



# **SDE Release 6.40.00, SandScript Configuration Guide**

**05-00244-A02  
2013-9-12**

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).

# Contents

|   |    |
|---|----|
| 1 SandScript Configuration.....               | 7  |
| 1.1 Introduction to SandScript.....           | 8  |
| 1.1.1 Rules.....                              | 8  |
| 1.1.2 SandScript Process.....                 | 9  |
| 1.2 SandScript Configuration Files.....       | 9  |
| 1.2.1 Control Center.....                     | 9  |
| 1.3 SandScript File.....                      | 10 |
| 1.3.1 Creating a SandScript File.....         | 10 |
| 1.3.2 svreload.....                           | 10 |
| 1.3.3 svreload -maps.....                     | 11 |
| 1.4 The subnets.xml File.....                 | 11 |
| 2 SandScript Grammar.....                     | 13 |
| 2.1 Policy Rule Structure.....                | 14 |
| 2.1.1 Rule Evaluation.....                    | 14 |
| 2.1.2 Logic.....                              | 14 |
| 2.1.3 SandScript Policy Groups.....           | 15 |
| 2.1.4 Include Statements.....                 | 15 |
| 2.2 Attributes.....                           | 16 |
| 2.2.1 Declaring Attributes in SandScript..... | 17 |
| 2.2.2 Enumerated Attributes.....              | 17 |
| 2.2.3 Non-enumerated Attributes.....          | 17 |
| 2.2.4 Timestamp Attribute.....                | 18 |
| 2.2.5 Local Attributes.....                   | 19 |
| 2.2.6 Session Attributes.....                 | 19 |
| 2.3 Conditions.....                           | 19 |
| 2.3.1 Any.....                                | 19 |
| 2.3.2 Expressions.....                        | 19 |
| 2.3.3 Time of Day.....                        | 20 |
| 2.3.4 True.....                               | 20 |
| 2.4 Actions.....                              | 21 |
| 2.4.1 Decrement.....                          | 21 |
| 2.4.2 Delete Row.....                         | 21 |
| 2.4.3 Deserialize.....                        | 21 |
| 2.4.4 Diameter.....                           | 23 |
| 2.4.5 Event.....                              | 25 |
| 2.4.6 Increment.....                          | 25 |

|   |    |
|---|----|
| 2.4.7 Record Decrement.....                 | 26 |
| 2.4.8 Record Increment.....                 | 26 |
| 2.4.9 Record Reset.....                     | 26 |
| 2.4.10 Record Set.....                      | 26 |
| 2.4.11 Set.....                             | 27 |
| 3 SandScript Definitions.....               | 29 |
| 3.1 Local Variables.....                    | 30 |
| 3.2 Classifiers.....                        | 31 |
| 3.3 Timers.....                             | 32 |
| 3.3.1 Timer Arm.....                        | 33 |
| 3.3.2 Timer Clear.....                      | 33 |
| 3.4 Table Syntax.....                       | 34 |
| 3.4.1 Accessing Table Rows.....             | 35 |
| 3.4.2 Table Clear.....                      | 36 |
| 3.4.3 Table Decrement.....                  | 36 |
| 3.4.4 Table Increment.....                  | 36 |
| 3.4.5 Table Persistence.....                | 36 |
| 3.4.6 Table Reset.....                      | 37 |
| 3.4.7 Table Set.....                        | 37 |
| 3.4.8 Table Timer Arm.....                  | 37 |
| 3.4.9 Table Timer Clear.....                | 37 |
| 3.5 User-defined Functions.....             | 37 |
| 3.6 Foreach.....                            | 38 |
| 3.7 Events.....                             | 39 |
| 3.8 Published Expressions.....              | 40 |
| 4 Measurements.....                         | 43 |
| 4.1 Measurements Overview.....              | 44 |
| 4.1.1 Measurements.....                     | 44 |
| 4.1.2 Generic Measurement.....              | 45 |
| 4.1.3 Sum Measurement.....                  | 46 |
| 4.1.4 Top Measurement.....                  | 47 |
| 4.1.5 Expression Measurement.....           | 48 |
| 4.1.6 Histogram Measurement.....            | 48 |
| 4.2 Measurement Groups.....                 | 49 |
| 4.3 Histograms.....                         | 49 |
| 4.3.1 Linear Histogram.....                 | 49 |
| 4.3.2 Custom Histograms.....                | 50 |
| 5 Subscriber to IP Mapping.....             | 53 |
| 5.1 Subscriber Aware SandScript Policy..... | 54 |

|   |    |
|---|----|
| 5.1.1 Updating SandScript to Uniquely Identify Subscribers.....     | 54 |
| 5.1.2 Subscriber Lookup.....  | 54 |
| 5.1.3 Populating Subscriber to IP Address Mappings.....             | 55 |
| 5.2 Subscriber Database Configuration.....                          | 55 |
| 5.3 Disabling Secure Tunnels Between the SPB and a PTS Element..... | 56 |
| 5.4 Troubleshooting Subscriber Mapping.....                         | 57 |
| 6 Maps.....   | 59 |
| 6.1 Map Overview.....   | 60 |
| 6.2 String Maps.....  | 60 |
| 6.2.1 SandScript Functions for a String Map.....                    | 61 |
| 6.3 Hostname Maps.....  | 62 |
| 6.3.1 SandScript Functions for Hostname Maps.....                   | 62 |
| 6.4 URL Maps.....   | 63 |
| 6.4.1 SandScript Functions for URL Maps.....                        | 63 |
| 6.5 IP Address Maps.....  | 64 |
| 6.6 Loading Map Contents.....                                       | 65 |
| 6.7 Map File Format.....  | 65 |
| 6.7.1 Glob Matching.....  | 66 |
| 6.7.2 Normalization for Hostname and URL Maps.....                  | 66 |
| 7 Diameter Stack Configuration.....                                 | 67 |
| 7.1 The Diameter Stack.....   | 68 |
| 7.2 Diameter Stack Base Protocol Messages.....                      | 68 |
| 7.2.1 CER Messages.....   | 68 |
| 7.2.2 CEA Messages.....   | 69 |
| 7.2.3 DWR Messages.....   | 69 |
| 7.2.4 DWA Messages.....   | 70 |
| 7.2.5 Outgoing DPR Messages.....                                    | 71 |
| 8 DHCP Actions.....   | 73 |
| 8.1 DHCP Actions.....   | 74 |
| 8.2 LeaseQueryByIp Syntax.....                                      | 74 |
| 8.3 DHCPv6.LeaseQueryByClientID Syntax.....                         | 74 |
| 8.4 DHCPv4.LeaseQueryByMac Syntax.....                              | 75 |
| 9 Logging.....  | 77 |
| 9.1 Logging.....  | 78 |
| 9.2 Logging.<LogId>.<DestinationId>.Write Syntax.....               | 78 |
| 9.3 Logging.<LogId>.<DestinationId>.Flush Syntax.....               | 78 |
| 9.4 Logging.<LogId>.<DestinationId>.Close Syntax.....               | 79 |
| 9.5 Logging Example.....  | 79 |
| 10 RADIUS Actions.....  | 81 |

