



# **Policy Traffic Switch SandScript Reference Guide, Release 6.30**

**05-00069-C01  
2013-12-15**

The most current version of this document is available on the Sandvine Customer Support web site at <https://support.sandvine.com>.

This document and the products described within are subject to copyright. Under copyright laws, neither this document nor the product may be reproduced, translated, or reduced to any electronic medium or machine readable or other form without prior written authorization from Sandvine.

Copyright 2013, Sandvine Incorporated ULC. All rights reserved. Sandvine™ is a trademark of Sandvine Incorporated ULC. All other product names mentioned herein are trademarks of their respective owners.

Sandvine is committed to ensuring the accuracy of our documentation and to continuous improvement. If you encounter errors or omissions in this user guide, or have comments, questions, or ideas, we welcome your feedback. Please send your comments to Sandvine via email at <https://support.sandvine.com>.

### **Contacting Sandvine**

To view the latest Sandvine documentation or to contact Sandvine Customer Support, register for an account at <https://support.sandvine.com>.

For a list of Sandvine Sales and Support offices, see [http://www.sandvine.com/about\\_us/contact.asp](http://www.sandvine.com/about_us/contact.asp).

Related documentation is available from the Sandvine Customer Support web site.

All documents are in PDF format, which you can open, read, or print using Adobe® Acrobat® Reader®. You can obtain a free copy of this software from the Adobe® web site.

Document	Part Number
PTS Release Notes	05-00214-E01
PTS Hardware Installation Guide	05-00185-A11
PTS Software Installation Guide	05-00245-B01
PTS Network Configuration Guide	05-00192-D01
PTS SandScript Configuration Guide	05-00217-D01
PTS SandScript Reference Guide	05-00069-C01
PTS Alarm Reference Guide	05-00262-A01
PTS CLI Reference Guide	05-00263-A01

# 1 Conventions

When reading through the guide, refer to these formatting conventions:

Formatting	Description	Example
Bold	Item you click on or information you enter.	<b>Click OK</b>
System font	Commands and command line responses.	<code>PTS&gt; show alarms</code>
<parameter in angle brackets>	Parameter for which you supply a value.	<password>
[parameter in square brackets]	Optional parameter	[port_num]
{Choice 1   choice 2 in curly brackets}	Alternative but required keywords separated by vertical bars ( ) in the command line interface.	{port-01   port-02}

# Contents

1 Expressions.....	8
1.1 About Expressions.....	9
1.2 Fields and Expressions.....	9
1.2.1 IP Address Data Type.....	9
1.2.2 Expressions and Syntax.....	10
1.2.3 Expression Scope.....	11
1.2.4 Null.....	12
1.2.5 Define.....	12
1.2.6 Expression Lists.....	12
1.3 Related Documentation.....	12
2 SandScript Fields.....	14
2.1 Flow Fields.....	16
2.1.1 Flow Measurement Fields.....	16
2.1.2 Flow Action Fields.....	19
2.1.3 Flow Direction Fields.....	21
2.1.4 Application Layer Fields.....	22
2.1.5 Transport Layer Fields.....	23
2.1.6 Identification Fields.....	24
2.1.7 Settable Flow Fields.....	25
2.2 Tee and Divert Fields.....	25
2.3 HTTP Fields.....	26
2.3.1 HTTP Fields Usage.....	26
2.3.2 Client Request Packets.....	27
2.3.3 Server Response Packets.....	28
2.4 RTSP Fields.....	29
2.5 RTMP Fields.....	30
2.6 Generic Fields.....	31
2.7 Video Fields.....	31
2.8 Analyzer Flow Fields.....	32
2.8.1 SSL Analyzer Fields.....	32
2.8.2 Instant Messaging Analyzer Fields.....	33
2.8.3 SIP Analyzer Fields.....	34
2.8.4 WAP1 Analyzer Field.....	34
2.9 Streaming Fields.....	34
2.10 Limiters.....	35
2.10.1 Limiter Fields.....	35

2.10.2 Limiter Fields Usage.....	35
2.11 Measurements.....	36
2.11.1 Measurement Fields.....	36
2.11.2 Measurement Fields Usage.....	38
2.12 Classifiers.....	38
2.13 Timers.....	39
2.14 Tables.....	39
2.15 Events.....	40
2.16 Shapers.....	40
2.16.1 Shaper Fields.....	40
2.16.2 Shaper Fields Usage.....	41
2.17 VoIP Quality of Experience .....	41
2.17.1 VoIP QoE Fields Usage.....	42
2.18 Subscriber Session Fields.....	42
2.19 Subscriber and Session Attributes.....	45
2.20 Tunneling.....	46
2.20.1 Supported Field Types.....	47
2.21 Network Protection.....	50
2.22 Utility Fields.....	50
2.23 Expressions for Unique-by.....	50
2.24 Published Expressions.....	51
2.25 Environment Fields.....	51
2.25.1 Environment Fields Example.....	52
2.26 Diameter Fields.....	52
2.27 Controller Fields.....	55
3 Functions.....	58
3.1 General Functions.....	59
3.1.1 Conversion Functions.....	59
3.1.2 Octet Conversion Functions.....	60
3.1.3 Time Functions.....	62
3.1.4 String Functions.....	63
3.1.5 Logical Operator Functions.....	64
3.1.6 Math Functions.....	65
3.1.7 Network Functions.....	66
3.1.8 BGP Functions.....	67
3.1.9 Encryption Functions.....	68
3.2 Map Functions.....	69
3.2.1 String Map Functions.....	69
3.2.2 Hostname Map Functions.....	70

---

3.2.3 URL Map Functions.....	70
3.3 Table Functions.....	71
3.4 Diameter Functions.....	71
3.5 Subscriber Mapping Functions.....	72
A Sample SandScript.....	74





# 1

## Expressions

- ["About Expressions" on page 9](#)
- ["Fields and Expressions" on page 9](#)
- ["Related Documentation" on page 12](#)



## 1.1 About Expressions

SandScript provides extended expression syntax for creating advanced policies that provide more information about traffic than basic SandScript policy language.

While the base language can only be used to create very basic measurements and conditions, extensions to the SandScript language allow a condition to be defined by mathematical and logical functions with constants (integers, floating points, strings) and fields that access data from a subscriber, message, event, flow, or statistic.

Extended language provides these major enhancements over basic language:

- For powerful network visibility and control, fields provide access to more specific information about network conditions and health.
- Complex mathematical operations can be carried out on fields, resulting in customized values and conditions for use in measurements, tables, limiters and shapers.

An expression takes the form of a condition when it is enclosed in `expr()`.

```
if expr(expression) then action(s)
```

## 1.2 Fields and Expressions

SandScript expressions supplement policy language, providing access to a wide range of information that may be used to create complex expressions.

Fields return these types of values:

- Integer (64-bit signed)
- Float (64-bit double precision)
- String (sequence of octets of any length)
- Boolean (true or false)
- IP address (either an IPv4 address or an IPv6 address). See [IP Address Data Type](#) on page 9.

For information on type conversion, see [Conversion Functions](#) on page 59.

Fields can be manipulated and compared with other expressions and constants using logical and mathematical operators to implement specific SandScript policies that manage and monitor network health comprehensively.

The online help (“manual page” or “man page”) is available on the PTS and SDE and provides a quick syntax reference for all available fields. To display the man page, at a command prompt, enter:

```
man pal
```

### 1.2.1 IP Address Data Type

Fields of `ipaddress` type can hold either an IPv4 address or an IPv6 address. IPv6 addresses with the IPv4-mapped IPv6 prefix of `0:0:0:0:fff::/96` are interpreted as IPv4 addresses.

In PTS 5.60 and earlier, you could use IP addresses as integers. But IPv6 addresses are 128 bits, so this is no longer possible in PTS 6.00 and later. Therefore, some policies that worked in 5.60 or earlier releases may fail to load on a 6.00 release. You can rewrite these older policies to work on a 6.00 release.

IP address type cannot be converted to or from any other field type except string. You can use these functions for the conversion:

- IpToString() and StringToIp() converts between a human-readable format and an IP address
- OctetIpToString() and StringIpToOctets() convert between octet strings (of 4- or 16-bytes) and IP address values
- Serialize() and Deserialize() manipulate IP addresses in payloads

For more information on these functions, see [Conversion Functions](#) on page 59 and [Octet Conversion Functions](#) on page 60.

## 1.2.2 Expressions and Syntax

Expressions can be created by combining any fields that provide values of compatible types. Fields are used with constants, logical operators, and mathematical operators to create expressions. Constants must also be of a type compatible with the fields in an expression.

Parenthesis ( ) indicate operation precedence, just as in basic policy language. For example, this rule will be true only if condition1 is true and either condition2 or condition3 is also true:

```
(condition1 and (condition2 or condition3))
```

However, the following rule is true if both condition1 and condition2 are true, or if condition3 is true:

```
((condition1 and condition2) or condition3)
```

A child statement refers to the expression(s) acted on by a particular operator, just as a function has arguments.

### 1.2.2.1 Unary Operator Syntax

The unary operators are:

Operation	Description
not	The logical negation operator for boolean fields or expressions. This negates the field or expression that follows it.
-	The arithmetic negation operator. The child expression must be of integer or floating point type.
is null	Tests if the preceding expression is null.
is not null	Tests if the preceding expression is non-null.

For example:

```
expr(not (Flow.IsReset))
```

When used in a condition statement, this policy excludes all flows which have had the reset action applied to them.

### 1.2.2.2 Binary Operator Syntax

The supported binary operators include:

Operation	Description
+	Numeric addition.
-	Numeric subtraction (or unary minus).
*	Numeric multiplication.
/	Numeric division.
%	Numeric modulo (remainder upon division).

Operation	Description
<	Numeric less-than, or string comparison (alpha-order).
>	Numeric greater-than, or string comparison (alpha-order).
<=	Numeric less-than-or-equal-to, or string comparison (alpha-order).
>=	Numeric greater-than-or-equal-to, or string comparison (alpha-order).
<<	Binary shift left. Operands must be integers.
>>	Binary shift right. Operands must be integers.
=	Equality operator. Operands must be compatible types.
!=	Inequality operator. Operands must be compatible types.
and	The logical intersection operator for boolean fields and expressions. The bitwise binary intersection operator for integer fields and expressions. The bitwise binary intersection operation for ipaddress expressions and fields, however, this operation will return null if the fields or expressions have different IP versions.
or	The logical union operator for boolean fields or expressions, or the bitwise binary union operator for integer fields, expressions and the ipaddress type if both are the same IP version.

### 1.2.2.3 Ternary Operator Syntax

The ternary operator is used to test any boolean condition, and provides instructions for both cases when it evaluates to 'True' and 'False'.

True and false types must be compatible. The syntax is:

```
<boolean_expr> ? <value1> : <value2>
```

This returns value1 if the boolean expression is true, otherwise value2 is returned.

For example, this policy sets the first bit in the settable 32-bit UserInteger field if “.jpg” is present in the HTTP resource field:

```
define "is_http_jpg"= Flow.Client.Stream.HTTP.Resource \  
  is not null ? \  
    regex("\.jpg",Flow.Client.Stream.HTTP.Resource) \  
    : Flow.UserInteger.Bit0  
if true then set Flow.UserInteger.Bit0=is_http_jpg
```

## 1.2.3 Expression Scope

Expressions are applicable in one or more of these scopes:

- Flow Scope:** Policy is executed on a new flow, when a flow is being re-evaluated, and at the end of the flow. All flows are periodically re-evaluated.
- Subscriber Scope:** Policy is periodically applied to mapped subscribers. This requires subscriber IP mapping. If subscriber IP mapping is not installed or licensed, subscriber scope policies are not applied. Subscriber scope is used to update attributes based on subscriber statistics. These dynamic attributes are used to apply traffic management policy (in the flow scope).
- Other Event Scope:** Policy is executed when some other event occurs. Examples of events that cause policy to be executed include policy reload, the receipt of a Diameter message, or the expiration of a policy timer.

Conditions and actions can be carried out in one or more scopes. The combined conditions and actions determine which scope a policy rule belongs to. If an inappropriate match is made, an error is generated. For example, this policy has an inappropriate

match, because the condition (if protocol bittorrent) is in the flow scope, but the action (then set\_attribute) is in the subscriber scope:

```
if protocol bittorrent then set_attribute "p2p_user" = "true" 24 hours
```

Scoping of rules within a policy group must be consistent. Flow scopes and subscriber scopes can not be combined within the same policy group. The entire scope within the policy group, including all nested policy groups must resolve to at least one scope.

## 1.2.4 Null

Expressions can return null results. Null is not like a normal value. You may interpret null as a value that is not known. So some operations with unknown input produces an unknown output.

Any operation on a null value returns a null value, with these exceptions:

```
(true OR null) = true  
(false AND null) = false
```

Null can also be tested: (<expr> is null) returns true if the expression is null.



**Note:**

(null = null) returns null.

## 1.2.5 Define

Define is used to create new fields to be referenced later in policy.

Define can be used to assign constant values, or values based on one or more existing fields.

```
define "seconds_per_minute" = 60  
define "Flow.age_in_minutes" = Flow.age / seconds_per_minute
```

You can use the newly defined fields as built-in fields in subsequent expressions. Each time an expression uses the newly defined fields, it substitutes the specified definition.

## 1.2.6 Expression Lists

Expression lists are used to specify multiple expressions as arguments. It is commonly used for unique by expressions:

```
measurement "MeasurementName" unique by (expr1, expr2)
```

Note that all expressions in the expression list must be of compatible scopes.

# 1.3 Related Documentation

For more information about PTS refer to:

Document	Has Information About...	Part Number
PTS & SPB 5.40 CLI Reference Guide	Configuring, monitoring, or maintaining an element. CLI command usage and output is described.	05-00193

Document	Has Information About...	Part Number
PTS 5.51 CLI Reference Guide	Configuring, monitoring, or maintaining an element. CLI command usage and output is described.	05-00182
Sandvine Operations Reference Guide	Alarms and CLI commands for PTS and SPB.	05-00183
Loadable Traffic Identification Package	Protocols, sub-protocols, and keywords used to identify the protocol in policies. This guide is relevant to anyone who is creating policy.	05-00070
PTS 5.40 Policy Guide PTS 5.51 Policy Guide PTS 5.60 Policy Guide PTS 6.00 Policy Guide PTS 6.10 Policy Guide	Creating policies on a Policy Traffic Switch (PTS) element. It is written for individuals tasked with creating policies.	05-00055-D12 05-00055-E10 05-00055-G07 05-00217-A01 05-00217-B01

To view the latest Sandvine documentation, register for an account at:

<https://support.sandvine.com>



# 2

## SandScript Fields

- ["Flow Fields" on page 16](#)
- ["Tee and Divert Fields" on page 25](#)
- ["HTTP Fields" on page 26](#)
- ["RTSP Fields" on page 29](#)
- ["RTMP Fields" on page 30](#)
- ["Generic Fields" on page 31](#)
- ["Video Fields" on page 31](#)
- ["Analyzer Flow Fields" on page 32](#)
- ["Streaming Fields" on page 34](#)
- ["Limiters" on page 35](#)
- ["Measurements" on page 36](#)
- ["Classifiers" on page 38](#)
- ["Timers" on page 39](#)
- ["Tables" on page 39](#)
- ["Events" on page 40](#)
- ["Shapers" on page 40](#)

- ["VoIP Quality of Experience " on page 41](#)
- ["Subscriber Session Fields" on page 42](#)
- ["Subscriber and Session Attributes" on page 45](#)
- ["Tunneling" on page 46](#)
- ["Network Protection" on page 50](#)
- ["Utility Fields" on page 50](#)
- ["Expressions for Unique-by" on page 50](#)
- ["Published Expressions" on page 51](#)
- ["Environment Fields" on page 51](#)
- ["Diameter Fields" on page 52](#)
- ["Controller Fields" on page 55](#)

## 2.1 Flow Fields

Properties of a flow can be exposed using flow fields. Flow fields are available per flow at the start-of-flow, mid-flow, and end-of-flow evaluations. Some flow fields return null depending on properties of the flow, including the protocol and subscriber state.

### 2.1.1 Flow Measurement Fields

The available fields include:

Field	Description	Type
Flow.Client Server Subscriber Internet.Rx Tx.Rate	Returns the bitrate over the last 5 seconds, or last 65,535 bytes, in bits per second for the specified node (Client, Server, Subscriber, or Internet) and direction (Rx or Tx). Can be used to apply SandScript to flows that are above a particular threshold. For example: <code>expr (Flow.Client.Rx.Rate &gt; 1000000)</code> would be true for flows where the client is receiving more than 1Mbps of traffic. The field has a value of NULL if it is not calculated.	Integer
Flow.Client Server Subscriber Internet.Rx Tx.TotalBytes Flow.Client Server Subscriber Internet.Rx Tx.TotalPackets	The total number of bytes/packets that have been transmitted by (Tx) or received by (Rx) the specified node since the beginning of the flow.	Integer
Flow.Client Server Subscriber Internet.Rx Tx.Bytes Flow.Client Server Subscriber Internet.Rx Tx.Packets	The number of bytes/packets that have been transmitted by (Tx) or received by (Rx) the specified node since the last time SandScript was evaluated for the flow.	Integer
Flow.Client Server Subscriber Internet.Tx.TOS	The value of the IP TOS/DSCP field of the flow transmitted by the specified node. The Flow.Subscriber Internet.Tx.TOS fields are only available if port roles are defined. The TOS field is updated packet by packet, and the PTS SandScript is not evaluated for each packet. So the SandScript rule that uses the TOS field may not catch TOS field change instantly. There is likely a lag of few packets between the actual TOS field change and the PTS detection of the change depends on the configuration of the PTS mid-flow re-evaluation interval.	Integer
Flow.Client Server Subscriber Internet.Rx Tx.ProtocolBytes Flow.Client Server Subscriber Internet.Rx Tx.ProtocolPackets	The number of bytes/packets that have been transmitted by (Tx) or received by (Rx) the specified node since the last time SandScript was evaluated for the flow. Has a value of 0 before the protocol is known. When the protocol first becomes known, returns the total bytes/packets since the start of the flow up to that point, thereafter is equivalent to .Bytes Packets. This is intended for use in protocol-dependent SandScript (for example, counting bytes/packets for a specific protocol).	Integer
Flow.Client Server Subscriber Internet.Rx Tx.SubscriberBytes Flow.Client Server Subscriber Internet.Rx Tx.SubscriberPackets	The number of bytes/packets that have been transmitted by (Tx) or received by (Rx) the specified node since the last time SandScript was evaluated for the flow. This has a value of 0 before the subscriber becomes mapped. When the subscriber first becomes mapped, returns the total bytes/packets	Integer



