events.
    */

   /* Commit this HTTP transaction only if the HTTP host header for
    * this response is defined and matches your 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 the 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();
    }
}
```

Verwandte Klassen

- Application
- DB
- HTTP