---

|  |     |
|--|-----|
| 10.1 RADIUS Actions.....   | 82  |
| 10.2 RADIUS Server Action.....   | 82  |
| 10.3 RADIUS Client Action.....   | 82  |
| 11 Provisioning.....   | 85  |
| 11.1 Provisioning.Session.ProvisionSubscriber().....                   | 86  |
| 11.2 Provisioning.Nat.Map or Unmap.....                                | 87  |
| 11.3 Provisioning.Subscriber.Get() .....                               | 88  |
| 11.4 Provisioning.Subscriber.Delete.....                               | 89  |
| 12 ID Allocation .....   | 91  |
| 12.1 IdAllocation.Lookup () .....                                      | 92  |
| 12.2 IdAllocation.Delete() .....                                       | 93  |
| 13 LDAP .....  | 95  |
| 13.1 Ldap.GetAttributes () .....                                       | 96  |
| 13.2 Ldap.Get() .....  | 96  |
| 13.3 Ldap.Search() .....   | 97  |
| 14 External Commands.....  | 99  |
| 14.1 External Commands.....  | 100 |
| 14.2 External Command Syntax.....                                      | 100 |
| 15 Troubleshooting SandScript.....                                     | 101 |
| 15.1 Tips for Using Unique By.....                                     | 102 |
| 15.2 Message Logs.....   | 102 |
| 15.3 Informative CLI Commands.....                                     | 102 |
| 15.4 Error Handling for LDAP Subsystem when Input Queue is Full .....  | 102 |
| 15.5 Error Handling for LDAP Subsystem when Output Queue is Full ..... | 103 |



# 1

## SandScript Configuration

- ["Introduction to SandScript" on page 8](#)
- ["SandScript Configuration Files" on page 9](#)
- ["SandScript File" on page 10](#)
- ["The subnets.xml File" on page 11](#)

# 1.1 Introduction to SandScript

SandScript is an event-based policy definition language that lets Communications Service Providers (CSPs) fully leverage the Sandvine policy engine with programmatic flexibility.

Sandvine's policy engine can be thought of as a "green box" into which conditions and subscriber entitlements flow, and out of which charging updates, management actions, and signals emerge. It evaluates policy logic on the lowest element in the network that has access to the requisite information. This approach maximizes scalability by reducing centralization, and maximizes performance by removing unnecessary communication between elements, thereby avoiding conditions of signaling overload.

These components work in concert and leverage the policy engine:

- The Policy Traffic Switch (PTS) identifies and measures traffic, makes local policy decisions and implements enforcement actions (fulfilling the functions of a PCEF).
- The Service Delivery Engine (SDE) is a policy decision and enforcement element that, like a PCRF, works together with the PTS to provide unified network policy control within multi-vendor and multi-access networks.
- The Subscriber Policy Broker (SPB) acts as a subscriber profile repository (SPR) and provides long-term storage for comprehensive business intelligence insight.

SandScript can be used to:

- Evaluate business logic rules in real-time.
- Enforce policy actions that affect network conditions and subscriber experiences.
- Define measurements and collect statistics for business intelligence insight into network traffic, subscriber usage, application quality of experience, and more.
- Configure the function of the PTS and SDE elements under specific circumstances.

## 1.1.1 Rules

The SandScript policy language is based on a collection of rules composed of conditions and actions.

Rules may be applicable in one or more contexts, such as subscriber context or Diameter context. Whenever the conditions of a rule evaluate to true, the actions of that rule are performed.

Conditions select events of a specific type or having a specific attribute. Examples of conditions include:

- Subscriber properties such as a tier (for example gold tier).
- A server within the network or external to the network.
- A time of day such as from 9 AM to 5 PM.
- Events generated by protocol subsystems such as DHCP, RADIUS, or Diameter.

Actions are performed on the subset of events selected by the condition. An action is applied while the condition remains true. Some examples of actions are:

- Set a subscriber attribute.
- Modify state in a policy table.
- Send a RADIUS or Diameter message.
- Generate a logging record.
- Execute a system command.



Conditions and actions are used together: determine what action you want performed on the traffic, then create the condition to filter the traffic on which to perform the action.

## 1.1.2 SandScript Process

The Service Deliver Engine (SDE) communicates with enforcement devices and other systems to implement multi-vendor network policies, called SandScript, that span many enforcement points. SandScript monitors events and performs some level of action on them.

# 1.2 SandScript Configuration Files

Use Control Center's **Power Edit** or **Quick Edit** to modify and manage these configuration files:

| File Location                       | Description   | Platform    |
|-------------------------------------|---|-------------|
| /usr/local/sandvine/etc/subnets.txt | Defines a text-based map of the network labeling internal versus external IP subnets. Internal and external designations are relative to the PTS and its interfaces.  | PTS         |
| /usr/local/sandvine/etc/subnets.xml | Defines an XML map of the network identifying network and policy classes. It associates the names of classes with groups of IP addresses or subnets and assigns attributes to the network and policy classes. | SDE         |
| /usr/local/sandvine/etc/policy.conf | Contains SandScript rules and actions that act on traffic or events.  | PTS and SDE |

## 1.2.1 Control Center

Control Center is Sandvine's unified policy and operations management graphical user interface, providing a single mechanism for monitoring operational information, editing network policies, configuring elements, and deploying network policy control solutions.

Control Center lets communications service providers (CSPs) centrally create and deploy service policies for the entire network in response to new opportunities or trends, and simplifies all aspects of Sandvine operations management, delivering real-time information and granular control.

Control Center lets you safely configure your policy in isolation from the physical devices in order to control when new policy behavior is enforced. Control Center is also a repository of configuration information for the elements (PTS, SDE, and SPB). The elements are responsible for managing real-time data and enforcing policy. Control Center manages all policy and configuration changes in its database. The migration of policy and configuration to the elements is called a deployment. Deployment is not automatic; you must start it manually.

### 1.2.1.1 Online Help

For further information about Control Center, see the online help. Control Center's policy creation wizards contain help content and tool tips are available when you hover your mouse over a GUI button.

## 1.3 SandScript File

A SandScript policy file is a text file consisting of definitions and rules.

Definitions in the SandScript file create complex constructs to be used in later definitions and rules. Rules consist of conditions and actions. A policy group can be made of rules and nested groups. A policy file implements and stores these SandScript components. The main SandScript file is `policy.conf`, but it can include several other SandScript policy files. The main components in a SandScript file are:

- Definition – Defines an attribute by giving it a name and a type.
- Rule – Describes the logic and actions of the SandScript policy.
- Include – Adds the rules from other policy files to this policy file.

### 1.3.1 Creating a SandScript File

The `policy.conf` file is the only SandScript file that is loaded onto the PTS or the SDE policy engine.

A sample SandScript file is available on each PTS or SDE element at `/usr/local/sandvine/etc/policy.conf.sample`. You can copy, then edit this sample file.