Field	Description	Type
	<p>since the start of the flow up to that point, then is equivalent to .Bytes Packets.</p> <p>This is intended for use in subscriber-dependent SandScript (for example, counting bytes/packets per subscriber).</p>	
Flow.Client Server Subscriber Internet.Rx Tx.NonSubscriberBytes Flow.Client Server Subscriber Internet.Rx Tx.NonSubscriberPackets	<p>The number of bytes/packets that have been transmitted by (Tx) or received by (Rx) the specified node since the last time SandScript was evaluated for the flow.</p> <p>Has a value of 0 until the maximum subscriber lookup interval has been exceeded (no subscriber was mapped for the flow in time). When the lookup is first exceeded and the subscriber is not mapped, returns the total bytes/packets since the start of the flow up to that point, then is equivalent to Bytes/Packets. Always has a value of 0 if the subscriber is mapped.</p> <p>After a subscriber becomes mapped, or the interval is over, SubscriberBytes/Packets and NonSubscriberBytes/Packets will never both have non-zero values during the same SandScript run. If SubscriberBytes/Packets + NonSubscriberBytes/Packets is measured on a flow, the value of the measurement is equal to TotalBytes/Packets at the end of the flow.</p>	Integer
Flow.Client Server Subscriber.IpAssignment.IpPrefix	<p>The user prefix of the IP address that the client, server, or subscriber specify. The user prefix is the first N bits of the IP address, where N is the value returned by the Flow.&lt;node&gt;.IpAssignment.PrefixLength field. For example, this field can return 2001:db8:9567:28:: for a subscriber mapped with a 64-bit prefix.</p>	IP address
Flow.Client Server Subscriber.IpAssignment.PrefixLength	<p>The length of the prefix of the IP assignment.</p> <p>If a subscriber is assigned an IPv6 address with a 64-bit prefix, the value of this field is 64.</p>	Integer
Flow.{<Node>.<Direction>}.<StatType>	<p>&lt;Node&gt; is one of:</p> <ul style="list-style-type: none"> <li>Client—The endpoint that initiated the flow.</li> <li>Server—The endpoint that did not initiate the flow.</li> <li>Subscriber—The endpoint that represents the subscriber.</li> <li>Internet—The endpoint that does not represent the subscriber.</li> </ul> <p>The node is optional (if Direction is specified). If omitted, the sum of the statistics across both sides of the flow is returned. For example:</p> <pre>Flow.Tx.Bytes = Flow.Client.Tx.Bytes + Flow.Server.Tx.Bytes</pre> <p>Direction is one of:</p> <ul style="list-style-type: none"> <li>Rx—Counts that the Node will receive.</li> <li>Tx—Counts that the Node has transmitted, but the PTS may yet still shape or drop.</li> </ul> <p>The direction is optional (if Node is specified). If omitted, the sum of the statistics in both directions for the node specified is returned. For example:</p> <pre>Flow.Client.Bytes = Flow.Client.Rx.Bytes + Flow.Client.Tx.Bytes)</pre> <p>StatType is one of:</p>	Integer

Field	Description	Type
	<ul style="list-style-type: none"> <li>TotalBytes—Total number of bytes counted since the start of the flow.</li> <li>TotalPackets—Total number of packets counted since the start of the flow.</li> <li>Bytes—Bytes counted since the last SandScript run for this flow.</li> <li>Packets—Packets counted since the last SandScript run for this flow.</li> <li>ProtocolBytes/Packets—Similar to Bytes/Packets, except that they return 0 until the protocol has been recognized. The first time SandScript runs after the protocol is recognized, return the total since the start of the flow, then return deltas from that point on. Used for collecting statistics where the measurement somehow depends on the protocol, to guarantee that all bytes/packets are counted (using plain Bytes/Packets in this case would cause all bytes before protocol recognition completes to go uncounted, or to be counted against the wrong statistic).</li> <li>SubscriberBytes/Packets—Return values when the subscriber sending the flow is known.</li> <li>NonSubscriberBytes/Packets—Return values when the subscriber sending the flow is not known.</li> </ul> <p>SubscriberBytes/Packets and NonSubscriberBytes/Packets are similar to ProtocolBytes/Packets, except that the behaviour of these fields is also dependent on subscriber mapping. In particular, for any flow, the following equations will always hold once the flow is complete:</p> $\begin{aligned} & \text{sum}(\text{Flow.Rx} \text{Tx.SubscriberBytes}) \\ & + \text{sum}(\text{Flow.Rx} \text{Tx.NonSubscriberBytes}) \\ & \text{-----} \\ & \text{sum}(\text{Flow.Rx} \text{Tx.Bytes}) \end{aligned}$ $\begin{aligned} & \text{sum}(\text{Flow.Rx} \text{Tx.SubscriberPackets}) \\ & + \text{sum}(\text{Flow.Rx} \text{Tx.NonSubscriberPackets}) \\ & \text{-----} \\ & \text{sum}(\text{Flow.Rx} \text{Tx.Packets}) \end{aligned}$ <p>If the protocol of a flow is not yet known, these fields will return 0. The first non-zero value returned by these fields after the protocol is known depends on the subscriber mapping state. If the subscriber is not mapped, these fields will continue to return 0 until one of these instances occurs:</p> <ul style="list-style-type: none"> <li>The maximum subscriber lookup wait time is exceeded.</li> <li>The subscriber becomes mapped.</li> <li>The flow ends.</li> </ul> <p>If the subscriber becomes mapped before the flow ends or the wait time is exceeded, the SubscriberBytes and SubscriberPackets fields will return totals since the beginning of the flow as their first non-zero value. If the flow ends or the wait time is exceeded and the subscriber is still not mapped, NonSubscriberBytes and NonSubscriberPackets will return totals since the beginning of the flow as their first non-zero value. From every SandScript evaluation after the first non-zero value, these fields will return deltas since the previous SandScript run. If the subscriber is mapped when the fields are evaluated, these deltas will be returned by SubscriberBytes and SubscriberPackets. If the subscriber is not mapped, the deltas will be returned instead by NonSubscriberBytes and NonSubscriberPackets.</p>	

Field	Description	Type
	<b>Note:</b> In any given SandScript run, only one of SubscriberBytes/NonSubscriberBytes will be non-zero, depending on the subscriber mapping. The same is true for SubscriberPackets/NonSubscriberPackets.	
Flow	The ID of the current flow. This is most commonly used for defining tables unique by flow.	Integer
Flow.Age	The amount of time the flow has existed in seconds. Example: <code>expr(Flow.Age &lt; 3600)</code> is true if the flow is less than one hour old. When timeliness is important, use with the <code>reevaluate flow after action</code> (see the <code>reevaluate flow</code> action in the <i>SandScript Configuration Guide</i> ).	Integer
Flow.Client[Server Subscriber Internet.TcpTimestamp.Value	The last "Timestamp Value" field sent in the TCP Timestamp option from the specified node.	Integer
Flow.Client[Server Subscriber Internet.TcpTimestamp.Clock	The time the last Timestamp Value was sent from the specified node. The value is reported in microseconds with 100us precision. This value rolls over every 4.97 days, take this into account when using it in SandScript.	Integer
Flow.IpVersion	This field has a value of 4 if the flow is IPv4, or 6 if the flow is IPv6.	Integer
Flow.FromSubscriber ToSubscriber.IpV6TransitionProtocol	The IPv6 transition protocol being used for packets originating from a subscriber, or destined to a subscriber. This field has one of these values (or NULL if the flow does not utilize an IPv6 transition technology): <ul style="list-style-type: none"> <li>IPv6TransitionProtocol.Teredo</li> <li>IPv6TransitionProtocol.SixToFour</li> <li>IPv6TransitionProtocol.SixOverFour</li> </ul>	Integer
Flow.VoipProviderName	The name of the provider for this particular VoIP data flow. If the flow is not a VoIP flow, or if it is a VoIP control flow, the value of the field is NULL.	String
Flow.TotalDroppedBytes Flow.TotalDroppedPackets	The total number of bytes/packets that have been dropped since the start of the flow. Bytes/packets that were dropped due to shaping are not counted by these fields.	Integer
Flow.DroppedBytes Flow.DroppedPackets	The number of bytes/packets that have been dropped since the last time SandScript was evaluated for the flow. Bytes/packets that were dropped due to shaping are not counted by these fields.	Integer

## 2.1.2 Flow Action Fields

Flow action fields are used to determine if an action has been previously applied. The available fields include:

Field	Description	Type
Flow.IsBlocked	True if the flow has had the block action applied to it during the current or previous policy evaluation.	Boolean
Flow.WasBlocked	True if the flow had the block action applied to it during the previous policy evaluation.	Boolean

Field	Description	Type
Flow.NowBlocked	True if the flow has had the block action applied to it during the current policy evaluation.	Boolean
Flow.IsCaptive	True if the flow has had the captive_portal action applied to it during the current or previous policy evaluation.	Boolean
Flow.WasCaptive	True if the flow had the captive_portal action applied to it during the previous policy evaluation.	Boolean
Flow.NowCaptive	True if the flow has had the captive_portal action applied to it during the current policy evaluation.	Boolean
Flow.IsDiverted	True if the flow had the divert action applied to it during the current or previous policy evaluation	Boolean
Flow.IsDiverted. DestinationName	True if flow is currently being diverted to location specified by DestinationName.	Boolean
Flow.WasDiverted	True if the flow had the divert action applied to it during the current or previous policy evaluation.	Boolean
Flow.NowDiverted	True if the flow has had the divert action applied to it during the current or previous policy evaluation	Boolean
Flow.IsNew	True for the first time a flow is being evaluated, false otherwise.	Boolean
Flow.IsEnd	Returns true at the end of a flow the last time policy is evaluated, otherwise false.	Boolean
Flow.IsReevaluatingPolicy	Used to apply different policies to new versus existing flows. Returns false if the flow is having policy applied for the first time or true if policy is being applied by a mid-flow evaluation.	Boolean
Flow.IsReset	True if the flow has had the limit or tcp_reset action applied to it during the current or previous policy evaluation. Prevents counting flows in the measurement that have already been killed.	Boolean
Flow.WasReset	True if the flow had the limit or tcp_reset action applied to it during the previous policy evaluation.	Boolean
Flow.NowReset	True if the flow has had the limit or tcp_reset action applied to it during the current policy evaluation.	Boolean
Flow.IsShaped	True if the flow has had the shape action applied to it during the current or previous policy evaluation.	Boolean
Flow.WasShaped	True if the flow had the shape action applied to it during the previous policy evaluation.	Boolean
Flow.NowShaped	True if the flow has had the shape action applied to it during the current policy evaluation.	Boolean
Flow.IsTeed	True if the flow has had the tee action applied to it during the current or previous policy evaluation.	Boolean
Flow.WasTeed	True if the flow had the tee action applied to it during the previous policy evaluation.	Boolean
Flow.NowTeed	True if the flow has had the tee action applied to it during the current policy evaluation.	Boolean

Field	Description	Type
Flow.Client Server.PortRole	The configured role of the node, may be one of: <ul style="list-style-type: none"><li>PortRole.Subscriber</li><li>PortRole.Internet</li><li>PortRole.Unknown (port roles are not configured)</li></ul>	Boolean

### 2.1.2.1 Flow Field Usage

In this example, a limiter is set to monitor the number of connections for upstream Gnutella flows which are older than 15 minutes. The age of a flow can be compared with a constant value using an expression. Flows which have recently had the reset action applied are excluded using the condition `expr(not (Flow.IsReset))`. This limiter returns “true” when the number of connections exceeds eight. An external class must be defined in `subnets.txt` for it to be used as a condition.

```
limiter "GnutellaUp" 8 connections where \  
protocol "gnutella" and receiver class "external" and \  
expr((Flow.Age > 900) and not(Flow.IsReset))
```

In this example, the measurement `ShapedConnections` counts the number of connections where flows have been shaped:

```
measurement "ShapedConnections" connections where expr(Flow.IsShaped)
```

## 2.1.3 Flow Direction Fields

These fields are used by comparing them with the constants listed in [Flow Direction Constants](#) on page 21. For example:

```
measurement "BidirectionalP2PConnections" connections where \  
  expr(Flow.TransferDirection=TransferDirection.Bidirectional \  
  and is_p2p)
```

Notice the field `is_p2p` is an alias of peer-to-peer protocols.

Field	Description
Flow.RecognizerTransferDirection	The transfer direction determined by the protocol recognizer. It is sticky (doesn't change for the life time of the flow), even if the data transfer direction is reversed.
Flow.DataTransferDirection	The direction of each flow is constantly being monitored. This is the current direction as determined by this dynamic direction tracking functionality (always begins as unknown). The classification is based on the amount and ratio of traffic being sent in each direction. It is only useful with bidirectional protocols.
Flow.TransferDirection	The union of RecognizerTransferDirection and Flow.RecognizerTransferDirection. This is initially based on recognition, but is updated by the direction tracking. It is only useful with bidirectional protocols.

### 2.1.3.1 Flow Direction Constants

The flow direction constants are:

Field	Description
TransferDirection.Unknown	The direction of the flow is unknown.
TransferDirection.ClientToServer	The flow is unidirectional and the bulk of the data is being transferred from the client to the server.

Field	Description
TransferDirection.ServerToClient	The flow is unidirectional and the bulk of the data is being transferred from the server to the client.
TransferDirection.Bidirectional	The flow is bidirectional (a significant amount of data is being transferred in both directions).
TransferDirection.NoDirection TransferDirection.None	No data or an insignificant amount of data is being transferred.

### 2.1.3.2 Flow Direction Field Usage

In this example, a measurement of transfer bitrate is defined from internal clients to external servers, only for connections where the data flow is unidirectional. Both the internal and external classes must exist in `subnets.txt`. An expression is used to check the direction of flows.

```
measurement "InternalClientUp" bitrate where \
client class "internal" and server class "external" and \
expr(Flow.TransferDirection = ClientToServer)
```

## 2.1.4 Application Layer Fields

Application layer fields return information related to the application layer protocol.

Field	Description
Flow.ApplicationProtocol	Returns the application layer protocol identified in the flow. Can be compared against <code>Protocol.&lt;ProtocolName&gt;</code> constants such as <code>Protocol.HTTP</code> . For more information, see the <i>Loadable Traffic Identification Package (LTIPS) Guide</i> .
Flow.ApplicationProtocol.IsNew	Returns true the first time policy is evaluated for a flow after the application protocol becomes known.
Flow.ApplicationProtocol.Parent	Returns the application layer parent protocol identified in the flow. Must be compared with constant field <code>Protocol.&lt;ProtocolName&gt;</code> . See the Supported Protocols Guide for a list of supported protocols.
Flow.ApplicationProtocol.StatsProtocol	Returns the application layer protocol which the flow will be logged against. Must be compared with constant field <code>Protocol.&lt;ProtocolName&gt;</code> .
Protocol.<ProtocolName>	Constant field used for testing application protocol. See the Supported Protocols Guide for a list of supported protocols.
Flow.SessionProtocol	Returns the code of the protocol used to implement the application protocol (for example, Facebook's session protocol is HTTP).
Flow.ApplicationType	Returns the code used to represent the application protocol category of the flow. Can be compared against the <code>ApplicationType.&lt;Name&gt;</code> constants, such as <code>ApplicationType.PeerToPeer</code> . Use this field when defining and publishing a measurement unique by application protocol category, for example:  <pre>measurement ... unique by (Flow.ApplicationType)</pre> To see the full list of Application Types, run the <code>show traffic</code> CLI command.
Flow.IsApplicationType(string type1, {string type2, {type3, {...}}})	Returns true if the application protocol category is one of the specified strings. The function accepts either <code>ApplicationType.&lt;Name&gt;</code> constants, such as <code>ApplicationType.PeerToPeer</code> , or constant string names such as "PeerToPeer".

Field	Description
Flow.Application	An alias for Flow.ApplicationProtocol.StatsProtocol.
Flow.Application.IsNew	An alias for Flow.ApplicationProtocol.IsNew.
Flow.IsApplicationProtocol	An alias for Flow.IsApplication.

## 2.1.5 Transport Layer Fields

Transport layer fields return UDP and TCP information.

Field	Description	Type
Flow.TransportProtocol	Returns the integer value representing the protocol field from the IP header. Can be compared against an integer or one of these constants: <ul style="list-style-type: none"><li>• TransportProtocol.ICMP</li><li>• TransportProtocol.TCP</li><li>• TransportProtocol.UDP</li></ul>	Integer
Flow.Client[Server Subscriber Internet].Layer4Port	Returns the layer 4 (either UDP or TCP) port number of the specified node (Client or Server). Returns NULL if the transport protocol is not TCP or UDP.	Integer
Flow.Client[Server Subscriber Internet].TcpPort	Returns the TCP port number of the specified node (Client or Server). Returns NULL if the transport protocol is not TCP.	Integer
Flow.Client[Server Subscriber Internet].UdpPort	Returns the UDP port number of the specified node (Client or Server). Returns NULL if the transport protocol is not UDP.	Integer
Flow.Client[Server].Stream	Returns the reassembled TCP data at the start of a new flow.	String
Flow.Client.HandshakeRTT	Delay between the first SYN/ACK and the first ACK packets in microseconds. May be NULL (when the value is unknown). This is always NULL for non-TCP flows.	Integer
Flow.Server.HandshakeRTT	Delay between SYN and the first SYN/ACK packets in microseconds. May be NULL (when the value is unknown). This is always NULL for non-TCP flows. Note that for flows with repeat SYN packets, the value will include SYN retry delay(s).	Integer
Flow.Internet[Subscriber].HandshakeRTT	The delay between the first SYN/ACK and first ACK packets, or the delay between the first SYN and the first SYN/ACK packets as appropriate for the TCP handshake. May be NULL (when the value is unknown). Always NULL for non-TCP flows.	Integer
Flow.SynCount	The number of SYN packets seen for this TCP connection. This is normally 1, and no more than 15. This may return 0 for some TCP flows (but never NULL). This is always NULL for non-TCP flows.	Integer
Flow.SynAckCount	The number of SYN/SCK packets seen for this TCP connection. This is normally 1, and no more than 15. This may return 0 for some TCP flows (but never NULL). This is always NULL for non-TCP flows.	Integer
Flow.UdpResponseTime	Delay between the first packet from the initiator and first packet from the responder. May be NULL. This is always NULL for non-UDP flows. Note	Integer

Field	Description	Type
	that this is meaningful only for request-response UDP sessions (like DNS).	

## 2.1.6 Identification Fields

The fields used to identify a flow are:

Field	Description	Type
Flow.FromSubscriber ToSubscriber. Interface.BridgeGroup	Returns the bridge-group number of the PTS interface which originally received this flow.	Integer
Flow.FromSubscriber ToSubscriber. Interface.Element.ID	Returns an integer equivalent of SerialNumber. Note that, for elements recently discovered, this field may temporarily return 0 as the unique identifier is being calculated.	Integer
Flow.FromSubscriber ToSubscriber. Interface.Element.SerialNumber	Returns the serial number of the element which originally received this flow.	String
Flow.Client Server.ClassIpAddress	Returns the IP address that was extracted from the packet and used to determine the corresponding client/server network and policy class. In most cases these fields are identical to Flow.Client Server.IpAddress, but may differ when some tunneling or IPv6 transitional protocols are used. The PTS attempts to use the most appropriate IP addresses for determining the physical location of the actual end-point. The IP addresses of the tunnel end-points may be used when some variant of IP-in-IP tunneling is found and the "skip IP" mode is NOT enabled for that protocol, meaning the tunnel is configured by the subscriber, not the Internet service provider. The subscriber-side ClassIpAddress always comes from the outer IP header of the tunnel packet. The internet-side ClassIpAddress comes from the outer IP of the tunneled packet, unless it is a private IPv4 address, in which case the inner IP is used. If both inner and outer addresses are private, the outer is used.	IP address
Flow.Client Server Subscriber  Internet.IpAddress	Returns the IP address of the specified node.	IP address
Flow.Site	Returns the site of the flow, based on the session qualifiers configuration.	Integer
Flow.Subscriber Internet Client  Server.Rx.Interface.Slot Flow.Subscriber Internet Client  Server.Rx.Interface.Port	Returns the slot and port number of the PTS interface which originally received this flow. These field represent the outermost interfaces in the cluster for the flow, and should be used in conjunction with the Element.SerialNumber or Element.ID fields as described above.	Integer
Flow.Client Server Internet  Subscriber.Rx Tx.Vlan	Returns the full 16-bit VLAN tag for the specified direction of traffic with respect to the specified node. The fields include both the VLAN ID and the priority bits of the vlan tag. For example, Flow.Client.Tx.Vlan is the VLAN used for packets transmitted from the client.	Integer
Flow.Client Server Internet  Subscriber.Rx Tx.ServiceVlan	In 802.1ad QinQ VLAN deployments, returns the full 16-bit service vlan tag for the specified direction of traffic with respect to the specified node. The fields include both the VLAN ID and the priority bits of the vlan tag.	Integer



Field	Description	Type
	For example, <code>Flow.Client.Tx.ServiceVlan</code> is the QinQ service VLAN used for packets transmitted from the client.	

## 2.1.7 Settable Flow Fields

These flow fields expose additional information about flows. They can be compared with constraints to apply dynamic SandScript based on flow conditions. Each flow has a settable integer field that can be used to store user-defined state. The values of these fields are retained between policy evaluations of the same flow.

The flow fields are:

Field	Description	Type
<code>Flow.UserInteger</code>	Single 64-bit integer for user-defined state.	Integer
<code>Flow.UserInteger.Long[0-1]</code>	Two 32-bit integers for user-defined state.	Integer
<code>Flow.UserInteger.Short[0-3]</code>	Four 16-bit integers for user-defined state.	Integer
<code>Flow.UserInteger.Byte[0-7]</code>	Eight 8-bit integers for user-defined state.	Integer
<code>Flow.UserInteger.Bit[0-63]</code>	Sixty-four single bit integers for user-defined state.	Boolean
<code>Flow.Debug.TraceEnabled</code>	Setting this field turns on internal tracing for the flow.	Boolean

These fields are mutually exclusive. For example, if you set `Flow.UserInteger`, then set `Flow.UserInteger.Short2`, the value of `Flow.UserInteger` will be modified and retained between policy evaluations of the same flow.

To modify a `UserInteger` field, use the set action:

```
set Flow.UserInteger = expression
```

Where expression can be any expression, variable, or constant.

For more information on the set action, see the *SandScript Configuration Guide*.

## 2.2 Tee and Divert Fields

Divert and tee actions are applied to flows once they have been identified. Information about the ongoing tee and divert actions are extracted using fields. These fields are used to create measurements for flows to destinations specified in SandScript.

The measurements can then be published and examined in Network Demographics. To view information about a published expression, run the `show policy publish` CLI command. For more information about tee and divert, see the *PTS SandScript Configuration Guide*.

The tee and divert fields are:

Field	Description	Type
<code>Destination.&lt;Destination Name&gt;.BytesTeed</code> <code>Destination.&lt;Destination Name&gt;.PacketsTeed</code>	Returns the number of bytes or packets teed to the destination. Each defined tee destination has these associated fields, where <code>DestinationName</code> is the name of the destination.	Integer

Field	Description	Type
Destination.<Destination Name>.Flows	Returns the number of active flows currently being diverted to the destination. Each defined divert destination has this associated field, where DestinationName is the name of the destination.	Integer
DestinationGroup.<Group Name>.<Destination Name>.IsUp	Provides access to the health status of destinations that are in a health-checked group.	Boolean
Flow.Stream.SaveData	SaveData applies only to TCP flows. When SandScript is evaluated, SaveData is initialized to "false" and you can set the SaveData field to "true". Data is saved if this field is "true" or if the protocol is not yet recognized. At each SandScript evaluation in flow context, SaveData is initialized to "false", and you need to set SaveData to "true" if you need the data later. At the end of each SandScript evaluation, if the value is "false" then the data is not saved.	Boolean

## 2.3 HTTP Fields

HTTP fields are string fields which return specific properties of HTTP flows by exposing fields in the client request packets. After identifying the HTTP flow, divert or tee actions may be applied to the flows. HTTP fields are only used as conditions by string comparison.

### 2.3.1 HTTP Fields Usage

Use fields to access the information from the headers, and expressions to identify and manipulate the header fields.

For more information, see [Functions](#) on page 58. All fields return strings that are constant for one flow.

HTTP fields are of limited use when creating measurements. The fields would more likely be of use when determining which flows meet certain conditions to be diverted or teed (or have any other action applied). For example, the `UserAgent` HTTP field returns information relevant to the type of browser being used.

This example tees common image files from specified browsers to a destination *destination1*. Notice the usage of the `ToLower()` function to ensure that file type extensions are in lowercase prior to comparison with the images list of extensions. This avoids the need to specify both `bmp` and `BMP`.

```
define "images" = "\S*\.(bmp|gif|jpe{0,1}g|png)\S*"
define "browsers"="(Firefox|MSIE)\S*"

destination "destination1" tee next_hop XXX.XX.X.XX payload ether

if expr(Flow.ApplicationProtocol = Protocol.Http) \
  and expr(Flow.Client.Stream.HTTP.Host is not null) \
  and expr(regex(images, ToLower(Flow.Client.Stream.HTTP.Resource))) \
  and expr(regex(browsers, Flow.Client.Stream.HTTP.UserAgent)) \
then tee from client destination "destination1" \
  and tee from server destination "destination1"
measurement Destination.destination1.PacketsTeed
```

## 2.3.2 Client Request Packets

These fields provide access to the fields in HTTP client request packets. They only return meaningful values for HTTP flows. For other types of traffic, they return NULL.

Field	Description	Type
Flow.Client.Stream. HTTP.Accept	The list of MIME types acceptable by the browser.	String
Flow.Client.Stream. HTTP.Accept Charset	The list of character sets supported by the client's browser.	String
Flow.Client.Stream. HTTP.Accept Encoding	The request encoding. A typical value for this field is gzip or deflate, allowing the web server to send compressed contents.	String
Flow.Client.Stream. HTTP.Accept Language	The languages acceptable by the browser.	String
Flow.Client.Stream. HTTP.Authorization	Authentication information.	String
Flow.Client.Stream. HTTP.Command	The request method: GET, HEAD, or POST	String
Flow.Client.Stream. HTTP.Content Length	Indicates the size of the body in bytes.	String
Flow.Client.Stream. HTTP.Content Type	Indicates the media type of the body.	String
Flow.Client.Stream. HTTP.Cookie	The cookie field offered by the browser. For example, Cookie: name=value.	String
Flow.Client.Stream. HTTP.Host	The host being requested.	String
Flow.Client.Stream. HTTP.HostPort	If the request contains a proxy-type request, then the HostPort field will be 8080. This is exposed because it may be different than the TCP port number. For example:  GET http://www.domain.com:8080/ index.htm HTTP/1.1	Integer
Flow.Client.Stream. HTTP.Malformed	Set to true if flow is detected as malformed. No other HTTP fields will be available if this is set to true.	Boolean
Flow.Client.Stream. HTTP.Referer	Web page where the request originated (or was referred). Note the spelling of Referer without a double 'r'.	String
Flow.Client.Stream. HTTP.Resource	The resource portion of the URL.	String
Flow.Client.Stream. HTTP.TransactionNumber	Uniquely identifies the request and corresponding response in a persistent HTTP connection.	String
Flow.Client.Stream. HTTP.UserAgent	Client web browser and operating system version and specifications.	String
Flow.Client.Stream. HTTP.URL	The URL in the HTTP client request.	String
Flow.Client.Stream. HTTP.Version	The string "HTTP/1.0" or "HTTP/1.1" or other if invalid protocol. Is found after white-space following the resource field, and ending at the new-line.	String
Flow.Client.Stream. HTTP.XForwardedFor	Used by some caches to indicate original client.	String

Field	Description	Type
Flow.Client.Stream. HTTP.XWap ClientIp	Used by some WAP proxies to indicate original client.	String
Flow.Client.Stream. HTTP.XNetworkInfo	Used to indicate original client.	String
Flow.Client.Stream. HTTP. StreamIsEnd	True when a particular StreamID is being exposed to policy for the last time. This allows the policy writer to clean up any state they might have been using for the duration of a particular stream.	Boolean

## 2.3.3 Server Response Packets

These fields provide access to the fields in HTTP server response packets. They only return meaningful values for HTTP flows. For other types of traffic they return NULL.

Field	Description	Type
Flow.Server.Stream. HTTP.CanHttpResponse	True, if the http_response command can be issued on the current response.	Boolean
Flow.Server.Stream. HTTP.Connection	Options specified by the sender for a particular connection.	String
Flow.Server.Stream. HTTP.ContentLength	Content-length field in the response.	Integer
Flow.Server.Stream. HTTP.ContentType	Indicates the media type of the body.	String
Flow.Server.Stream. HTTP.FailedHttpResponse	True, if http_response was issued and could not be completed	Boolean
Flow.Server.Stream. HTTP.Malformed	True, if flow is detected as malformed. No other HTTP fields will be available if this is set to true.	Boolean
Flow.Server.Stream. HTTP.Server	Contains information about the software used by the originating server that handled the request.	String
Flow.Server.Stream. HTTP.StatusCode	The HTTP Status Code in the response.	Integer
Flow.Server.Stream. HTTP.TransactionNumber	Uniquely identifies the request and corresponding response in a persistent HTTP connection.	Integer
Flow.Server.Stream. HTTP.StreamIsEnd	True when a particular StreamID is being exposed to policy for the last time. This allows the policy writer to clean up any state they might have been using for the duration of a particular stream	Boolean
Flow.Client Server Subscriber Internet.Stream.HTTP. Tx.<StatType>	<StatType> is one of: <ul style="list-style-type: none"><li>• Bytes: counts all bytes</li><li>• PayloadBytes: counts everything past the layer 5 header (application protocol)</li><li>• OverheadBytes: Counts everything up to and including the layer 5 header, including tunnel headers</li><li>• OverlapBytes: counts retransmitted TCP bytes</li></ul>	String

Field	Description	Type
	<p>For each &lt;StatType&gt; there exists a Total, Subscriber, and NonSubscriber variant (for example, TotalPayloadBytes, SubscriberPayloadBytes, NonSubscriberPayloadBytes). Their behavior is similar to that of the Flow.Client Server Subscriber Internet.Tx.TotalBytes, SubscriberBytes, and NonSubscriberBytes fields.</p> <p>Note: For Total fields, the count is from the start of the transaction, not the start of the flow.</p> <p>Note: NonSubscriber fields start to return non-zero values if a transaction ends prior to the mapping of a subscriber and the maximum subscriber lookup has not been exceeded.</p> <p>It is always true that Bytes = PayloadBytes + OverheadBytes + OverlapBytes. For pipelined flows, Overhead bytes will always be attributed to the first transaction of the packet. This means that pipelined transactions could have 0 overhead bytes. For bytes that are both Overlap and Payload, bytes are counted only as Overlap. For bytes that are both Overhead and Overlap, bytes are counted only as Overhead.</p>	
Flow.Client Server.Stream. HTTP. Userdefinedstring. <field>	<p>Allows access to any HTTP field where FieldName is the case insensitive name of the particular field. To use otherwise invalid characters in the field name, use single quotes. For example:</p> <p>'Flow.Client.Stream.HTTP.Userdefinedstring.Content-Length'.</p>	String

## 2.4 RTSP Fields

Real Time Streaming Protocol (RTSP) fields have specific properties of RTSP flows by exposing fields in the client request and server response packets. RTSP fields are used as conditions by string comparisons. Regular expression functions can be used on RTSP fields.

Note that all fields are only available for RTSP control flows. The only field available for RTSP data flows, Real Time Protocol (RTP) or Real Time Control Protocol (RTCP), is Flow.Client|Server.Stream.RTSP.StreamID.

Field	Description	Type
Flow.Client.Stream.RTSP. Method Flow.Client.Stream.RTSP. URI Flow.Client.Stream.RTSP. Version	These provide access to fields in the RTSP client request stream. They only return meaningful values for an RTSP flow.	String
Flow.Server.Stream.RTSP. StatusCode Flow.Server.Stream.RTSP. ReasonPhrase Flow.Server.Stream.RTSP. Version	These provide access to fields in the RTSP server response stream. They only return meaningful values for an RTSP flow.	String
Flow.Client Server.Stream. RTSP.StreamID	These provide a unique, local identifier for an RTSP/RTP/RTCP stream. The field is meant to be used to associate an RTSP stream with corresponding RTP and RTCP streams in SandScript. This is available for RTSP, RTP and RTCP flows. Note that an RTSP flow may have multiple, simultaneous streams.	Integer
Flow.Client Server.Stream. RTSP.Malformed	The field is used to provide status of whether the RTSP flow is considered malformed by the internal parser. If this field returns true, other RTSP fields may be unavailable.	Boolean

Field	Description	Type
Flow.Client Server.Stream. RTSP. Userdefinedstring.<FieldName>	Allows access to any RTSP field where FieldName is the case insensitive name of the field. To use otherwise invalid characters in the field name, use single quotes. For example: 'Flow.Client.Stream.RTSP.Userdefinedstring.Content-Length'.	String
Flow.Client Server.Stream. RTSP.StreamIsEnd	True when a particular StreamID is being exposed to SandScript for the last time. This allows you to clean up any state that might have been used for the duration of a particular stream.	Boolean
Flow.Client Server Subscriber  Internet.Stream.RTSP. Tx.Bytes	The number of bytes that have been transmitted by the specified node since the last time SandScript was evaluated for the flow.	Integer
Flow.Client Server Subscriber  Internet. Stream.RTSP. Tx.SubscriberBytes	The number of bytes that have been transmitted by the specified node since the last time SandScript was evaluated for the flow. Returns 0 before the subscriber becomes mapped. When the subscriber first becomes mapped, returns the total bytes/packets since the start of the flow up to that point, then is equivalent to .Bytes. This is intended to be used in subscriber-dependant SandScript (for example counting bytes per subscriber).	Integer
Flow.Client Server Subscriber  Internet. Stream.RTSP. Tx.NonSubscriberBytes	The number of bytes that have been transmitted by the specified node since the last time SandScript was evaluated for the flow. Returns 0 until the maximum subscriber lookup has been exceeded (no subscriber was mapped for the flow in time). When the lookup is first exceeded and the subscriber is not mapped, returns the total bytes/packets since the start of the flow up to that point, then is equivalent to .Bytes. Always returns 0 if the subscriber is mapped. Note that SubscriberBytes/Packets and NonSubscriberBytes/Packets will never both return non-zero values during the same SandScript run.	Integer

## 2.5 RTMP Fields

Real Time Messaging Protocol (RTMP) fields are a collection of fields which return specific properties of RTMP flows by exposing the fields in the first request of the connection.

These fields are useful for classifying video flows by things such as codec, URL of the player, URL of the video, and so on. These fields are only available on non-encrypted RTMP flows and are only available when the flow is being analyzed.

Field	Description	Type
Flow.Client.Stream. RTMP.MessageType	Exposes the name of the current RTMP message. When this field is non-null, the RTMP user-defined fields described are available to SandScript.	String
Flow.Client.Stream. RTMP.Malformed	Indicates whether the RTMP flow is considered malformed by the internal parser. If this field returns true, other RTMP fields may be unavailable.	Boolean
Flow.Client.Stream.RTMP. Userdefinedstring. <FieldName> Flow.Client.Stream.RTMP. Userdefinedfloat. <FieldName> Flow.Client.Stream.RTMP. Userdefinedboolean. <FieldName>	These fields allow access to any RTMP field where FieldName is the case insensitive name of the particular field. The different types of user-defined fields support the different types of fields that are available in RTMP flows. To access the value, both the name and the type must match the field in the flow. For example, to access a string field named swfUrl, use <code>Flow.Client.Stream.RTMP.UserDefinedString.swfUrl</code> .	String

## 2.6 Generic Fields

The generic fields are:

Field	Description	Type
Flow.Client[Server] Subscriber[Internet.Stream. Tx.<StatType>	Returns the sum of the corresponding HTTP, RTSP and RTMP stat fields (only one will be non-zero). Useful for getting bytes counted by an analyzer without knowing which analyzer is being used. <StatType> is one of: Bytes, SubscriberBytes, NonSubscriberBytes	Integer

## 2.7 Video Fields

These fields expose extra information about video and audio streams.

These fields require the video analyzer to be enabled using the `analyze analyzer "video"` policy action. The video fields have their own policy scope, which means they cannot be used with fields in other policy scopes, such as flow or subscriber. The smooth streaming analyzer parses videos conforming to the Microsoft SmoothStreaming specification to extract video information and detect shifts in bitrates of media streams.

The video fields are:

Field	Description	Type
Video.Id	Returns an ID for the current video. This number is unique per PTSD instance.	Integer
Video.Malformed	Returns true if a video was found but an error occurred while parsing. Fields may or may not be available for this video.	Boolean
Video.Flow	A flow ID (See <a href="#">Flow Fields</a> on page 16) for the flow that was used to transmit the video.	Integer
Video.Container.Name	The container format of the video.	Integer or String
Video.Container.<Container> {.Name}	Returns a constant integer or string for each supported video container. Currently supported containers are: <ul style="list-style-type: none"><li>• FLV - Flash Video</li><li>• MP4 - Mpeg-4 Part 14</li><li>• ASF - Advanced Systems Format (Windows Media)</li><li>• RTMP - Real Time Messaging Protocol (Adobe)</li><li>• RTSP - Real Time Streaming Protocol</li><li>• Other</li><li>• Unknown</li></ul>	Integer or String
Video.VideoCodec.Name	The codec used for the video stream.	Integer or String
Video.VideoCodec. <VideoCodec>{.Name}	Returns a constant integer or string for each recognized video codec. Currently recognized video codecs are: <ul style="list-style-type: none"><li>• H264 - Mpeg-4 Part 10, aka AVC or H.264</li><li>• VP6 - On2 VP6 (Commonly used with Flash)</li><li>• VP8 - Google VP8 (WebM)</li><li>• WMV - Windows Media Video</li></ul>	Integer or String

Field	Description	Type
	<ul style="list-style-type: none"><li>• SorensonSpark - also known as Sorenson H.263 or FLV1 (commonly used with Flash)</li><li>• Mpeg4ASP - Mpeg-2 Part 2, aka ASP/XviD/DivX</li><li>• RealVideo - RealNetworks video format</li><li>• Other</li><li>• Unknown</li><li>• None</li></ul>	
Video.AudioCodec.Name	The codec used for the audio stream.	Integer or String
Video.AudioCodec.<AudioCodec>{.Name}	Returns a constant integer or string for each recognized audio codec. Currently recognized audio codecs are: <ul style="list-style-type: none"><li>• MP3 - Mpeg-2 Audio Layer III</li><li>• AAC - Advanced Audio Coding, Mpeg-4 Part 14</li><li>• AMR - Adaptive Multi-Rate, used for storing speech audio</li><li>• WMA - Windows Media Audio</li><li>• Mpeg4ASP - Mpeg-2 Part 2, aka ASP/XviD/DivX</li><li>• RealAudio - RealNetworks audio format</li><li>• Other</li><li>• Unknown</li><li>• None</li></ul>	Integer or String
Video.EncodedVideoBitrate	Returns the average encoded bitrate (in kbps) for the video stream, as advertised by the container.	Integer
Video.EncodedAudioBitrate	Returns the average encoded bitrate (in kbps) for the audio stream, as advertised by the container.	Integer
Video.HorizontalResolution	Returns the horizontal resolution (width) of the video in pixels.	Integer
Video.VerticalResolution	Returns the vertical resolution (height) of the video in pixels.	Integer
Video.Duration	Returns the duration of the video in seconds as advertised by the container.	Integer
Video.AdaptiveTechnology.Name	Returns 'SmoothStreaming' if the video is being parsed by the smooth streaming analyzer, which uses Adaptive Video Streaming Technology, returns 'None' for other video analyzers.	String

## 2.8 Analyzer Flow Fields

This PTS release supports these analyzer flows:

- [SSL Analyzer Fields](#) on page 32
- [Instant Messaging Analyzer Fields](#) on page 33
- [SIP Analyzer Fields](#) on page 34

### 2.8.1 SSL Analyzer Fields

These fields expose information about Secured Socket Layer (SSL) analyzer flows.