1. As an administrative user, connect to the element.
2. Copy the `/usr/local/sandvine/etc/policy.conf.sample` file to `/usr/local/sandvine/etc/policy.conf`.
3. Edit `/usr/local/sandvine/etc/policy.conf` as appropriate.  
Use Control Center or the `edit policy` CLI command.
4. To reload the configuration files on each SDE element, run `svreload`.

### 1.3.2 svreload

After you make changes to SDE subsystem configurations or to `policy.conf`, you are required to perform an `svreload` to enable the changes:

- From the SDE shell, run:

```
svreload
```

A response similar to this appears:

```
Apply configuration to scdpd. Send reload to scdpd.....[OK]
```

```
Apply configuration to msd. Send reload to msd.....[OK]
```

```
Apply configuration to ecd. Send reload to ecd.....[OK]
```

```
Apply configuration to svside. Send reload to svside.....[OK]
```

```
All applications were successfully reloaded
```

### 1.3.3 svreload -maps

The `svreload -maps` command allows you to reload the contents of external map files after changing their values, without performing a full `svreload`. This is much faster than a full reload.



**Caution:**

This command should not be used if you change the map name, the name of the external file or anything in the actual `policy.conf`. In any of these cases, you should perform a full `svreload`, which loads the policy file with the map declarations, which indicates the map names, and the external file holding all the values.

To reload only the values in the external map files, from the SDE shell run:

```
svreload -maps
```

## 1.4 The subnets.xml File

The SDE must have a `subnets.xml` file which identifies network and policy classes.

The `subnets.xml` file is a map of the network. This file associates the names of classes with groups of Internet Protocol (IP) addresses or subnets and assigns attributes to the network and policy classes. The file is located at: `/usr/local/sandvine/etc`.

The syntax is:

```
<SubnetList>
  <Subnet>0.0.0.0/0</Subnet>
  <Subnet>::/0</Subnet>
</SubnetList>
```

Where each subnet must be on a separate line and Subnet 0 is not filtered by the SDE.

You can edit this file using any text editor. You must issue a reload for your changes to take effect.

You can use the `subnets.xml` file to control the session notifications from the SPB that the SDE listens to. If a notification is outside of the configured subnets the SDE ignores it. In most cases this is left at the default which instructs the SDE to process session notifications for all sessions.





# 2

## SandScript Grammar

- ["Policy Rule Structure" on page 14](#)
- ["Attributes" on page 16](#)
- ["Conditions" on page 19](#)
- ["Actions" on page 21](#)

## 2.1 Policy Rule Structure

Rules take the format:

```
if <condition(s)> then <action(s)>
```

For example:

```
if expr(Policy.reloaded) then set measurement.SomeMeasurement = 0
```

reads if <condition>, policy has been reloaded, is true then perform <action>, reset SomeMeasurement.

General guidelines for rules are:

- Rules are case sensitive. In general, all keywords are lower case with dashes separating words. Always use lower case unless upper case is specified.
- Any line starting with a pound sign (#) is commented out and ignored.
- White space is ignored.
- Rules can be broken into multiple lines by using the backslash (\) to escape the carriage return.
- Strings must be enclosed in single or double quotes. Use single quotes if you want the string to include double quotes and vice versa.
- Any line that does not start with an identifier will give an error in `/var/log/svlog`.

The conventions used when documenting rules are:

- Optional parameters are in braces { }.
- Angle brackets <> are used to identify arguments where you must supply the appropriate value.
- Mutually exclusive options are indicated by the "|" symbol.
- A backslash (\) at the end of a line indicates that the string is continued on the next line.

### 2.1.1 Rule Evaluation

SandScript policy rules are evaluated on many different types of events, and periodically on existing objects on the element.

A policy rule may apply to one or more events, but they are only evaluated on events where they make sense. The set of events where a rule can be evaluated is known as the scope of the rule.

Specifically, traffic rules are re-evaluated when subscriber attribute changes are pushed to the SDE from the SPB.

### 2.1.2 Logic

Modifiers can be used to specify logic operations in rules.

The available operators are:

- `and` - evaluates to true if both conditions are met
- `or` - evaluates to true if either condition is met
- `not` - evaluates to true if the condition is not met

When using more than one logical operator, use parentheses () to indicate precedence. The `and` operator takes precedence over the `or` operator.

## Actions

Rules can cause multiple actions to take place. To accomplish this, an `and` operator can be used with compatible actions.

## 2.1.3 SandScript Policy Groups

Rules can be organized into policy groups.

A policy group is a method of grouping a number of rules where the rule conditions are evaluated and by default only the actions of one rule in the list are performed. The actions specified in the first rule that matches the condition are performed and any remaining rules in the list are skipped. This speeds up processing. For example, a policy group could be used to apply rules based on time of day criteria.

### Policy group with conditions

Conditions can be placed directly on policy groups. The syntax is:

```
PolicyGroup condition(s) {  
    rules and/or nested policy groups...  
}
```

The rules within the policy group are only evaluated if the group condition is matched. By default, a policy group stops evaluating its rules when the first match is found, essentially creating an if/elseif/else... decision tree. When a matching rule is found, its actions are performed and any remaining rules and groups are skipped. If the condition of a nested policy group evaluates to true, this also causes any remaining rules and groups to be skipped.

The scope of a policy group is the intersection of the scope of all the rules/groups inside. So a single scope-restricting field nested deep in the group forces the whole group to have the scope of that field. If the intersection results in the empty set, there is a “conflicting scopes” error on reload.



#### Example:

In this example, the policy group ensures that the action applies to the user Bob, and only Bob. For a login event, action 1 is applied, for a logout event, action 2 is applied, if it is neither of these then action 3 is applied and, if the user is not Bob, action 4 is applied.

```
PolicyGroup {  
    PolicyGroup expr(RADIUS.'Accounting-Request'. 'User-Name' = "Bob") {  
        if expr(RADIUS.'Accounting-Request'. 'Acct-Status-Type' = 1) then action1  
        if expr(RADIUS.'Accounting-Request'. 'Acct-Status-Type' = 2) then action2  
        if true then action3  
    }  
    if true then action4  
}
```

### “all” policy group

An “all” policy group evaluates all the rules in the group if the group condition matches. This is a convenient way to write multiple rules that share one or more common conditions.

```
PolicyGroup all <condition(s)> {  
    rules and/or nested policy groups...  
}
```

## 2.1.4 Include Statements

SandScript policy files can be nested.

By adding an include statement to the policy file, additional policy files can be implemented. The file at `/usr/local/sandvine/etc/policy.conf` is always loaded. Other policy files are only loaded if they are included in the `/usr/local/sandvine/etc/policy.conf` file or nested within files included in it.

Include statements have the syntax:

```
include "<filepath>"
```

Where `<filepath>` is the full path name of the file to include. For example, `/usr/local/sandvine/etc/policy2.conf`

Include statements can appear anywhere in a SandScript policy file. The rules contained in the included file are evaluated in order, starting from the location where the file is included. See [Rule Evaluation](#) on page 14 for more information.