Add these fields to SandScript and execute `svreload` to enable SSL analyzer. You can then obtain NDS report for flows, as described in the *Network Demographics User Guide*.

The SSL analyzer fields are:

Field	Description	Type
Flow.Application.SSL. ClientHello	The first packet in a SSL handshake is ClientHello. When the Client Hello packet is recognized, this field value is set to "true".	Boolean Default Value: NULL
Flow.Application.SSL. ServerHello	The first packet from the server in the SSL handshake is Server Hello. When the ServerHello packet is recognized, this field value is set to "true".	Boolean Default Value: NULL
Flow.Application.SSL. ClientHello.ServerName	This field exposes SSL ClientHello packet extension server name field content.	String
Flow.Application.SSL.Server.Certificate.Subject.CommonName	This field exposes SSL server certificate packet's subject common name. All certificates common names are concatenated with ";" .	String

## 2.8.2 Instant Messaging Analyzer Fields

These fields expose information about WhatsApp and Viber analyzer flows.

Add these fields to SandScript and execute `svreload` to enable the WhatsApp and Viber analyzers. You can then obtain NDS report for the flows, as described in the *Network Demographics User Guide*.

The instant messaging analyzer fields are:

Field	Description	Type
Flow.Application.IM. Device	The device type running the IM application. Example: iPhone, Android, Blackberry.	String
Flow.Application.IM. Rx Tx.TotalMessages	The total number of messages sent (Tx) or received (Rx) by the subscriber since the beginning of the flow. The messages are the sum total of all the different message types such as text, pictures, and video.	Integer
Flow.Application.IM. Rx Tx.TotalMultimediaMessages	The total number of multimedia messages sent (Tx) or received (Rx) by the subscriber since the beginning of the flow. The messages are the sum total of all the different multimedia message types such as pictures, audio, and video.	Integer
Flow.Application.IM. Rx Tx.Messages	The total number of messages sent (Tx) or received (Rx) by the subscriber since the last time policy was evaluated for the flow.	Integer
Flow.Application.IM. Rx Tx.MultimediaMessages	The total number of multimedia messages sent (Tx) or received (Rx) by the subscriber since the last time policy was evaluated for the flow.	Integer
Flow.Application.IM.VoiceCalls	Returns the number of voice calls performed by the IM application in both directions (calling and called party). Returns NULL if the application does not support the voice call feature. This field is applicable for Viber analyzer only.	Integer

## 2.8.3 SIP Analyzer Fields

These fields expose information about Session Initiation Protocol (SIP) flows.

Add these fields to SandScript and execute `svreload` to enable the SIP analyzer. You can then obtain NDS report for flows, as described in the *Network Demographics User Guide*.

The SIP analyzer fields are:

Field	Description	Type
Flow.Application.Voip.Request	This field is set to <b>true</b> for a VoIP call request.	Boolean
Flow.Application.Voip.Response	This field is set to <b>true</b> for a VoIP call response.	Boolean
Flow.Application.Voip.ClientIsCaller	This field is set to <b>true</b> if the originator of the VoIP call is the one who initiated the flow (data or control) else it is set to false.	Boolean
Flow.Application.Voip.FromAddress	This field exposes the packet originator address.	String
Flow.Application.Voip.ToAddress	This field exposes the packet destination address.	String
Flow.Application.Voip.UserAgent	This field exposes the user agent of the packet originator. For example, an email reader is a Mail User Agent.	String
Flow.Application.Voip.CallId	This field exposes a unique call ID to identify the control and data flows.	String

## 2.8.4 WAP1 Analyzer Field

This field exposes information about Wireless Application Protocol 1 (WAP1) analyzer flows. WAP is a technical standard for accessing information over a mobile wireless network. A WAP browser is a web browser for mobile devices, such as mobile phones, that use the protocol.

Add these fields to SandScript and execute `svreload` to enable WAP1 analyzer. You can then obtain NDS report for flows, as described in the *Network Demographics User Guide*.



**Note:**

WAP1 analyzer is enabled on PTS 6.20 or later versions only.

The WAP1 analyzer field is:

Field	Description	Type
Flow.Application.WAP1.URL	This field exposes WAP1 browsed URLs.	String

## 2.9 Streaming Fields

These fields expose information about streaming flows.

Add these fields to streaming SandScript and execute `svreload` to enable the it. You can then obtain NDS report for flows, as described in the *Network Demographics User Guide*.



**Note:**

Streaming SandScript is enabled on PTS 6.20 or higher version only.

The streaming SandScript fields are:

Field	Description	Type
Flow.Application.Streaming.Live	Streaming flow is classified as Live or On Demand. This field is set to <b>true</b> when the streaming is Live; otherwise it is set to false.	Boolean type
Flow.Application.Streaming.VOD	Streaming flow is classified as Live or On Demand. This field is set to <b>true</b> when the streaming is On Demand; otherwise it is set to false.	Boolean type
Flow.Application.Streaming.P2P	Streaming flow is from provider servers or of P2P type. This field is set to <b>true</b> when streaming is of P2P type; otherwise it is set to false.	Boolean type

## 2.10 Limiters

Each limiter defined in policy has an associated field, where `<LimiterName>` is the name of the limiter and `<PriorityName>` is the name of a priority level (if one or more are declared) defined in `policy.conf`.

For more information on configuring limiters, see the *PTS SandScript Configuration Guide*.

### 2.10.1 Limiter Fields

These fields can be used with limiters:

Field	Description	Type
Limiter.<LimiterName>.Limit Limiter.<LimiterName>.<PriorityName>. Limit	Returns true if a flow should be limited (the threshold has been exceeded) and false otherwise.	Boolean
Limiter.<LimiterName>.LimitedFlows Limiter.<LimiterName>.<PriorityName>. LimitedFlows	Returns the number of flows that have been limited by the limiter <code>&lt;LimiterName&gt;</code> , (and priority <code>&lt;PriorityName&gt;</code> if specified) equivalent to the number of times the Limit field was evaluated for the limiter and returned true.	Integer

### 2.10.2 Limiter Fields Usage

This example tests Gnutella flows for number of connections, and returns “True” when it exceeds 20. The Limiter field is used to evaluate when the limiter returns true, and applies the TCP reset action accordingly. Note that the “internal” class must be defined in `subnets.txt`. Notice that expressions can also be used in the limiter definition; in this case as a condition with `expr(not(Flow.IsReset))`.

```
limiter "GnutellaUp" limit 20 connections where protocol "gnutella" \  
and transfer uni and sender class "internal" and \  
expr(not(Flow.IsReset)) \  
if expr(Limiter.GnutellaUp.Limit) and protocol "gnutella" \  
then tcp_reset
```

## 2.11 Measurements

Each measurement defined in a policy has an associated field, where `<MeasurementName>` is the name of the measurement. The field returns the current value of the measurement based on the measurement type.

For more information on using measurements, see *Measurements* in the *PTS SandScript Configuration Guide*.

### 2.11.1 Measurement Fields

Measurements are taken over time:

- Measurements with an “over” interval specified expose values from both the current and previous interval.
- The regular (non-“current”) fields return the value from the end of the previous interval.
- The “current” fields return the value so far, from the current interval. “Current” fields are not available for Top-N measurements, or measurements over infinity.

Field	Description	Type
Measurement.<MeasurementName>	Each measurement defined in SandScript has an associated field, where <code>&lt;MeasurementName&gt;</code> is the name of the measurement defined in <code>policy.conf</code> . The field returns the current value of the measurement based on the measurement type. <ul style="list-style-type: none"><li>• connections: current number of active connections</li><li>• bitrate: current bitrate, in bits per second</li><li>• hosts: current number of hosts</li><li>• expression: current value of the expression</li><li>• sum or generic: the previous interval value of the measurement if it is “over” some interval, the current value otherwise, settable for generic measurements</li><li>• histogram: returns NULL. Histogram measurements do not have a single value</li></ul>	Integer
Measurement.<MeasurementName>.Peak	Returns the peak value of the measurement. If unique-by, returns the peak value of the highest instance measured.	Integer
Measurement.<MeasurementName>.Instances	The total number of instances being counted by a unique-by measurement. The number of instances may be larger than the number of the unique-by qualifier. For example, for a measurement unique-by subscriber and the subscriber has multiple IPs, each IP could be assigned to different processing modules. Since each subscriber instance is counted for each assignment, there are more instances counted than there are unique-by qualifiers.	Integer
Measurement.<MeasurementName>.PeakInstances	Returns the peak value of Measurement.<MeasurementName>.Instances	Integer
Measurement.<MeasurementName>.Total	The sum of the values for all instances of a unique-by measurement.	Integer
Measurement.<MeasurementName>.PeakTotal	Returns the peak value of Measurement.<MeasurementName>.Total	Integer

Field	Description	Type
Measurement.<MeasurementName>. IsTop	Returns true if the current instance was in the top N during the previous measurement interval, otherwise returns false. Available for “top” measurements only.	Boolean
Measurement.<MeasurementName>. Min	Returns the floating point minimum value observed by the measurement. Available for histogram measurements only.	Float
Measurement.<MeasurementName>. OverallMin	Returns the overall minimum value observed by all instances of the measurement. Available for unique-by histogram measurements only	Float
Measurement.<MeasurementName>. Max	Returns the floating point maximum value observed by the measurement. Available for histogram measurements only.	Float
Measurement.<MeasurementName>. OverallMax	Returns the overall maximum value observed by all instances of the measurement. Available for unique-by histogram measurements only	Float
Measurement.<MeasurementName>. Sum	Returns the floating point sum of all values observed by the measurement. Available for histogram measurements only.	Float
Measurement.<MeasurementName>. TotalSum	Returns the sum of all values observed by all instances of the measurement. Available for unique-by histogram measurements only.	Float
Measurement.<MeasurementName>. Samples	Returns the number of values observed by the measurement. Available for histogram measurements only	Float
Measurement.<MeasurementName>. TotalSamples	Returns the number of values observed by all instances of the measurement. Available for unique-by histogram measurements only.	Float
Measurement.<MeasurementName>. Current	Returns the value of the measurement from the current interval. Settable for generic measurements with an “over” interval.	Integer
Measurement.<MeasurementName>. CurrentPeak	Returns the peak value of the measurement from the current interval.	Integer
Measurement.<MeasurementName>. CurrentInstances	The total number of instances being counted by a unique-by measurement from the current interval.	Integer
Measurement.<MeasurementName>. CurrentPeakInstances	The peak number of instances being counted by a unique-by measurement from the current interval.	Integer
Measurement.<MeasurementName>. CurrentTotal	The sum of the values for all instances of a unique-by measurement from the current interval.	Integer
Measurement.<MeasurementName>. CurrentPeakTotal	The peak sum of the values for all instances of a unique-by measurement from the current interval.	Integer
Measurement.<MeasurementName>. CurrentMin	Returns the floating point minimum value observed by the measurement from the current interval. Available for histogram measurements only.	Float
Measurement.<MeasurementName>. CurrentOverallMin	Returns the overall minimum value observed by all instances of the measurement from the current interval. Available for histogram measurements only.	Float

Field	Description	Type
Measurement.<MeasurementName>.CurrentMax	Returns the floating point maximum value observed by the measurement from the current interval. Available for histogram measurements only.	Float
Measurement.<MeasurementName>.CurrentOverallMax	Returns the overall maximum value observed by all instances of the measurement from the current interval. Available for histogram measurements only.	Float
Measurement.<MeasurementName>.CurrentSum	Returns the floating point sum of all values observed by the measurement from the current interval. Available for histogram measurements only.	Float
Measurement.<MeasurementName>.CurrentTotalSum	Returns the sum of all values observed by all instances of the measurement from the current interval. Available for histogram measurements only.	Float
Measurement.<MeasurementName>.CurrentSamples	Returns the number of values observed by the measurement from the current interval. Available for histogram measurements only.	Float
Measurement.<MeasurementName>.CurrentSamples	Returns the number of values observed by all instances of the measurement from the current interval. Available for histogram measurements only.	Float

## 2.11.2 Measurement Fields Usage

Fields are used to access existing measurements and information about them.



### Example:

This example defines a measurement “BittorrentUpPercentage” which expresses the upstream bandwidth consumed by BitTorrent as a percentage of the total available bandwidth (constant 15000000). An expression must be used to create the percentage measurement. Note that only scalar integer expressions can be used to create measurements. These expressions are not specific to any packet, flow, or subscriber.

```
measurement "UploadToExternal" bitrate where receiver class "external"
measurement "UpToExternalPercentageTot" \
  expr (Measurement.UploadToExternal*100/15000000)
```

## 2.12 Classifiers

Classifiers are used to apply labels to SandScript contexts such as flows or subscribers, and/or for categorization into meaningful groups.

Traffic that is measured unique-by one or more classifiers can be published and used to generate custom reports in Network Demographics. A classifier's value is set and checked via its field.

Field	Description	Type
Classifier.<ClassifierName> Flow.Classifier.<ClassifierName>	Used to set or check the value of a classifier, where <ClassifierName> is the name of the classifier. Using the Flow prefix restricts the field to flow scope.	Policy-defined
Classifier.<ClassifierName>.Name Flow.Classifier.<ClassifierName>.Name	Represents the name of an enumerated classifier. Using the Flow prefix restricts the field to flow scope.	String

Field	Description	Type
Classifier.<ClassifierName>.<ValueName>	A constant field used to set the value of an enumerated classifier. One of these is defined for each value specified in an enumerated classifier definition.	Integer

## 2.13 Timers

SandScript timers generate an event at some point in the future. When the event occurs, SandScript rules available in the timer's context are evaluated. See the Timers section of the *SandScript Configuration Guide* for additional information.

Field	Description	Type
Timer.<TimerName>.Expired	Returns true if the current SandScript policy run was caused because timer <TimerName> expired, otherwise this is false. Use this field to cause SandScript to run when a timer expires.	Boolean
Timer.<TimerName>.IsArmed	Returns true if the timer <TimerName> is currently armed. Otherwise it returns false.	Boolean
Timer.<TimerName>.ExpiryTime	Returns a timestamp for the time when timer <TimerName> will expire. You can interpret the integer as the number of seconds since January 1, 1970. This returns NULL if the timer is not armed.	Integer
Timer.<TimerName>.ExpiryTimeMs	Returns the time when the specified timer will expire. You can interpret the integer as the number of milliseconds since January 1, 1970. This returns NULL if the timer is not armed.	Integer

## 2.14 Tables

Tables are used to store state and build state machines in SandScript and are similar to tables in a database. See the Tables section of the *SandScript Configuration Guide* for additional information.

Field	Description	Type
Table.<TableName>.<ColumnName> Table.<TableName>[Key].<ColumnName>	The field used to read from or write to a table, where <TableName> is the name of the table and <ColumnName> is the name of the column. For explicit access to a table row, specify a Key list within square bracket accessors.	policy- defined
Table.<TableName> Table.<TableName>[Key]	The delete_row action must reference this field to delete a row from a table, where <TableName> is the name of the table. You can also use this field to assign a row to a cursor local variable. For explicit access to a table row, specify a Key list within square bracket accessors.	table.<TableName> :cursor
Table.<TableName>.<SecondaryIndexName>[<key>].Clear()	Deletes all rows in the table where the secondary index value is equal to <key>.	

## 2.15 Events

You can use SandScript to define events to let one application raise an event that other applications can listen for and take action on. See the Events section of the *SandScript Configuration Guide* for additional information.

Field	Description	Type
Event.<EventName>	Returns true whenever the event is raised. Otherwise, it returns false.	Boolean
Event.<EventName>.<ArgName>	Returns the value of argument <ArgName> of event <EventName>. Returns null if the argument was not specified when the event was raised. This field is only evaluated when the event <EventName> is raised.	policy- defined

## 2.16 Shapers

The properties of shaper input and output can be extracted using shaping fields.

Specifically, the fields expose the number of packets and bytes that pass in and out of a shaper, as well as the rate of input and output (for aggregate shapers only).

Expressions are also used in the creation of shaper actions, primarily to specify how shaper bandwidth is allocated. When fields are used, the list of expressions should be enclosed in parenthesis. For a list of suggested fields to use with shaper actions, see *Shaping* in the *PTS SandScript Configuration Guide*.

### 2.16.1 Shaper Fields

Each defined shaper has associated fields which expose its input and output statistics. Where *ShaperName* is the name of the shaper, these fields are:

Field	Description	Type
Shaper.<ShaperName>.BytesIn	Bytes entering the shaper.	Integer
Shaper.<ShaperName>.BytesOut	Bytes exiting, after shaping.	Integer
Shaper.<ShaperName>.BytesDropped	Bytes dropped by shaping.	Integer
Shaper.<ShaperName>.PacketsIn	Number of packets entering the shaper.	Integer
Shaper.<ShaperName>.PacketsOut	Number of packets exiting, after shaping.	Integer
Shaper.<ShaperName>.PacketsDropped	Number of packets dropped by shaper.	Integer

Each aggregate (non-unique-by) shaper has these associated fields, where *ShaperName* is the name of the shaper. The fields expose the input and output rates of the shaper.

Field	Description	Type
Shaper.<ShaperName>.CurrentRateIn	Bitrate into an aggregate shaper.	Integer
Shaper.<ShaperName>.CurrentRateOut	Bitrate exiting an aggregate shaper.	Integer



Field	Description	Type

## 2.16.2 Shaper Fields Usage

This example publishes two measurements: the number of packets dropped by the shaper, and the difference between the input and output rates of the same shaper. Note that the “internal” class must exist in subnets.txt. Fields and expressions access the dropped packets and rate in/out data for Shaper1.

```
shaper "HttpShaper" 1Mbps priority "low" algorithm shape max_rate 50%
if client class "internal" and protocol "http" then shape \
  to client shaper "HttpShaper" priority "low"
publish "ShaperDroppedPackets" expr (Shaper.HttpShaper.PacketsDropped)
publish "ShaperRxRate" expr (Shaper.HttpShaper.CurrentRateIn - \
  Shaper.HttpShaper.CurrentRateOut)
```

## 2.17 VoIP Quality of Experience

Voice Quality of Experience (QoE) fields are available for supported protocols for use in mid-flow policy decision-making. These fields apply to the voice call data (RTP) flows, not the call setup (control) flows. They provide access to quality metrics in each direction (Rx and Tx) for VoIP flows. Meaningful values are returned only for VoIP flows. For other types of traffic they return NULL.



**Note:**

VoIP QoE is a licensed feature. If the feature is not licensed, the fields are not available, resulting in a policy parsing error.

Field	Type	Units
Flow.Client Server.Rx Tx.VoIP.DelayAvg_EP	Float	seconds
Flow.Client Server.Rx Tx.VoIP.DelayAvg_OW	Float	seconds
Flow.Client Server.Rx Tx.VoIP.JitterDelay_EP_Avg	Float	seconds
Flow.Client Server.Rx Tx.VoIP.JitterDelay_OW_Avg	Float	seconds
Flow.Client Server.Rx Tx.VoIP.JitterDelay_OW_Max	Float	seconds
Flow.Client Server.Rx Tx.VoIP.MOS_CQ	Float	Range 0.0 - 5.0
Flow.Client Server.Rx Tx.VoIP.MOS_EP	Float	Range 0.0 - 5.0
Flow.Client Server.Rx Tx.VoIP.MOS_LQ	Float	Range 0.0 - 5.0
Flow.Client Server.Rx Tx.VoIP.MOS_Nominal	Float	Range 0.0 - 5.0
Flow.Client Server.Rx Tx.VoIP.PacketsLost_EP_Pct	Integer	packets
Flow.Client Server.Rx Tx.VoIP.PacketsLost_OW	Integer	packets
Flow.Client Server.Rx Tx.VoIP.PacketsReceived_OW	Integer	packets
Flow.Client Server.Rx Tx.VoIP.RFactor_CQ	Integer	none
Flow.Client Server.Rx Tx.VoIP.RFactor_EP	Integer	none
Flow.Client Server.Rx Tx.VoIP.RFactor_G107	Integer	none

Field	Type	Units
Flow.Client Server.Rx Tx.VoIP.RFactor_LQ	Integer	none

## 2.17.1 VoIP QoE Fields Usage

The VoIP QoE fields can be used to generate measurements for viewing wide scale trends. Measurements can be viewed using CLI commands on the PTS, or via Sandvine's Network Demographics Server reports after measurements have been published.

The following example returns the number of “Bad Calls” and number of “Good Calls” based on a MOS Conversational Quality (MOS CQ) threshold of 3.5.

```
measurement "BadCalls" connections where client class "internal" and\
  expr (Flow.Client.Rx.VoIP.MOS_CQ <= 3.5)
measurement "GoodCalls" connections where client class "internal" and\
  expr (Flow.Client.Rx.VoIP.MOS_CQ > 3.5)
```

## 2.18 Subscriber Session Fields

You must have Subscriber IP mapping deployed to use subscriber IP mapping fields.

The flow-scope fields are used to test each flow for properties of the host at the specified node. In contrast, the subscriber-scope rules run independently of flow evaluation, and compare subscriber properties on the database with corresponding values on the PTS.

Session fields are identical to subscriber-scope IP mapping fields when used without square bracket accessors. When used with square bracket accessors, session fields are more versatile than regular IP mapping fields since they can access properties and attributes of any subscriber session for which the IP address or session ID is known.



### Note:

When the Session Qualifiers feature is enabled, for subscribers whose NAT mapping is released before the end of policy run, the flows of such subscribers will become unmapped.

For Session fields, <Key> may be one of these types:

- IP Address - returns results for the subscriber session mapped to this IP address (in site 0), or NULL if no session is mapped to this IP address.
- Integer- returns results for the subscriber session with a session ID equal to this value, or NULL if there is no session in the system with the given ID.

Field	Description	Type
Session.IpPrefix Session[<Key>].IpPrefix IpAssignment.IpPrefix	Returns the user prefix of the IP address mapped to this subscriber session. The user prefix is the first N bits of the IP address, where N is the value that the Session.PrefixLength field returns.	IP address
Session.PrefixLength Session[<Key>].PrefixLength IpAssignment.PrefixLength	Returns the length of the IP prefix mapped to this subscriber session. If a subscriber is assigned an IPv6 address with a 64-bit prefix, this field will return 64.	IP address
Session.IsMapped Session[<Key>].IsMapped	Returns a boolean value having the value true if the subscriber for this session is discovered.	Boolean
Flow.Client Server.Subscriber.IsMapped Flow.Subscriber.IsMapped	Returns a boolean value indicating whether or not the host that the node specifies (Client or Server) is mapped. Use Flow.Subscriber.IsMapped if port roles are defined.	Boolean

Field	Description	Type
Subscriber.IsMapped IpAssignment.IsMapped	Returns a boolean value indicating whether or not the host is mapped. This field is only used in a subscriber-scoped rule.	Boolean
Session.Age Session[<Key>].Age	Returns the number of seconds that elapsed since a subscriber mapping request was sent for the specified IP address.	Integer
Flow.Client Server  Subscriber.IpAssignment. Age IpAssignment.Age	Returns the number of seconds that has elapsed since a subscriber mapping request was sent for the specified node.	Integer
Session.Subscriber Session[<Key>].Subscriber Session.Subscriber.Id Session[<Key>].Subscriber.Id	Returns the mapped subscriber identification for the specified IP address. Returns NULL if the IP is not mapped.	Integer
Session.SessionQualifier Session[<Key>].SessionQualifier	Returns a string representing the qualifier that applies to this subscriber session. Examples are: "Site:10" or "Site:200". Returns NULL if no subscriber session exists.	String
Flow.Session.SessionQualifier	Refer to Session.SessionQualifier, where the specified node is the implicit session tied to the flow.	String
Session.PublicIPs Session[<Key>].PublicIPs	<p>Returns a list of public IP addresses and port ranges associated with the current subscriber session. These fields are typically used to acquire details about the post-NAT IP address and port ranges in use by this subscriber. This field can be used in a foreach statement, where the following fields can be accessed:</p> <ul style="list-style-type: none"> <li>IP—Contains the actual IP address portion of the public IP.</li> <li>LowPort—Contains the lower bound (inclusive) of the range of ports for this public IP.</li> <li>HighPort—Contains the higher bound (inclusive) of the range of ports for this public IP.</li> </ul> <p>For example:</p> <pre>foreach "mapping" in Session.PublicIPs\ {   if expr(mapping.IpAddress = StringToIP("10.10.10.1") and\ mapping.LowPort &gt; 100 and\ mapping.HighPort &lt; 1000)\ then\     increment Measurement.SpecificMappings\ }</pre>	
Flow.Client Server.Subscriber Flow.Client Server.Subscriber.Id Flow.Subscriber.Id	Returns the mapped subscriber identification for the specified node if present. Returns NULL if the IP is not mapped. Use Flow.Subscriber.Id if port roles are defined.	Integer
Flow.Client Server  Subscriber.IpAssignment. Site	Refer to Session.Site, where the Session reference is that of the specified Client or Server.	Integer
Session.Site Session[<Key>].Site	Returns the site session qualifier number of the subscriber session.	Integer
Flow.Session.Id	Refer to Session.Id, where the specified node is the implicit Session tied to the flow.	Integer
Session.Id Session[<Key>].Id	Returns an integer representing a unique identifier for this subscriber session. Returns NULL if no subscriber session exists.	Integer
Subscriber Subscriber.Id IpAssignment.Subscriber.Id	Subscriber scope field returning the mapped subscriber identification. Returns NULL if the IP is not mapped.	Integer

Field	Description	Type
Session.Subscriber.Name Session[<Key>].Subscriber.Name	Returns the name of the subscriber mapped to the specified IP address. Returns NULL if the IP is not mapped.	String
Flow.Client[Server].Subscriber.Name Flow.Subscriber.Name	Returns the name of the subscriber for the specified node. Returns NULL if the IP is not mapped. Use Flow.Subscriber.Name if port roles are defined.	String
IpAssignment.Subscriber.Name	Subscriber scope field returning the name of the subscriber. Returns NULL if the IP is not mapped.	String
Session.CreatedTime Session[<Key>].CreatedTime	Returns a timestamp (number of seconds since epoch) indicating the time that a subscriber mapping request was sent for the specified IP address. Returns NULL if no subscriber session exists.	Integer
Flow.Client[Server] Subscriber.IpAssignment. CreatedTime IpAssignment.CreatedTime	Returns a timestamp (number of seconds since epoch) indicating the time that a subscriber mapping request was sent for the specified node. Returns NULL if the IP is not mapped.	Integer
Session.CreatedTimeMs Session[<Key>].CreatedTimeMs	Returns a timestamp (number of milliseconds since epoch) indicating the time that a subscriber mapping request was sent for the specified IP address. Returns NULL if no subscriber session exists.	Integer
Session.MappedTime Session[<Key>].MappedTime	Returns a timestamp (number of seconds since epoch) indicating the time that the session at the specified IP address became mapped (subscriber mapping response was received). Returns NULL if no subscriber session exists.	Integer
Flow.Client[Server] Subscriber.IpAssignment. MappedTime IpAssignment.MappedTime	Returns a timestamp (number of seconds since epoch) indicating the time that the specified node became mapped (subscriber mapping response was received). Returns NULL if the IP is not mapped.	Integer
Session.MappedTimeMs Session[<Key>].MappedTimeMs	Returns a timestamp (number of milliseconds since epoch) indicating the time that the session at the specified IP address became mapped (subscriber mapping response was received). Returns NULL if no subscriber session exists.	Integer
Session.NetworkMappedTime Session[<Key>].NetworkMappedTime	Returns a timestamp (number of seconds since epoch) indicating the time of session creation on the accounting host (beginning of authorization server host accounting record). Returns NULL if the IP is not mapped.	Integer
Session.NetworkMappedTimeMs Session[<Key>].NetworkMappedTimeMs	Returns a timestamp (number of milliseconds since epoch) indicating the time of IP assignment on the accounting host (beginning of authorization server host accounting record). Returns NULL if no subscriber session exists.	Integer
Session.Timeout Session[<Key>].Timeout	Returns a timestamp (number of seconds since epoch) representing the expiry time of the subscriber mapping at the specified IP address. Returns NULL if no subscriber session exists.	Integer
Subscriber.Timeout	Returns a timestamp (number of seconds since epoch) representing the expiry time of the subscriber mapping. Returns NULL if the IP is not mapped.	Integer
Flow.Client[Server].Subscriber.Timeout	Returns a timestamp (number of seconds since epoch) representing the expiry time of the subscriber mapping at the specified node. Returns NULL if the IP is not mapped.	Integer
Session.IpAddress Session[<Key>].IpAddress IpAssignment.IpAddress	Returns the IP address of an IP assignment (an IP that has been looked up for subscriber mapping).	IP address
Flow.Client[Server] Subscriber[Internet]. IpAddress	Returns an integer representing the IP address of the specified node.	IP address

Field	Description	Type
Session.IsNew	Returns true the first time SandScript runs on an IP that has just been mapped to a subscriber, otherwise returns false.	Boolean
IpAssignment.IsNew	Returns true the first time SandScript runs on an IP that has just been mapped to a subscriber, otherwise returns false.	Boolean
Session.IsEnd IpAssignment.IsEnd	Returns true the first time SandScript runs on an IP that has just been unmapped, otherwise returns false.	Boolean
Session.IsLogout IpAssignment.IsLogout	Returns true when Session.IsEnd is true and the IP became unmapped due to a logout notification, otherwise returns false.	Boolean
Session.IsTimeout IpAssignment.IsTimeout	Returns true when Session.IsEnd is true and the IP mapping timed out due to an absence of traffic over a configured period of time (see the output of this CLI command: <code>show config service subscriber-management timeout inactivity</code> ).	Boolean

## 2.19 Subscriber and Session Attributes

A session attribute is associated with a single subscriber session (a single subscriber to IP mapping). When that session ends, the attribute is no longer defined, even if the subscriber has other active sessions.

Subscriber attributes are associated with a particular subscriber and all of their flows for the lifetime of the subscriber.

Subscriber attributes are writable, but session attributes are read-only. Session attributes are typically gleaned from a subscriber's Diameter/RADIUS login session.

Field	Description	Type
Session.Subscriber.Attribute.<AttributeName> Session[<Key>].Subscriber.Attribute.<AttributeName>	Returns the value of the subscriber attribute <AttributeName> for the specified IP address.	user- defined
Flow.Client Server Subscriber.Attribute.<AttributeName>	Returns the value of the subscriber attribute AttributeName for the specified node (Client Server Subscriber). Use <code>Flow.Subscriber.Attribute.&lt;AttributeName&gt;</code> if port roles are defined.	user- defined
Subscriber.Attribute. <AttributeName>	Returns the value of the subscriber attribute <AttributeName>. The returned value can be used to specify a load-balancing, unique-by or shared-by specification, or in subscriber scope to check/set the value of the subscriber attribute.	user- defined
Session.Subscriber.Attribute.<AttributeName>.IsUpdated Subscriber.Attribute.<AttributeName>.IsUpdated	Returns true the first time SandScript runs on a subscriber whose subscriber attribute <AttributeName> was changed external to SandScript, false otherwise.	Boolean
Session.Attribute. <AttributeName> Session[<Key>].Attribute. <AttributeName>	Returns the value of the session-based attribute <AttributeName>. The returned value can be used to specify a unique-by or shared-by specification, or to check the value of the session attribute.	policy- defined

Field	Description	Type
	In flow scope (when SandScript is being evaluated on a flow), this field is a synonym for <code>Flow.Session.Attribute.&lt;AttributeName&gt;</code> . In subscriber scope (when SandScript is being evaluated on a subscriber) this field references the primary session associated with the subscriber. Session Attributes are read-only.	
<code>Flow.Session.Attribute. &lt;AttributeName&gt;</code>	Returns the value of the session-based attribute <code>AttributeName</code> that is associated with the subscriber IP of the flow. Use <code>Flow.Session.Attribute.&lt;AttributeName&gt;</code> if port roles are defined. Session Attributes are read-only.	user- defined
<code>Session.Attribute. &lt;AttributeName&gt;.IsUpdated</code>	Returns true the first time SandScript runs on a subscriber whose session-based attribute <code>&lt;AttributeName&gt;</code> was changed external to SandScript, false otherwise.	Boolean

## 2.20 Tunneling

Tunneling fields provide access to tunneling headers . Tunneling fields allow SandScript access to the protocol headers of the tunneling protocol in which the flow is carried. For example, if subscriber traffic is carried within a GTPU tunnel, the flow and subscriber are based on the inner subscriber IP packets, but you can also access the outer IP addresses (the tunnel end-points).

Tunneling field syntax

```
Flow.FromSubTunnel.<Header>.[<Header...>].<FieldName>
Flow.ToSubTunnel.<Header>.[<Header...>].<FieldName>
Flow.FromSubTunnel.IsFragmented
Flow.ToSubTunnel.IsFragmented
```

Field	Description
Header	Refers to supported tunneling headers. The supported headers are: <ul style="list-style-type: none"> <li>• Ether</li> <li>• MPLS</li> <li>• VLAN</li> <li>• IP</li> <li>• L2TP</li> <li>• GRE</li> <li>• GTPU</li> <li>• CAPWAP</li> </ul>
IsFragmented	IsFragmented is characteristic of a flow, and is valid per tunnel, per direction. Returns true if any packet of the flow has been found within a fragmented IP tunnel, for the specified direction. Returns false for flows not within tunnels and for flows without fragmentation. These fields are reset to false if a flow moves to different tunnel.

To discover the tunnel in the Ethernet frame use:

```
Flow.FromSubTunnel.Ether.Tunnel
Flow.ToSubTunnel.Ether.Tunnel
```

The values returned are:

- Tunnel.None
- Tunnel.IPIP
- Tunnel.GRE
- Tunnel.L2TP
- Tunnel.GTPU
- Tunnel.CAPWAP

Syntax examples

```
Flow.FromSubTunnel.Ether.Ip.SourceAddress
Flow.FromSubTunnel.Ether.ServiceVlan.Vlan.Vid
Flow.FromSubTunnel.Ether.Vlan.Vid
Flow.FromSubTunnel.Ether.Mpls.Mpls.Label
Flow.FromSubTunnel.Ether.CAPWAP.RadioID
Flow.FromSubTunnel.Ether.CAPWAP.Type
Flow.ToSubTunnel.Ether.GRE.Ip.SourceAddress
Flow.ToSubTunnel.Ether.Tunnel
```

Policy examples

This policy example counts the number of times that the GRE tunnels are encountered. The condition is true, if the IP TOS field of the outer IP traffic from subscriber to internet is 16 (hex value 0x10).

```
measurement "TOSisSet"
measurement "greTunnel"

if expr(Flow.FromSubTunnel.Ether.IP.TOS = 0x10) then increment Measurement.TOSisSet
if expr(Flow.IsNew and Flow.FromSubTunnel.Ether.Tunnel = Tunnel.Gre)\
  then increment Measurement.greTunnel
```


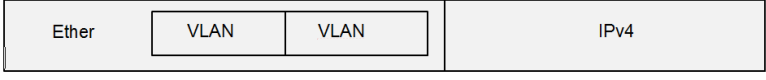

This policy example counts the number of times that the CAPWAP tunnels are encountered.

```
measurement "FromSubTunnelCAPWAP"
if expr(Flow.FromSubTunnel.Ether.Tunnel = Tunnel.CAPWAP) then increment\
  Measurement.FromSubTunnelCAPWAP
measurement "ToSubTunnelCAPWAP"
if expr(Flow.ToSubTunnel.Ether.Tunnel = Tunnel.CAPWAP) then increment\
  Measurement.ToSubTunnelCAPWAP
```





## 2.20.1 Supported Field Types

Tunneling supports these field types:

Field	Supported
Ether	<p>Supported fields from the ethernet header are:</p> <ul style="list-style-type: none"><li>• Ether.SourceMac—Source MAC address of the Ether header.</li><li>• Ether.DestinationMac—Destination MAC address of the Ether header.</li></ul> <p>VLAN and MPLS sub-headers are also supported after an ethernet header. The Ethernet header encapsulates VLAN and MPLS headers which means we skip VLAN and MPLS in the expression when we refer to the next L3 header.</p> <div><div>Ether</div><div>VLAN</div><div>Ipv4</div></div>

Field	Supported
	
VLAN	<p>VLAN sub-header is supported after an ethernet header.</p> <p>802.1Q VLAN—Returns the full 16-bit vlan tag for the specified direction of traffic with respect to the specified node. This includes both the vlan ID and the priority bits of the vlan tag.</p> <ul style="list-style-type: none"> <li>Ether.Vlan.Vid—VID of the VLAN header.</li> <li>Ether.Vlan.Priority—Priority of the VLAN header. This field can be set and read in policy. Range is from 0 to 7.</li> <li>Ether.Vlan.Priority.Mask—This field controls which bits of the VLAN priority are modified when Vlan.Priority is set. Only masked bits will be changed. This field can only be set in policy. Range is from 0 to 7.</li> </ul>  <p>802.1ad QinQ VLAN—Returns the full 16-bit service vlan tag for the specified direction of traffic with respect to the specified node. This includes both the vlan ID and the priority bits of the vlan tag.</p> <ul style="list-style-type: none"> <li>Ether.ServiceVlan.Vid—VID of the QinQ VLAN header.</li> <li>Ether.ServiceVlan.Priority—Priority of the QinQ VLAN header. This field can be set and read in policy. Range is from 0 to 7</li> <li>Ether.ServiceVlan.Priority.Mask—This field controls which bits of the QinQ VLAN priority are modified when ServiceVlan.Priority is set. Only masked bits will be changed. This field can only be set in policy. Range is from 0 to 7.</li> </ul> <p>Up to one level of nested VLAN or QinQ headers are supported.</p>
MPLS	<p>MPLS sub-header is supported after an ethernet header. Up to one level of nested MPLS headers are supported.</p> <ul style="list-style-type: none"> <li>Ether.Mpls.Label.</li> <li>Ether.Mpls.ExperimentalBits.</li> </ul> 
IP	<p>Supported fields from the Internet Protocol (IP) header are:</p> <ul style="list-style-type: none"> <li>Ip.Tos—ToS field of the IP header. This field can be set and read in policy. Range is from 0 to 255.</li> <li>Ip.Tos.Mask—This field controls which bits of the IP ToS field are modified when Ip.Tos is set. Only masked bits will be changed. This field can only be set in policy. Range is from 0 to 255.</li> <li>IP.SourceAddresss—Source IP address of the IP header.</li> <li>IP.DestinationAddress—Destination IP address of the IP header.</li> <li>IP.Protocol—Protocol field of the IP header.</li> </ul>
GRE	<p>Supported fields from the Generic Routing Encapsulation (GRE) header are:</p> <ul style="list-style-type: none"> <li>Gre.ProtocolType.</li> <li>Gre.Key.</li> </ul> <p>GRE encapsulates the IPv4 tunneling header.</p>