## 2.2 Attributes

Attributes provide a mechanism for associating data with a subscriber or with a subscriber sessions. SandScript can be used to define attributes and to associate them with one or more subscribers.

The attributes that are attached to subscribers can be used to trigger SandScript-defined management of the subscribers and their flows. The limitations for attributes are:

- An attribute name cannot exceed 128 characters.
- An attribute value cannot exceed 4000 characters.
- Attribute names must start with a letter or an underscore and can only contain alphanumeric characters and underscores (`_`).
- The maximum number of attributes that may be defined in SandScript is 1000.

Attributes have a few important properties which are described below.

### Enumerated versus Non-enumerated

Enumerated attributes are string-based attributes that can only take on certain pre-defined values. These values are defined in SandScript when the attribute is declared.

Non-enumerated attributes are typed attributes that can assume any value that is valid for their type. Supported types are strings, integers, floating point values, timestamps (ISO-8601), IP addresses, and booleans.

### Local versus Non-local

Local attributes are lightweight attributes that are not persisted to the subscriber database. This causes them to consume fewer system resources. However, the lack of persistence means that local attributes cannot be reported on using Network Demographics, and these attributes are only defined on the element on which they are created (not cluster-wide).

Non-local attributes are persisted to the subscriber database, making them more resource-intensive. However, they can be reported on and they are cluster wide.

### Subscriber-based versus Session-based:

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

In contrast, a session attribute is associated with a single subscriber session. When that session ends, the attribute is no longer defined, even if the subscriber has other active sessions. Subscriber attributes are writable in SandScript but session attributes are read-only in SandScript. Session attributes are typically gleaned from a DIAMETER/RADIUS login session.

**Note:**

Session attributes are only supported when all SPBs in the cluster node are of release 5.60 or greater.



## Record Attributes

Record attributes are a collection of attributes under one name. Each attribute in the collection can have its own type and store a unique value. These attributes correspond to RADIUS-mapped or DHCP-mapped attributes on the SPB, where multiple values from the RADIUS or DHCP packet are stored in a single SPB attribute.

Record attributes are read-only and cannot be defined as local attributes. Additionally, record attributes are only accessible through SandScript fields - see the PTS or SDE *SandScript Reference* for more information on the `Subscriber.Attribute` and `Session.Attribute` fields.

## 2.2.1 Declaring Attributes in SandScript

Attribute declarations must precede their use elsewhere in SandScript policy.

## 2.2.2 Enumerated Attributes

```
attribute {subscriber|session} "<attribute_name>" values "<value1>" "<value2>" "<value3>" ...
```

Where:

- {subscriber|session} defines whether the attribute is attached to a subscriber or a session. Defaults to subscriber.
- <attribute\_name> is the name of the attribute
- <value1> and so on are the values.

For example:

```
# Declare a subscriber attribute that describes a subscriber's bandwidth usage.
# This is a subscriber attribute because the session keyword was omitted.
attribute "subscriber_bandwidth" values "light_user" "heavy_user" "abuser" "top_talker"
# Declare a session attribute for Diameter Credit Control Request (CCR) types
attribute session "diameter_ccr" values "initial" "update" "termination"
```

## 2.2.3 Non-enumerated Attributes

```
attribute {subscriber|session} "<attribute_name>" type <attribute_type>
attribute {subscriber|session} "<attribute_name>" type record \
(<attribute_type> "<field_name_1>" {, <attribute_type> "<field_name_2>" ...})
```

Where:

- {subscriber|session} defaults to subscriber.
- <attribute\_name> is the name of the attribute.
- <attribute\_type> defines the type of the attribute. Supported types include string, integer, float, boolean, timestamp, ipaddress.

For example:

```
# Declare an integer attribute to count a subscriber's flows
attribute "subscriber_flow_count" type integer
# Declare a timestamp attribute to store a subscriber's login time
attribute session "login_time" type timestamp
# Declare a record attribute to store radius-mapped attributes for a subscriber session
attribute session "radius_info" type record ( \
    ipaddress "framed_ip_address", \
    string "account_session_id", \
    integer "calling_station_id" )
```

Type ranges for non-enumerated attributes are:

- Integer values can range from -9223372036854775808 to 9223372036854775807.
- Float values can be defined with values such as 3.1415926 or 1e10.
- Boolean attributes are stored in database as either true or false. These values are interpreted as true:
  - "t" or "T"
  - "true" or "TRUE"
  - "y" or "Y"
  - "yes" or "YES"
  - any non-zero number, such as "1", "1000", "+1", "-1", or ".5"
- IP Addresses support IPv4 and IPv6 addresses. IPv6 addresses with the IPv4-mapped IPv6 prefix of 0:0:0:0:ffff::/96 are interpreted as IPv4 addresses. IP address type cannot be converted to or from any other field type except string.

## 2.2.4 Timestamp Attribute

Timestamp is a data type. Timestamp attributes are integers which represent the number of seconds since the UNIX epoch (January 1, 1970 at 00:00:00 UTC).

```
attribute "<attribute_name>" type timestamp
local_attribute "<attribute_name>" type timestamp
```

Where:

- Timestamp attribute values are stored in the database in ISO-8601 compliant format `yyyymmddThhmmssZ` always as UTC. For example, `20071231T2359Z`. Timestamp attributes are equivalent integer attributes.
- Timestamp string parsing supports a subset of ISO 8601, date followed by time followed by time zone specification.
- Only the date part is required. Everything else is optional.
- Dates may be expressed as either `yyyy-mm-dd` or `yyyymmdd`.
- The time component is optional. If present, it follows the date. The date and time are separated with the letter T.
- Minutes and seconds are optional. Hour, minute, and second components must have exactly 2 digits.
- Time specification may contain fractional seconds after a dot. This is ignored when parsing (for example, `23:59:59.999`).
- If the time zone is not included, UTC is assumed.
- Two formats are supported. The letter Z which means UTC, or the offset in hours and minutes from GMT. The colon (:) is optional.



### Example:

UTC time examples:

```
20071231T235959Z means UTC
20071231T235959 means UTC (by default)
20071231T235959+02:00 means UTC + 2 hours
20071231T235959-0430 means UTC - 4 hours and 30 minutes
20071231T235959Z-06 means UTC - 6 hours
```

Local time examples:

```
2007-12-31T23 means Dec 31, 2007 23:00:00
20071231T23:59 means Dec 31, 2007 23:59:00
20071231T2359 means Dec 31, 2007 23:59:00
20071231T23:59:59 means Dec 31, 2007 23:59:59
20071231T235959 means Dec 31, 2007 23:59:59
```

## 2.2.5 Local Attributes

Local attributes have similar syntax to non-local attributes, except they begin with the `local_attribute` keyword. In addition, local attributes are always session-based since they are not intended to be persistent.

```
local_attribute {session} "<attribute_name>" values "<v1>" "<v2>" "<v3>" "<v4>" ...  
local_attribute {session} "<attribute_name>" type <attribute_type>
```

Where `<attribute_type>` defines the type of the attribute. Supported types include string, integer, float, boolean, timestamp, ipaddress.



**Example:**

```
local_attribute "subscriber_type" values "light_user" "heavy_user" "abuser" "top_talker"  
local_attribute "local_login_time" type timestamp
```

## 2.2.6 Session Attributes

The optional `{session}` modifier can be used in an attribute declaration to specify if the attribute is session-based or subscriber-based.

The primary difference between these two modes is the lifetime of the attribute. Subscriber-based attributes can be accessed in SandScript as long as a subscriber is mapped. However, session-based attributes are defined only for a single subscriber-ip session. Note that local attributes are always session-based, but all other attribute types can be either session or subscriber based.



**Example:**

```
attribute session "subscriber_bandwidth" values "light_user" "heavy_user" "abuser"  
attribute subscriber "subscriber_flow_count" type integer  
local_attribute session "local_login_time" type timestamp
```

## 2.3 Conditions

Conditions are used to identify traffic or subscribers to which specific actions should be applied.

Conditions can be combined or modified using logical operators. Brackets `()` should always be used to indicate precedence.

### 2.3.1 Any

A token that can be used in a condition.

It evaluates to true if the expression has a non-NULL value. It should not be used in quotes. For example:

```
if time wday = any
```

### 2.3.2 Expressions

Expressions are extended SandScript policy syntax for accessing the fields provided by SandScript.

A SandScript expression takes the form of a condition where it is enclosed within an `expr ( )` operator. Expressions are defined in terms of fields, constants, operators, and other expressions. For detailed information on expressions, refer to the SDE or PTS *SandScript Reference*.

## 2.3.3 Time of Day

The time of day condition lets you limit an action to a specific time frame.

```
time wday <wday_list>
time wday <wday_list> hours <hhmm-hhmm> {hours <hhmm-hhmm>}...
time hours <hhmm-hhmm> {hours <hhmm-hhmm>}...
```

Where:

- <wday\_list>** Days of the week are specified using a three character abbreviation for the day (Mon) or by spelling out the entire day (Monday). The first character in the string can be either upper or lower case, with the remaining characters being lower case (Mon is acceptable, MON is not).
- Week day abbreviations are Mon, Tue, Wed, Thu, Fri, Sat and Sun. A range of days is specified using a dash (Mon-Fri, Friday-Monday). For example:
- wday Mon, Wed, Fri indicates Monday, Wednesday and Friday
  - wday Wednesday-Saturday indicates the range from Wednesday to Saturday
- <hhmm-hhmm> {GMT}** Hours when the condition is valid is indicated by a range. Time conditions may be followed by GMT which indicates that days and times are in Coordinated Universal Time (UTC). When GMT is not present, local time is assumed. Note that only local time and UTC are supported.
- For example: hours 0800-1800 indicates the hours from 8 AM to 6PM, local time. 2400 can not be used to indicate midnight. For example, to specify 5 PM to midnight, use 1700-0000.
- Note that if a weekday and hours are both specified, the hours can extend beyond the specified week day. For example if Friday and a time of hours 1800-0200 is specified, the rule will extend from 6 PM on Friday to 2 AM on Saturday.



### Example:

In this example, subscribers in the bronze class are shaped during prime time hours - Monday to Friday from 1800 to 0200 and on weekends.

```
if sender "servicelevel"="bronze" and (time hours 1800-0200 \
or time wdays sat, sun) then shape to client shaper \
"httpserver" unique by Flow.Client.Stream.Http.Host
```

## 2.3.4 True

The true condition indicates that the rule is always true.

```
if true then <action>
```

## 2.4 Actions

All policy rules must have an action. Use actions to specify a behavior when a condition is matched. Multiple actions can be associated with a single condition by combining them using the `and` operator.

```
if <conditions> then <action> {and <action>}
```

### 2.4.1 Decrement

The decrement action decrements the value of the specified SandScript field.

Decrement is only compatible with writeable integer fields. By default, the value of the field is decremented by 1. The optional `by` syntax can be used to decrement the value by some other value. When it is used to decrement the value of a subscriber attribute, the attribute expiry time can be set using the `for` syntax.

```
decrement <field> {by <expression>} {for <time>}
```

Where:

|                   |  |
|-------------------|--|
| <b>field</b>      | An integer SandScript field that can be set  |
| <b>expression</b> | An integer SandScript expression that defines the value to decrement by.   |
| <b>time</b>       | An integer SandScript expression plus a unit of time, for example, “10 minutes” or “LocalTime(2010,01,01) - Now seconds”. Specifies when a subscriber attribute decremented by the action will expire and become null. |

### 2.4.2 Delete Row

Use the `delete_row` action to explicitly delete rows from a table.

```
delete_row <row1> {, <row2>} {, <row3>} ...
```

Where the rows are SandScript fields representing table rows. If the table is implicitly indexed, you do not have to specify the row. For example:

```
Table.<TableName>  
Table.<TableName>{Key}  
cursorLocalVariable
```

### 2.4.3 Deserialize

Use the `deserialize` action to deserialize binary data into writable SandScript fields according to a specified format. This action is the logical inverse of the `serialize` function.

```
deserialize( \  
  input:input_expression, \  
  format:format_expression, \  
  'output{0}':output_expression, \  
  'output{1}':output_expression, \  
  'output{2}':output_expression, ...)
```

Where:

|                          |   |
|--------------------------|---|
| <b>input_expression</b>  | The expression to deserialize.          |
| <b>format_expression</b> | The format of the input.                |
| <b>output_expression</b> | An expression used to store the output. |

Each deserialized value is assigned to an output expression. Output expressions must be writeable (for example, local variables) and compatible with the types specified in the format string. The number of output expressions must exactly match the number of items to be deserialized as specified in the format argument.

When deserializing an input that does not have enough data to fill all output expressions, as many expressions as possible will be deserialized and the rest will be set to null.

### 2.4.3.1 Format String

Each format character in a format string, except padding, requires an additional output expression to be provided as an argument to the deserialize action. The format string must be composed of these format characters:

| Type              | Characters  | Notes  |
|-------------------|---|--|
| Integer           | <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>  | May be followed by a "u" to indicate an unsigned value |
| Floating point    | <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>   | Supports IEEE floating point values only               |
| Padding           | <ul style="list-style-type: none"> <li>x - skip 1 byte of character padding</li> <li>x&lt;n&gt; - to skip &lt;n&gt; number of bytes</li> <li>x"*" - to ignore the remainder of the input. This option is only allowed if the padding is the last item in the format.</li> </ul>   |  |
| Strings           | <ul style="list-style-type: none"> <li>a - to indicate a string</li> <li>a&lt;n&gt; - to specify a fixed length of &lt;n&gt;</li> <li>a"*" - if an arbitrary length string must be deserialized. The "*" option is only allowed if the string is the last item in the format.</li> </ul>  |  |
| IP address values | <ul style="list-style-type: none"> <li>Z - IP address, network order. Only allowed if the IP address is the last item in the format string. If there are greater than 16 bytes left in the input expression, an IPv6 address is read and if there are 16 bytes, an IPv4 address is read. Less than 16 bytes is read as a NULL value.</li> <li>Z4 - IPv4 address, network order</li> <li>Z6 - IPv6 address, network order. May also be used to deserialize an IPv4-mapped-IPv6 address.</li> <li>z4 - IPv6 address, little endian</li> </ul> |  |



#### Example:

This example gets the port and address from a string and puts them into corresponding local variables. It assumes a string input that contains 2 bytes of padding, a 2-byte port number, and a 4-byte IPv4 address, where both port and address are in network (big endian) order. The format string, "x2SZ4", function contains:

- x2 for 2 bytes of padding.
- S for a big-endian 16-bit integer.
- Z4 for an IPv4 address.