Field	Supported
	
L2TP	<p>Supported fields from the Layer 2 Tunneling Protocol (L2TP) header are:</p> <ul style="list-style-type: none"> <li>L2tp.TunnelId.</li> <li>L2tp.SessionId.</li> </ul> <p>L2TP encapsulates the inner IP and PPP headers.</p> 
GTPU	<p>Supported fields from the from the GTP user (GTPU) data tunneling protocol header are:</p> <ul style="list-style-type: none"> <li>GTPU.TEID.</li> <li>GTPU.HasSequenceNum.</li> <li>GTPU.HasExtensionHeader.</li> <li>GTPU.HasNpduNumber.</li> </ul> <p>GTPU encapsulates the inner IP and UDP headers.</p> 
CAPWAP	<p>Supported fields from the from the Control And Provisioning of Wireless Access Points (CAPWAP) header are:</p> <ul style="list-style-type: none"> <li>CAPWAP.RadioID—One WTP can have multiple physical radios. Radio ID field is used to indicate with which physical radio the message is associated. Range is from 1 to 31.</li> <li>CAPWAP.Type—Indicate the type of payload in CAPWAP encapsulated tunnel. Values are CAPWAP.Type.IEEE80211 and CAPWAP.Type.IEEE8023.</li> <li>CAPWAP.RadioMACAddress—Indicates the MAC address of the radio receiving the packet. 6-byte integer.</li> <li>CAPWAP.RSSI—Indicates the received signal strength indication (RSSI, signed 8 bits, in units of dBm) from IEEE 802.11 wireless specific information field. Range is from -128 to 125.</li> <li>CAPWAP.SNR—Indicates the Signal to Noise field (SNR, signed 8 bits, in units dB), from IEEE 802.11 wireless specific information field. Range is from -128 to 127.</li> <li>CAPWAP.DataRate—Indicates the Data Rate Field (unsigned 16 bits, in units of 0.1 Mbps) from IEEE 802.11 wireless specific information field. Range is from 0 to 2^16-1.</li> <li>CAPWAP.WLAN.SourceMac—Indicates the Source MAC address retrieved from IEEE 802.11 header. 6-byte integer.</li> <li>CAPWAP.WLAN.DestinationMac—Indicates the Destination MAC address retrieved from IEEE 802.11 header. 6-byte integer.</li> <li>CAPWAP.WLAN.BSSID—Indicates the Basic Service Set ID (BSSID) retrieved from IEEE 802.11 header. 6-byte integer.</li> </ul> <p>CAPWAP encapsulates the inner IP and UDP packets.</p> 

## 2.21 Network Protection

Network protection policy fields are applied to flows identified by Network Protection (formerly WDTM) to be malicious.

Field	Description
Flow.Wdtm.IsMitigated	Returns true if the flow is being mitigated, false otherwise.
Flow.Wdtm.MaliciousClass	The malicious cost class associated with the flow.
Flow.Wdtm.MaliciousIp	The malicious IP associated with the flow.
Flow.Wdtm.DetectionType	<p>The type of event that was detected on the flow. This field is used in expressions. For example, <code>expr(Flow.Wdtm.DetectionType = DetectionType.AddressScan)</code>. The constants are:</p> <ul style="list-style-type: none"><li>• <code>DetectionType.AddressScan</code>: An address scan event was detected on the flow.</li><li>• <code>DetectionType.FlowFlood</code>: A flow flood event was detected on the flow.</li><li>• <code>DetectionType.SynFlood</code>: A syn flood event was detected on the flow.</li><li>• <code>DetectionType.UserBandwidth</code>: A user bandwidth attack was detected on the flow.</li><li>• <code>DetectionType.SignatureMatch</code>: A signature match was detected on the flow.</li><li>• <code>DetectionType.Spammer</code>: A spammer event was detected on the flow.</li><li>• <code>DetectionType.UserTxBandwidth</code>: A user tx bandwidth attack was detected on the flow.</li><li>• <code>DetectionType.DnsRequestFlood</code>: A DNS request flood was detected on the flow.</li><li>• <code>DetectionType.DnsMaliciousDomain</code>: A DNS malicious domain was detected on the flow.</li><li>• <code>DetectionType.DnsOutstandingSessions</code>: DNS outstanding sessions were detected on the flow.</li><li>• <code>DetectionType.Unconditional</code>: An unconditional event was detected on the flow.</li></ul>
Flow.Wdtm.RuleNumber	The Network Protection rule number that caused this flow to be mitigated.

## 2.22 Utility Fields

Utility fields include:

Field	Description	Type
Random	Returns a value between 0 and 1.	Float
Now	Returns a value representing the number of seconds elapsed since January 1, 1970.	Integer

## 2.23 Expressions for Unique-by

Unique-by clauses can use single expressions or multiple expressions by putting expressions in expression lists.

“unique-by” can be used to:

- Treat different categories of flows. This is useful for shapers, limiters, measurements, and tables.

This example causes a shaper to act on HTTP traffic. A field is used here to specify that bandwidth is allocated to each IP address instance.

```
shaper "Web" 10kbps priority "Normal" algorithm police
if protocol "http" then shape to subscriber \
shaper "Web" priority "Normal" \
    unique by (Flow.Subscriber.IpAddress)
```

This example creates a measurement which counts Ares connections which are unique by IP address.

```
measurement "InternalAresConnections" connections where protocol \
"ares" unique by (Flow.Subscriber.IpAddress)
```

## 2.24 Published Expressions

Some PTS expressions can be periodically logged for reporting in Network Demographics.

The syntax used to publish an expression is:

```
publish "published_expr_name" expression
```

Where:

- "published\_expr\_name" is a string which identifies the published expression.
- expression defines the value of the expression.

Published expressions can be viewed using the CLI and via Network Demographics.

Measurements, limiters and shapers are commonly published. For example:

```
publish "MyMeasurement" measurement.MeasurementName
publish "ShaperDroppedPackets" Shaper.HttpShaper.PacketsDropped
publish "Limiter" Limiter.LimiterName.LimitedFlows
```

## 2.25 Environment Fields

Environment fields reveal the state of the element in policy.

Field	Description	Type
Cluster.NumActiveModules	Returns the number of PTSD instances running on the current PTS.	Integer
Cluster.Name	Returns the name of the cluster.	String
Element.Hostname	Returns a string representing the hostname of the element on which the policy engine is running.	String
Element.IsInline	True when the element is inline; False when it is offline. This field is not settable.	Boolean
Element.StartTimestamp	Returns a timestamp for when the element came on, represented as the number of seconds elapsed since January 1, 1970.	Integer

Field	Description	Type
Module.Name	Returns the name of the module on which the policy engine is running. For single module elements, the value is the same as Element.Hostname but, for other platforms this string is returned:  <platform>-X.module-<N>.<hostname>  Where: <ul style="list-style-type: none"> <li>• &lt;platform&gt;—Is either ptsd or sde.</li> <li>• &lt;X&gt;—Is a unique policy engine number within the module.</li> <li>• &lt;N&gt;—Is a unique module number within the element.</li> <li>• &lt;hostname&gt;—Identifies the hostname of the element.</li> </ul>	String
Policy.Reloaded	Returns true if policy is being run because of a successful reload. This field is only available in policy reload scope.	Boolean
Policy.ReloadTimestamp	Returns a timestamp for the last time policy was successfully reloaded, represented as the number of seconds elapsed since January 1, 1970. This field can be used in any scope.	Integer
PolicyEngine.Name	Returns the name of the policy engine that is currently evaluating policy. For single module elements, the value will be the same as Element.Hostname. For platforms with multiple modules, each running a single policy engine, the value will be the same as Module.Name.	String

## 2.25.1 Environment Fields Example

```

PolicyGroup {
# if inline then block
if expr(Element.IsInline) and protocol "http" then block
# else if offline captive portal
if protocol "http" then captive_portal "http://mycaptive.com"
}

```

## 2.26 Diameter Fields

Diameter fields reveal information from Diameter messages according to a dictionary defined by the Diameter subsystem. These fields are available in a Diameter message scope defined by the subsystem.

Field	Description	Type
Diameter.<DictionaryName>. <CommandName-CommandSuffix>. <HeaderField>	Returns the value of a Diameter message header field. DictionaryName is the name of the dictionary that defines the AVP. CommandName is the name of the application command that defines the message. CommandSuffix is one of: <ul style="list-style-type: none"> <li>• Request</li> <li>• Answer</li> <li>• Answer-Error</li> </ul> HeaderField is one of:	Dictionary-defined

Field	Description	Type
	<ul style="list-style-type: none"> <li>Proxiable: Returns true if the p-bit in the message header is set, false otherwise.</li> <li>Retransmitted: Returns true if the t-bit in the message header is set, false otherwise.</li> <li>End-to-End: Returns the integer value of the end-to-end identifier from the Diameter message header.</li> <li>Hop-by-Hop: Returns the integer value of the hop-by-hop identifier from the Diameter message header.</li> <li>Application-Id: Returns the integer value of the application-id from the Diameter message header.</li> <li>PeerIndex: Returns an integer that identifies the peer that sent the Diameter message. The value of this field from a request message should be used as the value for the PeerIndex argument when sending the answer to the request.</li> </ul>	
Diameter.<DictionaryName>. <CommandName-CommandSuffix>. <AvpName1>{[Index1]} {.<AvpName2>{[Index2]} ...}	<p>Returns the value of an Attribute-Value Pair (AVP). DictionaryName is the name of the dictionary that defines the AVP. CommandName is the name of the application command that defines the message. CommandSuffix is one of:</p> <ul style="list-style-type: none"> <li>Request</li> <li>Answer</li> <li>Answer-Error</li> </ul> <p>AvpName1 is the name of an AVP from the dictionary. Index is an integer expression that specifies the instance of the AVP to access. If the Index expression is absent a value of 0 is assumed. Grouped AVPs are of type boolean. When a grouped AVP is present its value is true. When absent the value is null.</p> <p>If a dictionary name, command name, or AVP name starts with something other than a lower or upper case letter, or if the name contains anything other than alphanumeric characters or underscores, then the name needs to be surrounded by single quotes. For example:</p> <pre>Diameter.Gy.'CC-Answer'.'Session-Id'</pre>	Dictionary-defined
Diameter.<DictionaryName>. <CommandName-AnswerSuffix>. Request.<RequestField>	<p>Returns the value of a request AVP. CommandName is the name of the application command that defines the message. CommandSuffix is one of:</p> <ul style="list-style-type: none"> <li>Answer</li> <li>Answer-Error</li> </ul> <p>All Diameter answers expose request AVP fields. The request AVP fields can be accessed with the ".Request" and then the AVP accessor fields. For example:</p> <pre>Diameter.Gy.'CC-Answer'.Request.'CC-Request-Type'</pre>	Dictionary-defined
<AVPFieldName>.RawValue	<p>Returns the raw form of an AVP as a string. Each AVP can be accessed in policy through its accessor field. The raw value of an AVP can be accessed with the ".RawValue" field. The ".RawValue" field of an AVP evaluates to a string containing the full raw AVP value</p>	Dictionary-defined

Field	Description	Type
	<p>which includes the AVP header, value and padding. The request raw AVP values are also accessible in an answer. FieldName is any Diameter AVP field. For example:</p> <pre>Diameter.Gy.'CC-Answer'. 'Result-Code'.RawValue Diameter.Gy.'CC-Answer'.Request.'CC-Request-Type'.RawValue</pre>	
<p>&lt;AVPFieldName&gt;.Add ({Before: PAL_expression}, {After: PAL_expression}, Value: PAL_expression)</p>	<p>Adds the specified Diameter AVP to the message being processed by the policy engine. The action indicates if the action was successfully added to the Diameter message.</p> <ul style="list-style-type: none"> <li>• &lt;AVPFieldName&gt; is any Diameter AVP field.</li> <li>• Before is the zero-indexed ordinal position of the AVP (within the same group as the AVP to be added) before which the AVP should be added. If this value is larger than the number of AVPs in the group, then the AVP is added at the end of the group. If this argument is not specified, you must specify the 'After' argument.</li> <li>• After is the zero-indexed ordinal position of the AVP (within the same group as the AVP to be added) after which the AVP should be added. If this value is larger than the number of AVPs in the group, then the AVP is added at the end of the group. If this argument is not specified, you must specify the 'Before' argument.</li> <li>• Value indicates the value to write in the AVP. If the AVP being added is a group AVP, this argument is not accepted and an empty group AVP is created. The type of the expression passed as the Value argument must be the type of data stored in the AVP.</li> </ul> <p><b>Examples:</b> This action creates an empty Multiple-Services-Credit-Control Grouped AVP after the Session-Id AVP:</p> <pre>Diameter.Gy.'CC-Answer'. 'Multiple-Services-Credit-Control'.   \   -- Add (After: Diameter.Gy.'CC-Answer'. 'Session-Id')</pre> <p>This action adds the Rating-Group AVP with a value of 7 at the beginning of the second Multiple-Services-Credit-Control AVP in the Diameter message:</p> <pre>Diameter.Gy.'CC-Answer'. 'Multiple-Services-Credit-Control'[1].   \   -- 'Rating-Group'.Add (Before: 0, Value: 7)</pre>	Boolean
<p>&lt;AVPFieldName&gt;.Move ({Before: PAL_expression}, {After: PAL_expression})</p>	<p>Moves the specified Diameter AVP in the message being processed by the policy engine to the specified position within its group. The action indicates if the AVP was successfully moved.</p> <ul style="list-style-type: none"> <li>• &lt;AVPFieldName&gt; is any Diameter AVP field.</li> <li>• Before is the zero-indexed ordinal position of the AVP (within the same group as the AVP to be moved) before which the AVP should be moved. If this value is larger than the number of AVPs in the group, then the AVP is moved to the end of the group. If this argument is not specified, you must specify the 'After' argument.</li> <li>• After is the zero-indexed ordinal position of the AVP (within the same group as the AVP to be moved) after which the AVP should be moved. If this value is larger than the number of AVPs in the</li> </ul>	Boolean

Field	Description	Type
	<p>group, then the AVP is moved to the end of the group. If this argument is not specified, you must specify the 'Before' argument. For example, this action moves the first Multiple-Services-Credit-Control AVP in the Diameter message immediately after the Session-Id AVP:</p> <pre>Diameter.Gy.'CC-Answer'. 'Multiple-Services-Credit-Control'. ___\___ Move (After: Diameter.Gy.'CC-Answer'. 'Session-Id')</pre>	
<AVPFieldName>.Remove()	<p>Removes the specified Diameter AVP in the message being processed by the policy engine. The indicates if the AVP was successfully removed from the Diameter message.</p> <p>&lt;AVPFieldName&gt; is any Diameter AVP field.</p> <p>For example, this action removes the third Multiple-Services-Credit-Control, and all AVPs which are members of that grouped AVP, from the Diameter message:</p> <pre>Diameter.Gy.'CC-Answer'. 'Multiple-Services-Credit-Control' [2]. ___\___ Remove ()</pre>	Boolean
<AVPFieldName>.GroupPosition	<p>Returns the zero-indexed ordinal position of the AVP within its parent AVP group. For top level AVPs, this field returns the ordinal position within the message. Within a given group, or in the top level, a grouped AVP is counted as one AVP.</p> <p>&lt;AVPFieldName&gt; is any Diameter AVP field.</p> <p>For example, this field returns the position of Result-Code within the CC-Answer Diameter message:</p> <pre>Diameter.Gy.'CC-Answer'. 'Result-Code'.GroupPosition</pre>	Integer
<AVPFieldName>.Count	<p>Returns the number of occurrences of an AVP within its parent AVP group. For top level AVPs, this field returns the number of occurrences of the AVP in the top level. This field never returns null. If no instances of the AVP are found, the field returns 0 (zero).</p> <p>&lt;AVPFieldName&gt; is any Diameter AVP field.</p> <p>For example:</p> <pre>Diameter.Gy.'CC-Answer'. 'Result-Code'.Count  Diameter.Gy.'CC-Answer'.Request.'CC-Request-Type'.Count</pre>	Integer

## 2.27 Controller Fields

Controller fields are used for publishing controller data.

Field	Description
Controller.<ControllerName>.Output.Current	The current value of the controller. This value is updated according to the interval specified in the controller definition. This field can only be used as a shaper rate for a shaper that is defined with dynamic distribution.

Field	Description
Controller.<ControllerName>.Score	Controller field for publishing the scores that have been calculated within the last publish interval. The publish format is a linear histogram from 0...100 containing 20 bins. The field may only be used for publishing.
Controller.<ControllerName>.Demand	An integer representing the average demand seen on the controller over the last publish interval. The field may only be used for publishing, and is only available if a shaper with this controller output as its rate is defined.
Controller.<ControllerName>.Output	Controller field for publishing the outputs that have been calculated by the control system during the previous publish interval. The publish format is a histogram, using the definition of the output_histogram control system parameter. The field may only be used for publishing, and is only available if the controller option output_histogram is set to a valid histogram expression.
Controller.<ControllerName>.<MetricName>	Controller field for publishing the calculated metric value which has been input to the control system over the last publish interval. The publish format is a value representing the average of each measurement over the publish interval. The field may only be used for publishing.
Controller.<ControllerName>.<MetricName>.Data	Controller field for publishing the full set of calculated metric value which has been input to the control system over the last publish interval. The publish format is the expression that was passed as the 'data' controller metric parameter. The field may only be used for publishing.







# 3

## Functions

- ["General Functions" on page 59](#)
- ["Map Functions" on page 69](#)
- ["Table Functions" on page 71](#)
- ["Diameter Functions" on page 71](#)
- ["Subscriber Mapping Functions" on page 72](#)

## 3.1 General Functions

Functions take one or more arguments (in the form of expressions) and return a value.

The function argument types and names are specified. Optional arguments are enclosed in { } and the default value follows the equal sign. For example: {integer optional=10}.

### 3.1.1 Conversion Functions

Functions that convert input to some other form are:

Function	Description	Type
BinaryString (string s)	Converts s to a binary string by interpreting special backslash-escaped characters. Supported special characters are: <ul style="list-style-type: none"><li>• \n -&gt; newline</li><li>• \r -&gt; carriage return</li><li>• \" -&gt; "</li><li>• \' -&gt; '</li><li>• \\ -&gt; \</li></ul>	String
Boolean(expression e)	Converts the value of the expression to a boolean. Accepts integer, float and string types. For integers and floats, returns true for non-zero values, false otherwise. For strings: <ul style="list-style-type: none"><li>• "true", "t", "yes", "y" are converted to true</li><li>• "false", "f", "no", "n" are converted to false</li><li>• any other string returns null</li></ul>	Boolean
Float(expression e)	Converts the value of the expression to a float. Typically used to convert string (for example "12.345", "0xab"). If the conversion is invalid (for example float("nowork")) returns NULL. Will accept integer, string, and boolean types.	Float
Integer(expression e)	Converts the value of the expression to an integer. Typically used to convert strings (for example "12345", "0xab"). If the conversion is invalid (for example integer("nowork")) returns NULL. Will accept float, string, and boolean types.	Integer
IpAddress(expression e)	Converts the value of the expression to an IP address. Typically used to convert strings. For example: <ul style="list-style-type: none"><li>• "192.168.1.1"</li><li>• "2001:1::3"</li></ul>	IpAddress
IpToString(integer ip)	Converts the integer IP value to a formatted string. Can be used to convert IP fields like Flow.Client.IpAddress to human readable form.	String
NetmaskToPrefixLength (ipaddress ip)	Returns the prefix length of a netmask represented as an IP address. In case of an input which is not a valid mask, the function returns null. For example: <ul style="list-style-type: none"><li>• NetmaskToPrefixLength(StringToIp("FFFF::")) returns 16</li><li>• NetmaskToPrefixLength(StringToIp("255.255.255.0")) returns 24</li><li>• NetmaskToPrefixLength(StringToIp("255.128.255.0")) returns null</li></ul>	

Function	Description	Type
PrefixLengthToNetmask (integer length, {bool isV4})	Returns a netmask represented as an IP address that matches the prefix length in the input. If isV4=true, an IPv4 netmask is returned. If isV4=false, then an IPv6 netmask is returned. If the length argument is less than 0 or greater than the maximum length of the requested IP version, null is returned. If length <= 32, then isV4 defaults to true. If length > 32, it defaults to false. For example: <ul style="list-style-type: none"> <li>PrefixLengthToNetmask(23) returns 255.255.254.0</li> <li>PrefixLengthToNetmask(23, true) returns 255.255.254.0</li> <li>PrefixLengthToNetmask(23, false) returns FFFF:FE00::</li> <li>PrefixLengthToNetmask(38) returns FFFF:FFFF:FC00::</li> <li>PrefixLengthToNetmask(129) returns null</li> </ul>	
String(expression e)	Converts the value of the expression to a string. Will accept integer, float, boolean and IPAddress types.	String
StringToIp(string ip)	Converts a human readable IPv4 or IPv6 address in string format to an IP Address. For example: <ul style="list-style-type: none"> <li>StringToIp("192.168.1.1")</li> <li>StringToIp("2001:1::3")</li> </ul> <p><b>Note:</b> If a string literal is passed to this function instead of a field, and the string is not a valid IP address, policy will fail to reload.</p>	IPAddress
StringToMac(string mac)	Takes a readable MAC in string format and converts it to an integer. The MAC string format is: xx:xx:xx:xx:xx:xx	Integer

## 3.1.2 Octet Conversion Functions

These functions convert to and from octets:

Function	Description
HexToOctets(string hexString)	Converts a hex formatted string to its octet string equivalent. Odd length strings will assume a leading zero. null is returned on invalid hex characters. Valid hex characters are: 0-9, a-f, A-F. For example: HexToOctets("0054FEDE") returns {00,54,FE,DE}
OctetsIpToString(string OctetsIP)	Converts the octet sequence string to its dotted/colon IPv4/IPv6 policy string equivalent. PolicyString must be either 4 or 16 bytes. IPv6 addresses are returned in their compressed format. For example: <ul style="list-style-type: none"> <li>OctetsIpToString({C0,A8,00,01}) returns "192.168.0.1"</li> <li>OctetsIpToString({C0,A8,00,01, 01, 01}) returns null</li> <li>OctetsIpToString({C0,A8}) returns null</li> <li>OctetsIpToString({C0,A8,00,01,01,01,01,01,01,01,01,01,01,01,01,01}) returns "c0a8:1:101:101:101:101:101:101"</li> </ul>
OctetsToHex(string octetsString)	Converts a string of octets (binary) to its hex string equivalent with leading zeros. This means each byte will be converted to a two-character string. The string is

Function	Description
	<p>converted in order, so the first byte will be the first two characters, and the first character will be the first 4 bits or the first byte. For example: OctetsToHex({00,54,FE,DE}) returns "0054fed"</p>
Serialize(string format, expression e1, {expression e2, {...}})	<p>Converts a list of policy expressions into an octet string. Each format character, except the padding, requires an additional argument to be provided to the Serialize function.</p> <p>The octet string uses these format characters, as indicated.</p> <p>Integers:</p> <ul style="list-style-type: none"> <li>c - 8 bit integer</li> <li>s - 16 bit integer, little endian</li> <li>S - 16 bit integer, big endian</li> <li>i - 32 bit integer, little endian</li> <li>I - 32 bit integer, big endian</li> <li>w - 64 bit integer, little endian</li> <li>W - 64 bit integer, big endian</li> </ul> <p>Integer types may be followed by a "u" to indicate an unsigned value.</p> <p>Floating point values:</p> <ul style="list-style-type: none"> <li>q - 64 bit floating point, little endian</li> <li>Q - 64 bit floating point, big endian</li> </ul> <p>Padding:</p> <ul style="list-style-type: none"> <li>x - 1 byte of null (0) character padding</li> </ul> <p>The padding type may be followed by a count if more than one padding byte is required.</p> <p>Strings:</p> <ul style="list-style-type: none"> <li>a - string padded by null (0) characters</li> <li>A - string padded by spaces</li> </ul> <p>String types may be followed by a count specifying a fixed length or by "*" if an arbitrary length string must be serialized. If a fixed length is specified, the input string will be padded or truncated to match the specified length. Arbitrary length strings will not be padded.</p> <p>IP address values:</p> <ul style="list-style-type: none"> <li>Z - IP address, network order. 4 bytes are serialized if the input expression is an IPv4 address. 16 bytes are serialized if the input expression is an IPv6 address</li> <li>Z4 - IPv4 address, network order. If the input expression is an IPv6 address, the address 0.0.0.0 is serialized</li> <li>Z6 - IPv6 address, network order. If the input expression is an IPv4 address, the address will be serialized using IPv4-mapped-IPv6 format</li> <li>z4 - IPv6 address, little endian. If the input expression is an IPv6 address, the address 0.0.0.0 is serialized</li> </ul> <p>For example: Serialize("ccx2", 5, 9) returns {05, 09, 00, 00}</p>
StringIpToOctets(string ip)	<p>Converts the dotted IPv4 policy string to its 4 byte octet string equivalent. For example: StringIpToOctets("192.168.0.1") returns {C0,A8,00,01}</p>

### 3.1.3 Time Functions

Functions that can be used to manage time data or timestamps are:

Function	Description												
GmTime(integer year, integer month, integer day, {integer hour=0}, {integer minute=0}, {integer seconds=0})	Constructs and returns a timestamp integer from the specified arguments representing UTC.												
LocalTime(integer year, integer month, integer day, {integer hour=0}, {integer minute=0}, {integer seconds=0})	Constructs and returns a timestamp integer from the specified arguments representing local time.												
TimeIntervalMonth(integer rollover_day)	Returns an integer that uniquely identifies the current month. The value returned will change on the day of the month specified by the rollover_day argument. For example, if rollover_day is 8, the value will change on the 8th day of each month, at 00:00 local time. The function returns NULL for rollover_day values less than 1 and greater than 28.												
TimeIntervalWeek(integer rollover_day_of_week)	Returns an integer that uniquely identifies the current week. The value returned will change on the day of the week specified by the rollover_day_of_week argument, where 0 is Sunday, 1 is Monday, etc., 6 is Saturday. For example, if rollover_day_of_week is 2, the value will change every Tuesday at 00:00 local time. The function returns NULL for rollover_day_of_week values less than 0 and greater than 6.												
TimeIntervalDay(integer rollover_hour)	Returns an integer that uniquely identifies the current day. The value returned will change at the top of the hour specified by the rollover_hour argument. For example, if rollover_hour is 22, the value will change each day at 22:00 local time. The function returns NULL for rollover_hour values less than 0 and greater than 23.												
TimeIntervalHour(integer rollover_minute)	Returns an integer that uniquely identifies the current hour. The value returned will change at the minute of each hour specified by the rollover_minute argument. For example, if rollover_minute is 45, the value will change each hour at 45 minutes past the hour, local time. The function returns NULL for rollover_minute values less than 0 and greater than 59.												
TimeIntervalMonth.Remaining (integer rollover_day)	Returns the number seconds remaining before the value returned by TimeIntervalMonth will change.												
TimeIntervalWeek.Remaining (integer rollover_day)	Returns the number seconds remaining before the value returned by TimeIntervalWeek will change.												
TimeIntervalDay.Remaining (integer rollover_day)	Returns the number seconds remaining before the value returned by TimeIntervalDay will change.												
TimeIntervalHour.Remaining (integer rollover_day)	Returns the number seconds remaining before the value returned by TimeIntervalHour will change.												
Timestamp(string s)	Converts a timestamp string (for example, "2007-12-31T23:59Z") to an integer value corresponding to the number of seconds since January 1, 1970.												
TimestampToString(integer date, {integer format = DateFormat.DateAndTime24, {string customFormat = Null}})	<p>Converts a UTF integer to a timestamp string (for example, "2011-Mar-17 16:32:20") to one of these formats:</p> <table> <tr> <td>DateFormat.DateAndTime24</td><td>YYYY-MM-DD HH:MM:SS</td></tr> <tr> <td>DateFormat.DateAndTime24DMY</td><td>DD-MMM-YYYY HH:MM:SS</td></tr> <tr> <td>DateFormat.DateAndTime</td><td>YYYY-MM-DD HH:MM:SS BM</td></tr> <tr> <td>DateFormat.DateAndTimeDMY</td><td>DD-MMM-YYYY HH:MM:SS BM</td></tr> <tr> <td>DateFormat.Date</td><td>YYYY-MM-DD</td></tr> <tr> <td>DateFormat.DateDMY</td><td>DD-MMM-YYYY</td></tr> </table>	DateFormat.DateAndTime24	YYYY-MM-DD HH:MM:SS	DateFormat.DateAndTime24DMY	DD-MMM-YYYY HH:MM:SS	DateFormat.DateAndTime	YYYY-MM-DD HH:MM:SS BM	DateFormat.DateAndTimeDMY	DD-MMM-YYYY HH:MM:SS BM	DateFormat.Date	YYYY-MM-DD	DateFormat.DateDMY	DD-MMM-YYYY
DateFormat.DateAndTime24	YYYY-MM-DD HH:MM:SS												
DateFormat.DateAndTime24DMY	DD-MMM-YYYY HH:MM:SS												
DateFormat.DateAndTime	YYYY-MM-DD HH:MM:SS BM												
DateFormat.DateAndTimeDMY	DD-MMM-YYYY HH:MM:SS BM												
DateFormat.Date	YYYY-MM-DD												
DateFormat.DateDMY	DD-MMM-YYYY												

Function	Description
	<p>DateFormat.DateNumber                    YYYY/MM/DD  DateFormat.DateNumberDMY                DD/MM/YYYY  DateForamt.Time                            HH:MM:SS BM  DateFormat.Time24                          HH:MM:SS  DateFormat.DateIsoStandard                YYYYMMDDTHHMMSS  DateFormat.Custom</p> <p>When specifying a custom format the customFormat parameter must also be supplied. The customFormat string can use these symbols to build the desired string:</p> <p>%d    day of the month as a decimal number (range 01 to 31).  %y    year as a decimal number without a century.  %Y    year as a decimal number including the century.  %H    hour as a decimal number using a 24-hour clock).  %M    minute as a decimal number (range 00 to 59).  %S    second as a decimal number (range 00 to 60).  %B    full month name according to the current locale.  %D    Equivalent to %m/%d/%y.  %T    The time in 24-hour notation (%H:%M:%S).</p>
ToMicroseconds(expression e)	Converts an integer or float expression to have units microseconds. The expression must have time units. For example, calling ToMicroseconds on an expression with a value of 5 milliseconds returns 5000 microseconds.
ToMilliseconds(expression e)	Converts an integer or float expression to have units milliseconds. The expression must have time units. For example, calling ToMilliseconds on an expression with a value of 5 seconds returns 5000 milliseconds.
ToSeconds(expression e)	Converts an integer or float expression to have units seconds. The expression must have time units. For example, calling ToSeconds on an expression with a value of 1000 milliseconds returns 1 second.
Year(integer timestamp) Month(integer timestamp) Day(integer timestamp) DayOfWeek(Integer timestamp) Hour(integer timestamp) Minute(integer timestamp) Second(integer timestamp)	These functions extract and return an integer representing the unit of time specified by the timestamp argument. DayOfWeek() returns 0 for Sunday, 1 for Monday, and so on.

## 3.1.4 String Functions

Functions for manipulating strings are:

Function	Description
Concat(expression e1, {expression e2, {e3, {...}}})	Concatenates all of the arguments together and returns the resulting string. The arguments can be of any type (for example, Concat("hello", 123) returns "hello123").
Contains(string substr, string s)	Returns true if s contains substr, false otherwise.
Extract(string delim, integer index, string s, {boolean nullIfEmpty=false})	Extract a string field from a character-delimited list. The first field has index 0, the second has index 1, and so on. Negative indices can be used to locate fields starting at the end of the string (the last field has index -1). The optional nullIfEmpty argument, available only in 5.60 and higher, can be specified to return null instead of the empty string. For example:

Function	Description
	<ul style="list-style-type: none"> <li>Extract(",", 0, "abc,,de,f") returns "abc"</li> <li>Extract(",", 2, "abc,,de,f") returns "de"</li> <li>Extract(",", -1, "abc,,de,f") returns "f"</li> <li>Extract(",", 1, "abc,,de,f", true) returns null</li> </ul>
Join(string delim, expression e1, {expression e2, {e3, {...}}})	Joins together a list of expressions separated by a constant single-character delimiter and returns the resulting string. Join is a specialized, optimized version of Concat. The expression arguments can be of any type (for example Join(",", "hello", 123, true) returns "hello,123,true").
Left(string s, {integer count=1})	Returns a string made up by the first count characters of s.
Length(string s)	Returns an integer representing the length of s.
Mid(string s, integer pos, {integer count=1})	Extract a substring of s of length count, starting at position pos. If count is not specified, it defaults to 1.
Regex(string reg, string s)	Returns true if s matches regular expression reg, false otherwise.
RegSub(string reg, string replacement, string s)	Returns the string s with the parts that match the regular expression reg replaced by the value of the string replacement.
ReplaceFirst(string substr, string replacement, string s)	Returns the string s with the first occurrence of substr replaced by the value of the string replacement. Returns s if s does not contain substr.
Right(string s, {integer count=1})	Returns a string made up by the last count characters of s.
SubStr(string s, integer pos, {integer count=end})	Extract a substring of s of length count, starting at position pos. If count is not specified, the substring is extracted to the end of s.
ToLower(string s)	Converts s to lower case.
ToUpper(string s)	Converts s to upper case.

## 3.1.5 Logical Operator Functions

These functions perform logical operations:

Function	Description
All(expression e1, {expression e2, {e3, {...}}})	Returns true only if all of the arguments are true. Returns false if any of the arguments are false, even if some of the arguments are null. The arguments must all be of type boolean. This function is an optimized form of a compound AND expression (e1 and e2 and e3 ...).
Any(expression e1, {expression e2, {e3, {...}}})	Returns true if any of the arguments are true, even if some of the arguments are null. Returns false only if all of the arguments are false. The arguments must all be of type boolean. This function is an optimized form of a compound OR expression (e1 or e2 or e3 ...).
OneOf(expression e1, expression e2, {e3, {...}})	Returns true if e1 is equal to e2, or e3, and so on. At least two arguments, of any type, are required. For example: OneOf(subscriber.attribute.color, "red", "green", "blue", "yellow") is equivalent to subscriber.attribute.color="red" or \         subscriber.attribute.color="green" or \



Function	Description
	subscriber.attribute.color="blue" or \ subscriber.attribute.color="yellow" Except the OneOf() version is shorter and much more efficient.

## 3.1.6 Math Functions

Functions used for mathematical operations are:

Function	Description
Abs(integer value) Abs(float value)	Returns the absolute value of value. Returns an integer if value is an integer, returns a float if value is a float.
Between(expression lower, expression upper, expression value1, {value2 ...})	Returns true if any of the value arguments are within the range (lower <= value and value < upper), even if some other value arguments are null. Returns false if none of the arguments are within the range and none of the arguments are null. Returns null if none of the non-null arguments are within the range and one or more of the arguments are null. Returns null if either lower or upper is null.
Ceil(float value, {integer precision=0})	Returns a float that is value rounded up to the nearest integer value. The second argument may specify the number of digits to the right (if positive) or to the left (if negative) to round to.
Coalesce(expression e1, {expression e2, {e3, {...}}})	Returns the value of the first non-null argument, null only if all arguments are null. The arguments can be of any type. Returns the dominant type of the arguments specified.
FindThreshold(integer value, integer max, integer threshold1, integer threshold2, ...)	Finds the highest threshold (expressed as a percentage) that has been exceeded by value, relative to max. For example, FindThreshold(500, 1000, 25, 40, 100) returns 40 (500 is 50% of 1000, 40 is the highest threshold exceeded). At least one threshold must be specified, at most 10. The thresholds must be constant and do not need to be specified in order. Returns null if the value is below the lowest threshold.
Floor(float value, {integer precision=0})	Returns a float that is value rounded down to the nearest integer value. The second argument may specify the number of digits to the right (if positive) or to the left (if negative) to round to.
GreatestCommonDivisor(integer a, integer b)	Returns the greatest common divisor for the input parameters <i>a</i> and <i>b</i> . For example: <ul style="list-style-type: none"> <li>GreatestCommonDivisor(100, 20) = 20</li> <li>GreatestCommonDivisor(50, 20) = 10</li> </ul>
Maximum(expression value1, expression value2, {expression value3 ...})	Returns the maximum value from the list of arguments. Returns the dominant type of the arguments specified.
Minimum(expression value1, expression value2, {expression value3 ...})	Returns the minimum value from the list of arguments. Returns the dominant type of the arguments specified.
Round(float value, {integer precision=0})	Returns a float that is value rounded to the nearest integer value. The second argument may specify the number of digits to the right (if positive) or to the left (if negative) to round to. The default value is 0. For example, Round(1.234, 1) will return 1.2, Round(123.9, -2) will return 100.

Function	Description
TestNull(expression e1, expression e2)	Returns null if e1 is null, otherwise returns e2. The arguments can be of any type. Equivalent to (e1 is null ? null : e2).