If the `binary_data` input is smaller than 8 bytes, the `addr` result will be null; if the `binary_data` is smaller than 4 bytes then both `port` and `addr` will be null.

```
function "extract_port_and_address" (string "binary_data", \
                                   writable ipaddress "addr", \
                                   writable integer "port_number"){
    if true then \
        deserialize(input: binary_data, format: "x2SZ4", \
                    'output[0]': port_number, 'output[1]': addr)
}
```

## 2.4.4 Diameter

Diameter actions attempt to create and send a message to a remote Diameter peer. The actions have static built-in arguments and dynamic arguments. Diameter actions correspond to Diameter commands defined in a dictionary.

```
Diameter.<DictionaryName>.<CommandName>-<CommandSuffix>(ArgName1:arg_expression1,
ArgName2:arg_expression2, ...)
```

Where:

|                       |   |
|-----------------------|---|
| <b>DictionaryName</b> | is the name of the dictionary that defines the command, for example, "Gy".  |
| <b>CommandName</b>    | is the name of the Diameter command from the dictionary, for example "CC".  |
| <b>CommandSuffix</b>  | is one of: <ul style="list-style-type: none"> <li>• Request sets the r-bit in the Diameter message header.</li> <li>• Answer clears the r-bit in the Diameter message header.</li> <li>• Answer-Error clears the r-bit in the Diameter message header and sets the e-bit in the Diameter message header.</li> </ul> |

Static arguments are:

| Argument      | Description   | Type    | Required (Yes/No)  |
|---------------|---|---------|--|
| Proxiable     | Sets the p-bit in the outgoing Diameter message header. If omitted, or if the expression evaluates to null, the default value from the dictionary is used.                          | Boolean | No   |
| Retransmitted | Sets the t-bit in the outgoing Diameter message header. If omitted, or if the expression evaluates to null, a default of false is used.   | Boolean | No   |
| End-to-End    | Sets the end-to-end identifier in the outgoing Diameter message header. If omitted, or if the expression evaluates to null, an internally generated locally unique integer is used. | Integer | Yes for "-Answer" and "-Answer-Error" actions<br>No for "-Request" actions |
| Hop-by-Hop    | Sets the hop-by-hop identifier in the outgoing Diameter message header. If omitted, or if the expression evaluates to null, an internally generated locally unique integer is used. | Integer | Yes for "-Answer" and "-Answer-Error" actions                              |

| Argument            | Description  | Type    | Required (Yes/No)  |
|---------------------|--|---------|--|
|                     |  |         | No for “-Request” actions  |
| Application-Id      | Sets the application identifier in the outgoing Diameter message header. If omitted, or if the expression evaluates to null, the application id from the dictionary is used.   | Integer | Yes for “-Answer” and “-Answer-Error” actions<br>No for “-Request” actions |
| PeerIndex           | Specifies the index of the peer to which to send the message. The value is obtained from the request PeerIndex field. When the argument is present, and the expression evaluates to null, the message is not sent.   | Integer | Yes for “-Answer” and “-Answer-Error” actions<br>No for “-Request” actions |
| Request-Origin-Host | Sets the origin host of the request for an outgoing answer. The value of this argument should be the origin host of the peer that sent the request for this answer. If present, the value of the expression is used along with the end-to-end ID when saving the answer in the pending answer queue. When absent, the answer is not saved in the pending answer queue. | String  | No   |
| Queued-For-Delivery | Serves as a return code for the action. When present, the expression is set to true if the message is added to a peer's outgoing queue for delivery, and set to false otherwise. A value of true does not necessarily mean the message was sent to the peer, only that the message was added to a peer's queue.  | Boolean | No   |

Dynamic arguments are:

```
'<AvpName1>{ [Index1] } { .<AvpName2>{ [Index2] ... }'
```

Where:

- AvpName is the name of an attribute-value pair (AVP) from the dictionary for example, “Result-Code”
- Index is a numeric string. When Index is absent, an instance number of 0 is assumed.

The dynamic arguments are valid if all required arguments are present. Duplicate arguments cause policy reload to fail. If an AVP is required in a command, but the minimum number of instances is not present, policy will fail to reload. Optional dynamic arguments have a minimum instance count of 0 in the dictionary. For arguments that allow multiple instances, the index within the square brackets specifies the instance number. When more than one instance is used with an action, they must be used in the proper order starting with instance 0. So the user must specify instance 0 first, then 1, then 2 and so on. If multiple instances of an argument are not specified in the correct, order policy will fail to reload. The type of the argument value must be assignment-compatible with the type of the argument.

If an AVP name starts with anything other than a lower or upper case letter, or if the name contains anything other than alphanumeric characters or underscores, then the full argument name needs to be surrounded by single quotes. For example:

```
'Session-Id', 'Multiple-Services-Credit-Control{1}.Requested-Service-Unit'
```



#### Example:

This policy responds to a request with a specific session ID:

```
if expr(Diameter.Gy.'CC-Request'.Session-Id = "mysessionid") then \
  Diameter.Gy.'CC-Answer' \
    (PeerIndex: Diameter.Gy.'CC-Request'.PeerIndex, \
     'End-to-End': Diameter.Gy.'CC-Request'.End-to-End', \
     'Hop-by-Hop': Diameter.Gy.'CC-Request'.Hop-by-Hop', \
     'Request-Origin-Host': Diameter.Gy.'CC-Request'.Origin-Host', \
     'Result-Code': 2001, \
     'Origin-Host': PolicyEngine.Name, \
     'Origin-Realm': "sandvine.com", \
     'Auth-Application-Id': Diameter.Gy.'CC-Request'.Auth-Application-Id', \
     'CC-Request-Type': Diameter.Gy.'CC-Request'.CC-Request-Type', \
     'CC-Request-Number': Diameter.Gy.'CC-Request'.CC-Request-Number', \
     'Session-Id': Diameter.Gy.'CC-Request'.Session-Id', \
     'Application-Id': Diameter.Gy.'CC-Request'.Application-Id')
```



## 2.4.5 Event

Events allow one application to raise an event that other applications can listen for and take action on.