## 3.1.7 Network Functions

Functions used to manage and manipulate data relating to the network or policy are:

Function	Description
ApplySubnetMask(address ip, integer length)	Returns the IP address obtained by applying a subnet mask of the input length to the input IP address. If the input address is an IPv4 address and the input length is greater than 32, then a length of 32 will be used. If the input address is an IPv6 address and the input length is greater than 128, then a length of 128 will be used. If either argument is null, or the length argument is negative, the function will return null. For example: <ul style="list-style-type: none"> <li>ApplySubnetMask(IPAddress("192.168.0.1"), 8) returns the IP address 192.0.0.0</li> <li>ApplySubnetMask(IPAddress("1234:5678:9ABC:DEF0::"), 32) returns the IP address 1234:5678::</li> </ul>
DomainRollup (string domain)	Returns domain truncated to the registered domain name and suffix. This function uses the Mozilla Public Suffix List to determine the longest applicable suffix and returns the suffix plus one additional label to the left. This allows URLs from the same top domain to be rolled up and grouped typically for use with reporting base on top level domains. For example: <ul style="list-style-type: none"> <li>DomainRollup("www.google.com") returns "google.com"</li> <li>DomainRollup("video1.youtube.co.uk") returns "youtube.co.uk"</li> <li>DomainRollup("video2.youtube.co.uk") returns "youtube.co.uk"</li> </ul>
Flow.IsApplicationProtocol(string proto1, {string proto2, {proto3,{...}}})	Returns true if flow application protocol (generic or more specific) is proto1, or proto2, etc. Only constant arguments are supported. The string values should contain short protocol IDs such as "http", "http_get", "bittorrent", and so on. (the same ones that are accepted by the protocol condition).
IDNADecode(string s)	Converts s to unicoded String if s is IDN-encoded. Otherwise does no conversion.
IpInClass(ipaddress ip, integer site, string className)	Returns true if the IP address in the given site belongs to network/policy class className. If site is not provided, the default of 0 is assumed.
NetworkName(ipaddress ip, integer site)	Returns the name of the network the IP address in the given site belongs to. If site is not provided, the default of 0 is assumed. Typically will not return NULL, since there usually is a default catch-all network in subnets.txt.
NormalizePath(string path)	Returns path in normalized form as a string. Normalization is done in a fashion similar to RFC 1808. The path is made up of a series of directories, separated by slashes. If a directory name is a single period, it is ignored. If the directory name is two periods, the directory and its previous directory are ignored. If there are extra directories named "." without preceding directories to remove, the extra "." directories are ignored. Paths are also modified to make sure they always start with a slash. This policy function can be used to normalize the resource path of an HTTP transaction to test against known paths, to ensure that the test is not circumvented by using these special directories to specify paths that look different but resolve to the same URL. For example:

Function	Description
	<ul style="list-style-type: none"><li>• <code>NormalizePath("/directory1/./index.html")</code> returns <code>"/directory1/index.html"</code></li><li>• <code>NormalizePath("/directory1/./index.html")</code> returns <code>"/index.html"</code></li><li>• <code>NormalizePath("index.html")</code> returns <code>"/index.html"</code></li></ul>
<code>PercentDecode(string s)</code>	Returns <code>s</code> with percent-encoded characters decoded back to ASCII, as per RFC 3986. This function undoes the operation performed by <code>PercentEncode</code> . This policy function can be used to decode any percent-encoding in a the resource path of an HTTP transaction to test against known paths, to ensure that the test isn't circumvented by using percent-encoding to specify paths that look different but resolve to the same URL.
<code>PercentEncode(string s)</code>	Returns <code>s</code> with special characters percent-encoded as per RFC 3986. Percent-encoding is a process where irregular characters are represented by replacing them with a percent sign, followed by that character's ASCII value in hex. Characters that are left untouched by this function are: <ul style="list-style-type: none"><li>• lower-case letters</li><li>• upper-case letters</li><li>• numbers</li><li>• dash (-)</li><li>• underscore (_)</li><li>• period (.)</li><li>• exclamation mark (!)</li><li>• tilde (~)</li><li>• asterisk (*)</li></ul> This function can be used to construct URLs for redirecting subscribers using a HTTP 307 response.
<code>PolicyClassName(ipaddress ip, integer site)</code>	Returns the name of the policy class that the IP address in the given site belongs to. If site is not provided, the default of 0 is assumed. Returns NULL if ip does not belong to any policy class.

## 3.1.8 BGP Functions

Functions to handle BGP data are:

Function	Description
<code>ASHop(ipaddress IP, integer n)</code>	Returns the <i>n</i> th (0-indexed) hop in the AS path to the subnet containing IP. Returns null if the path is not known, or the path has fewer than <i>n</i> hops. If the hop is an AS_SET (which is counted as one hop), an arbitrary hop from the set is returned. For example, if the path to IP is "[1,2,3],[4,5,6]", then <code>ASHop(IP, 0)</code> would return 1, <code>ASHop(IP, 3)</code> would return either 4, 5, or 6, and <code>ASHop(IP, 4)</code> would return null.
<code>ASHopCount(ipaddress IP)</code>	Returns the number of hops in the AS path to the subnet containing IP. Returns null if the path is not known. For example, if the path to IP is "[1,2,3,4],[5,6,7]", <code>ASHopCount(IP)</code> returns 5.
<code>ASHopIsOneOf(ipaddress IP, integer n, integer as1, {integer as2, {...}})</code>	Returns true if the <i>n</i> th hop in the AS path to the subnet containing IP is one of <i>as1</i> , <i>as2</i> , or any following argument. Returns null if the path is not known. If the hop is an

Function	Description
	AS_SET, returns whether or not the set contains any one of as1, as2, or any following argument.
ASPath(ipaddress IP)	Returns a string representing the AS path to the subnet containing IP. If none is known, returns null. The string is formatted with: <ul style="list-style-type: none"> <li>AS_SEQUENCES in []</li> <li>AS_SETs in {}</li> <li>AS_CONFED_SEQUENCES in ()</li> <li>AS_CONFED_SETs in &lt;&gt;</li> </ul>
ASPathContainsOneOf(ipaddress IP, integer as1, {integer as2, {...}})	Returns true if the AS path to the subnet containing IP includes any one of the other arguments. Returns true even if the AS number is present only in an AS_SET. Returns null if the path is not known.
EndpointAS(ipaddress IP)	Returns the endpoint AS in the AS path to the subnet containing IP. Returns null if the path is not known. If the path ends with an AS_SET, an arbitrary hop from the set is returned.
EndpointASIsOneOf(ipaddress IP, integer as1, {integer as2, {...}})	Returns true if the endpoint AS in the AS path to the subnet containing IP is equal to one of the other arguments. Returns null if the path is not known. If the endpoint is in an AS_SET, returns true if the set contains one of as1, as2, and so on.
GetCommunity(ipaddress IP, integer as)	Returns the community information associated with an AS for a subnet containing the IP. For example, GetCommunity(IP, 100) would return 10 if IP were associated with the community 100:10. Returns null if no community is known for the subnet.
HasCommunity(ipaddress IP, integer as, integer community)	Returns true if the subnet containing IP is associated with the community encoded as as:community. For instance, HasCommunity(IP, 10, 10) would return true if IP were associated with the community 10:10. Returns null if no community information is known for the subnet.
NexthopAS(ipaddress IP)	Returns the first hop in the AS path to the subnet containing IP. Returns null if the path is not known. NexthopAS(IP) is equivalent to ASHop(IP, 0).
NexthopASIsOneOf(ipaddress IP, integer as1, {integer as2, {...}})	Returns true if the next hop AS in the AS path to the subnet containing IP is equal to one of the other arguments. Returns null if the path is not known. If the next hop is in an AS_SET, returns whether or not the set contains one of as1, as2, or any following argument.

## 3.1.9 Encryption Functions

Functions to handle encrypted data are:

Function	Description
AES(string key, string text)	Encrypts the value of text using an Advanced Encryption Standard (AES) cipher and the key. The key must be base64 encoded. Use Base64Encode or HexToBase64 functions to format the key. The input string text is padded to a length that is a multiple of 16 bytes to ensure correct decryption. The padding scheme uses the syntax of PKCS#7 as specified in RFC 5652.
Base64Encode(string value)	Translates the specified string into a base 64 representation.

Function	Description
HexToBase64(string value)	Translates the specified hex string (two hexadecimal characters per byte) into a base 64 representation of the corresponding binary data. Useful for specifying keys for encryption functions in hex format, for example:  <code>AES (HexToBase64 ("00010203050607080A0B0C0D0F101112", ...))</code>
HMACMD5(string key, string text)	Produces a keyed-Hash Message Authentication Code for text using key and an MD5 hash function. Use Base64Encode or HexToBase64 functions to format the key. Returns a raw string (can be converted back to base 64 with Base64Encode). <code>Base64Encode(HMACMD5(key (in base 64), text))</code> .
RC5(string key, string text)	Encrypts the value of text using an RC5 cipher and the key. The key must be base64 encoded. Use Base64Encode or HexToBase64 functions to format the key. The input string text is padded to a length that is a multiple of 16 bytes to ensure correct decryption. The padding scheme uses the syntax of PKCS#7 as specified in RFC 5652.

## 3.2 Map Functions

Each map type provides extended functionality and optimizations targeted towards a specific use case. See the SDE SandScript Configuration Guide for additional information on Map Functions.

The different map types are:

<b>String map:</b>	Allows for a map to contain any string.
<b>Host map:</b>	Provides options to perform normalized match of hostnames.
<b>URL map:</b>	Provides option to perform normalized match of URLs. For use cases where maps contain only full URLs.
<b>IP Address map:</b>	A map that is optimized to search for IP addresses.

Each map function has this syntax:

`Map.<Name>.<Function>`

Where:

- `<Name>` is the unique name given to the map in the map definition.
- `<Function>` is the name of a function. Each function transforms input and finds a match according to optional parameters specified in the map declaration.

### 3.2.1 String Map Functions

Functions to handle a string map are:

Function Name	Description
Contains(string)	Returns true if string is matched by a key/pattern in the map, compared as specified by <code>case_sensitive</code> parameter in the map declaration.

Function Name	Description
FindLabel(string)	Returns the label associated with a key/pattern, if string is matched by a key/pattern in the map, compared as specified by the case_sensitive parameter in the map declaration. The return type is an integer/string as specified by label_type parameter in the map declaration.
FindPattern(string)	Returns the key/pattern associated with a key/pattern, if string is matched by a key/pattern in the map, compared as specified by the case_sensitive parameter in the map declaration.

## 3.2.2 Hostname Map Functions

Functions to handle a hostname map are:

Function Name	Description
Contains(hostname)	Returns true if string is matched by a key/pattern in the map, compared as specified by normalize and case_sensitive parameters in the map declaration.
FindLabel(hostname)	Returns the label/value associated with a key/pattern, if string is matched by a key/pattern in the map, compared as specified by normalize and case_sensitive parameters in the map declaration. The return type is an integer/string based on the label_type in the map declaration.
FindPattern(hostname)	Returns the key/pattern associated with a key/pattern, if string is matched by a key/pattern in the map, compared as specified by normalize and case_sensitive parameters in the map declaration.
FindHost(hostname)	Returns the host, if hostname is matched by a key/pattern in the map using a compared as specified by normalize and case_sensitive parameters in the map declaration.

## 3.2.3 URL Map Functions

Functions to handle a URL map are:

Function Name	Description
Contains(hostname, resource)	Returns true if string is matched by a key/pattern in the map, compared as specified by normalize, case_sensitive and resource_param_match_order parameters in the map declaration.
FindLabel(hostname,resource)	Returns the label/value associated with a key/pattern, if string is matched by a key/pattern in the map, using a compared as specified by normalize, case_sensitive and resource_param_match_order parameters in the map declaration. The return type is an integer/string based on the label_type in the map declaration.
FindPattern(hostname, resource)	Returns the key/pattern associated with a key/pattern, if string is matched by a key/pattern in the map, using a compared as specified by normalize, case_sensitive and resource_param_match_order parameters in the map declaration.
FindHost(hostname, resource)	Returns the host, if hostname is matched by a key/pattern in the map, compared as specified by the normalize, case_sensitive and resource_param_match_order parameters in the map declaration.
FindResource(hostname, resource)	Returns the resource, if resource is matched by a key/pattern in the map, compared as specified by normalize, case_sensitive and resource_param_match_order parameters in the map declaration.

Function Name	Description
FindUrl(hostname, resource)	Returns the URL, if URL is matched by a key/pattern in the map, compared as specified by normalize, case_sensitive and resource_param_match_order parameters in the map declaration.

## 3.3 Table Functions

Table functions take one or more arguments (in the form of expressions) and return a value.

Table functions operate on specific rows and columns of a particular table. Table functions can be used on any defined tables.

Field	Description
Table.<TableName>. All(expression e1, {expression e2, {e3, {...}}}) Table.<TableName>[Key]. All(expression e1, {expression e2, {e3, {...}}})	Returns true only if all of the arguments are true. Returns false if any of the arguments are false, even if some of the arguments are null. The arguments must all be of type boolean and of the form Table.<TableName>.<ColumnName>. This function is an optimized form of a compound AND expression that contains only boolean columns of a table (Table.<TableName>.ColumnName1 and Table.<TableName>.ColumnName2 and Table.<TableName>.ColumnName3).
Table.<TableName>. Any(expression e1, {expression e2, {e3, {...}}}) Table.<TableName>[Key]. Any(expression e1, {expression e2, {e3, {...}}})	Returns true if any of the arguments are true, even if some of the arguments are null. Returns false only if all of the arguments are false. The arguments must all be of type boolean and of the form Table.<TableName>.<ColumnName>. This function is an optimized form of a compound OR expression that contains only boolean columns of a table (Table.<TableName>.ColumnName1 or Table.<TableName>.ColumnName2 or Table.<TableName>.ColumnName3).
Table.<TableName>.Create() Table.<TableName>.Create(Key)	Returns a table row, creating the row if it does not exist, where TableName is the name of the table. Used to assign a row to a cursor local variable. Optionally specify a Key list for explicit access to a table row.
Table.<TableName>. <ColumnName>.Count() Table.<TableName>[Key]. <ColumnName>.Count()	Returns the current number of rows in the table TableName. ColumnName can be any column in the table.

## 3.4 Diameter Functions

Diameter functions reveal information about Diameter peers or the state of the Diameter subsystem.

Diameter functions take one or more arguments (in the form of expressions) and return a value. Diameter functions are available according to a dictionary defined by the Diameter subsystem.

Field	Description
Diameter.<DictionaryName>. ErrorToText(integer i)	Returns a string description of the Diameter result code provided by integer i. DictionaryName is the name of the Diameter dictionary containing the mapping from result code to description. Each Diameter dictionary makes one of these functions available to SandScript. If the expression is null, or if the result code is not defined in the dictionary, then null will be returned.

Field	Description
Diameter.PeerStatus(string s)	Returns true to indicate that the peer identified by the string s is in the connected state. Returns false otherwise. If the argument is null, or if no peer with a matching identity can be found, then false is returned to indicate a lack of connectivity.
Diameter.TransmitQueue Percent(string s)	Returns an integer value representing the percent of the outgoing queue that is full of a Diameter peer identified by the string s. If the argument is null, or if no peer with a matching identity can be found, then the integer value 100 is returned to indicate a full queue.

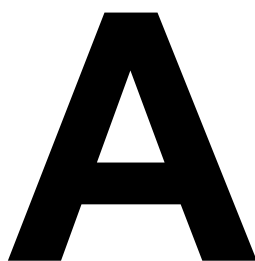
## 3.5 Subscriber Mapping Functions

Functions that retrieve a session ID are:

Function	Description
Session.GetIdFromSite(ipaddress ip, integer site-number)	Returns the identifier of a session whose key is the given IP Address and site number. Returns NULL if no session is mapped with the provided key.
Session.GetIdFromPort(ipaddress public-ip, integer port-number)	Returns the identifier of a session whose private IP maps to the provided public IP and port number in the NAT mapping table. Returns NULL if no session is mapped with the provided public IP and port number.







# Sample SandScript

This section provides some sample SandScripts. To use these SandScripts, add the desired example to **policy.conf**, and then run the `svreload` command to fetch NDS reports.

## SSL Analyzer SandScript

This SandScript measures the "Hello packets" exchanged between server and client.

```
classifier "SSL_Analyzer_Hit_Hello_Field" type string {
}

PolicyGroup expr(Flow.SessionProtocol = Protocol.ssl) {

  PolicyGroup {
    if expr( (Flow.Application.SSL.ClientHello is not null) or \
             (Flow.Application.SSL.ServerHello is not null)) \
    then \
      set Flow.Classifier.SSL_Analyzer_Hit_Hello_Field = \
        "Flows that Hit the Client and Server Hello PAL Fields"
    }

    if true then \
      set Flow.Classifier.SSL_Analyzer_Hit_Hello_Field = \
        "Flows Not-Hit by Client and Server Hello PAL Fields"
    }
}

measurement "SSL_Analyzer_Hit_Hello_Fields" \
  sum(Flow.Client.tx.bytes + Flow.Server.tx.bytes) \
  over publish_interval \
  where expr(Flow.Classifier.SSL_Analyzer_Hit_Hello_Field is not null) \
  unique by (Flow.Classifier.SSL_Analyzer_Hit_Hello_Field)
```

```
publish "SSL_Analyzer_Hit_Hello_Fields" Measurement.SSL_Analyzer_Hit_Hello_Fields
```

### WhatsApp Analyzer SandScript

This SandScript measures the total messages transmitted (tx) for all subscribers.

```
measurement "WSmob_WhatsAppMSGTotalTx" over publish_interval
```

```
publish "WSmob_WhatsAppMSGTotalTx" measurement.WSmob_WhatsAppMSGTotalTx
```

```
PolicyGroup expr(Flow.ApplicationProtocol.Parent = Protocol.Whatsapp) {  
    if expr(Flow.ApplicationProtocol = Protocol.WhatsappControl) then \  
        increment measurement.WSmob_WhatsAppMSGTotalTx by Flow.Application.IM.Tx.Messages  
}
```

### Streaming SandScript

This SandScript generates individual NDS reports for LiveStreaming, VOD, and P2P traffic.


```
measurement "StreamingVOD" sum(Flow.client.tx.bytes + flow.server.tx.bytes) \  
    over publish interval \  
    where expr(Flow.Application.Streaming.VOD) \  
    unique by (Flow.ApplicationProtocol.StatsProtocol)  
publish "StreamingVOD" Measurement.StreamingVOD
```

```
measurement "StreamingLive" sum(Flow.client.tx.bytes + flow.server.tx.bytes) \  
    over publish interval \  
    where expr(Flow.Application.Streaming.Live) \  
    unique by (Flow.ApplicationProtocol.StatsProtocol)  
publish "StreamingLive" Measurement.StreamingLive
```

```
measurement "StreamingP2P" sum(Flow.client.tx.bytes + flow.server.tx.bytes) \  
    over publish interval \  
    where expr(Flow.Application.Streaming.P2P) \  
    unique by (Flow.ApplicationProtocol.StatsProtocol)  
publish "StreamingP2P" Measurement.StreamingP2P
```





A large, light green stylized 'X' logo is centered on the page. The 'X' is composed of four curved, overlapping strokes that intersect in the middle. The background is white, and a large green circular shape is partially visible on the right side of the page.

Sandvine Incorporated  
408 Albert Street  
Waterloo, Ontario, Canada  
N2L 3V3

Phone: (+1) 519-880-2600  
Fax: (+1) 519-884-9892

Web Site: [www.sandvine.com](http://www.sandvine.com)