The event must be defined in policy before it can be used as an action.

```
Event.<Name>(<ArgName>:<arg_expression>, <ArgName>:<arg_expression>, ...)
```

Where:

|                               |   |
|-------------------------------|---|
| <b>&lt;Name&gt;</b>           | is the name of the event as provided in the definition.   |
| <b>&lt;ArgName&gt;</b>        | is the name of an event argument from the definition.   |
| <b>&lt;arg_expression&gt;</b> | is assigned to the corresponding ArgName argument when the event is raised. The expression must be assignment compatible with the type of the argument. |



### Example:

This example raises an event on every new flow. When the event is run, a row is inserted in the table. If the flow was an HTTP flow, the string HTTP is stored in the table, otherwise NULL is stored in the table. The remainder of the policy evaluation happens before the event is executed, but there is no guarantee on how soon after the policy evaluation finishes that the event will be run.

```
table "ExampleTable" (integer "key", string "value") timeout none \
unique by (Table.ExampleTable.Key) \
event "ExampleEvent" (integer "IntegerArg", string "StringArg")

PolicyGroup expr(Flow.IsNew){
  if expr(Flow.SessionProtocol = Protocol.Http) then \
    Event.ExampleEvent(IntegerArg: Flow, StringArg: "HTTP")
  if true then Event.ExampleEvent(IntegerArg: Flow)
}

if expr(Event.ExampleEvent) then \
  set ExampleTable[Event.ExampleEvent.IntegerArg].Value = \
  Event.ExampleEvent.StringArg
if expr(Flow.IsEnd) then delete_row Table.ExampleTable[Flow]
```

## 2.4.6 Increment

The increment action increments the value of the specified SandScript field.

Increment is only compatible with writeable integer fields. When incrementing a histogram measurement, the histogram bin is specified using the "bin" syntax. By default, the value of the field is incremented by 1. The optional "by" syntax can be used to increment the value by some other value. When it is used to increment the value of a subscriber attribute, the attribute expiry time can be set using the "for" syntax.

```
increment <field> {bin <bin_expression>}{by <expression>}{for <time>}
```

Where:

|                       |   |
|-----------------------|---|
| <b>field</b>          | Any writeable integer field.  |
| <b>bin_expression</b> | An integer expression that defines the histogram bin to increment. Can only be used if <code>field</code> is a histogram measurement.   |
| <b>expression</b>     | An integer expression that defines the value to increment by. If the expression evaluates to null, the <code>field</code> will not be changed (it is not set to null).                                      |
| <b>time</b>           | An integer expression plus a unit of time, for example, "10 minutes" or "LocalTime(2010,01,01) - Now seconds". Specifies when a subscriber attribute incremented by the action will expire and become null. |

## 2.4.7 Record Decrement

Decrements the values of multiple members in a record local variable. Supported for integer members only.

```
<RecordVariable>.Decrement(<MemeberName1>:<expression>, <MemeberName2>:<expression>, ...)
```



### Example:

For example:

```
record "ExampleRecord" (integer "intMember" default 5, ipaddress "addressMember")
Record.ExampleRecord "RecordVariable"
```

```
# Decrease the integer member of the variable by 7.
if true then RecordVariable.Decrement(intMember: 7)
```

## 2.4.8 Record Increment

Increments the values of multiple members in a record local variable. Supported for integer members only.

```
<RecordVariable>.Increment(<MemeberName1>:<expression>, <MemeberName2>:<expression>, ...)
```



### Example:

For example:

```
record "ExampleRecord" (integer "intMember" default 5, ipaddress "addressMember")
Record.ExampleRecord "RecordVariable"
```

```
#Increase the integer member of the variable by 5.
if true then RecordVariable.Increment(intMember: 5)
```

## 2.4.9 Record Reset

Sets the values of all members in a record local variable to their default values.

```
<RecordVariable>.Reset()
```



### Example:

```
record "ExampleRecord" (integer "intMember" default 5, ipaddress "addressMember")
Record.ExampleRecord "RecordVariable"
```

```
#Set the integer member of the variable to 5 (the default) and the address member to null
if true then RecordVariable.Reset()
```

## 2.4.10 Record Set

Sets the values of multiple members in a record local variable. The order in which members are set is not guaranteed. If order is important, multiple set actions should be used instead.

```
<RecordVariable>.Set((<MemeberName1>:<expression>, <MemeberName2>:<expression>, ...)
```



### Example:

For example:

```
record "ExampleRecord" (integer "intMember" default 5, ipaddress "addressMember")
Record.ExampleRecord "RecordVariable"
```

```
# Set both members of the variable.  
if true then RecordVariable.Set(addressMember: IPAddress("192.168.2.1"), intMember: 10)
```

## 2.4.11 Set

Use the set action to set SandScript fields that support modification.

```
set <fieldname> = <expression> {for <timeout>}
```

Where timeout is an integer expression plus a unit of time, for example, "10 minutes" or "LocalTime(2010,01,01) - Now seconds". Specifies when the expression will expire (and become null).



**Example:**

For example:

```
set flow.client.attribute.x = 1  
set flow.server.attribute.some_attr = NULL
```





# 3

## SandScript Definitions

- ["Local Variables" on page 30](#)
- ["Classifiers" on page 31](#)
- ["Timers" on page 32](#)
- ["Table Syntax" on page 34](#)
- ["User-defined Functions" on page 37](#)
- ["Foreach" on page 38](#)
- ["Events" on page 39](#)
- ["Published Expressions" on page 40](#)

## 3.1 Local Variables

Local variables are used to store the values of frequently used complex expressions (a complex expression is any expression that uses one or more operators or functions).

By storing the result of such an expression, the value can subsequently be retrieved quickly using the local variable instead of evaluating the more expensive expression multiple times. Local variables can be initialized to a value when they are defined and can be set/reset any number of times.

Local variables must be defined within a PolicyGroup. Typically, variables only have scope within the containing PolicyGroup, unless they are defined as shared. The scope of shared variables extends until the end of policy. Shared local variables defined in a PolicyGroup with a context different from that in which policy is being run are available but initialized to null.

Local variables never retain their value across policy evaluations. The value of a local variable is always reset to its initial value, or null if one is not specified, when it comes into scope. Local variables are defined as:

```
{shared} <type> "<Name>" {= <initial_value>}
```

Where:

|                      |  |
|----------------------|--|
| <b>shared</b>        | Optional. Extends the variable's scope until the end of policy.  |
| <b>type</b>          | Type of the local variable. Supported types include: string, float, integer, boolean, ipaddress, Table.TableName:cursor (a table row), Session:cursor (an IP indexed session), Record.<name>.  |
| <b>Name</b>          | Name of the local variable. Note that the name: <ul style="list-style-type: none"> <li>• Must begin with a letter and can only contain letters, digits, and underscores.</li> <li>• Must not be a reserved SandScript keyword.</li> <li>• Is not case sensitive.</li> <li>• Cannot conflict with other local variables that are still in scope.</li> </ul> <p>The name is used to lookup or set the value of the local variable. In the case of a cursor local variable, the name is used to:</p> <ul style="list-style-type: none"> <li>• Obtain column values of a particular table row (for example, VarName.ColumnName).</li> <li>• Access SandScript fields of a particular IP indexed session (for example VarName.IsMapped).</li> </ul> |
| <b>initial_value</b> | A complex expression which is assigned to the local variable. The expression must be assignment compatible with the type of the local variable. If a local variable is not assigned an initial value, it will remain null until it is assigned a value using the set action.   |



### Example:

For example:

```
PolicyGroup all {
    integer "var1"
    string "var2" = "Value"
    boolean "var3" = expression
    shared integer "sharedVar" = 180
    if condition(s) then set var1 = expression
    if condition(s) then set expression = var2
    if expr(var3) then action(s)
}
table "MyTable" (integer "key", integer "value") timeout none \
    unique by (Table.MyTable.Key)
PolicyGroup all {
    Table.MyTable:cursor "row1"
    Table.MyTable:cursor "row2" = Table.MyTable[2]
    Table.MyTable:cursor "row3" = Table.MyTable.Create(2)
    if condition(s) then row2.Set(value: 7)
```

```
if condition(s) then set row3.value = 180
if condition(s) then set row1 = row2
if expr(row2 = row3) then action(s)
if expr(row1.value > 10) then actions(s)
}
```

## 3.2 Classifiers

Classifiers are used to apply labels to policy contexts such as flows or subscribers and to categorize flows or subscribers into meaningful groups.

The set action is used to set the classifier's value through its SandScript field. The SandScript field can be set any number of times in policy and can be used by the classifier to: apply a condition, measure, limit, or shape unique-by. The Flow.Classifier.Name SandScript field can only be used in rules and classifications that apply to flows. Use Classifier.Name when using the classifier on subscribers.

Classifiers are defined either by type or listed explicitly. The syntax of classifier instances specified by type is:

```
classifier "<Name>" type
string|integer|integer8|integer16|integer32|unsigned8|unsigned16|unsigned32
```

If defined this way, the classifier can be set to any SandScript field of the appropriate type. You can limit the range of input values by using one of the smaller types. If you try to assign a value larger than the type to the classifier, the value is truncated to fit within the classifier.

The syntax of classifier instances listed explicitly is:

```
classifier "<Name>" values "<value1>" {"<value2>" ...}
```

Where:

- Name is the name of the classifier used to identify the classifier in the CLI and when referencing the classifier using SandScript fields.
- "value1" {"value2" ...} is a list of classifier values. These values are used as the names of the constant SandScript fields that can be used to set the value of the classifier (Classifier.Name.Value).

Classifier names and values must follow these rules:

- Must begin with a letter and can only contain letters, digits, and underscores.
- Not case sensitive.
- Names must be unique amongst names and values must be unique amongst values for the given classifier (different classifiers can have the same values).

Each classifier begins each policy evaluation without a value (the Classifier.Name SandScript field is null). A classification does NOT persist across policy evaluations. The value is reassigned each time policy is evaluated. If none of the classification conditions are matched, the classifier will remain without a value (the SandScript field will remain null).

The maximum number of subscriber classifiers that can be used is 76. Of that number, 64 can be a sum and 12 can be used to report the maximum value over the past 12 hours.



### Example:

This example classifies traffic into protocol categories, measures and publishes the number of bytes transferred per category per subscriber.

```
classifier "Category" values "Web" "P2P" "Other"
PolicyGroup {
  if expr(Flow.SessionProtocol = Protocol.Http) then \
    set Flow.Classifier.Category = Classifier.Category.Web
  if expr(Flow.IsApplicationProtocol("ares","bittorrent","edonkey","gnutella")) then \
```

```

    set Flow.Classifier.Category = Classifier.Category.P2P
  if true then \
    set Flow.Classifier.Category = Classifier.Category.Other
  }

measurement "BytesPerCategoryPerSub" sum(Flow.Subscriber.SubscriberBytes) \
  over publish_interval unique by (Subscriber, Flow.Classifier.Category)
publish "BytesPerCategoryPerSub" Measurement.BytesPerCategoryPerSub

```

**Example:**

This example measures and publishes the number of bytes transmitted and received per protocol for each CMTS and QAM (CMTS and QAM are subscriber attributes):

```

attribute "CMTS" type string
attribute "QAM" type string
classifier "CMTS" type string
classifier "QAM" type string
if true then \
  set Flow.Classifier.CMTS = Flow.Subscriber.Attribute.CMTS and \
  set Flow.Classifier.QAM = Flow.Subscriber.Attribute.QAM

measurement "DownstreamBytesPerProtocolPerCmtsQam" \
  sum(Flow.Subscriber.Rx.ProtocolBytes) \
  over publish_interval \
  unique by (Flow.ApplicationProtocol.StatsProtocol,
    Flow.Classifier.CMTS,
    Flow.Classifier.QAM)
publish "DownstreamBytesPerProtocolPerCmtsQam" \
  Measurement.DownstreamBytesPerProtocolPerCmtsQam
measurement "UpstreamBytesPerProtocolPerCmtsQam" \
  sum(Flow.Subscriber.Tx.ProtocolBytes) \
  over publish_interval \
  unique by (Flow.ApplicationProtocol.StatsProtocol,
    Flow.Classifier.CMTS,
    Flow.Classifier.QAM)
publish "UpstreamBytesPerProtocolPerCmtsQam" \
  Measurement.UpstreamBytesPerProtocolPerCmtsQam

```

## 3.3 Timers

Timers are used to generate an event at some point in the future. When the event occurs, SandScript rules available in the timer's context are evaluated.

There are two types of timers: repeating and non-repeating. Repeating timers will be immediately rearmed every time they are fired. Every time a repeating timer is armed, it will use the same duration. The duration of repeating timers cannot be changed. Non-repeating timers will only fire once, unless they are rearmed by the timer arm action.

Information about timers can be viewed by using the `show policy timers` CLI commands. Timers can be explicitly armed and cleared in SandScript using timer arm and timer clear actions.

```

timer "<Name>" {arm <timer_interval> {repeat true|false}} {tolerance <maximum_lag>|infinite}
{resolution <accuracy>}

```

Where:

|             |   |
|-------------|---|
| <b>Name</b> | Name of the timer, used to identify the timer in the CLI and when referencing the timer using SandScript expressions. Note that it: <ul style="list-style-type: none"> <li>• Must begin with a letter and can only contain letters, digits, and underscores.</li> <li>• Is not case sensitive.</li> <li>• Must not be a reserved SandScript keyword.</li> </ul> |
|-------------|---|



- Must be unique amongst timers.
- Determines the timer type (`Timer.Name`) that can be used as column types of a SandScript table.

**arm <timer\_interval>  
{repeat {true|false}}**

Sets the interval that the timer should be armed for as soon as it is created. If omitted, the timer will be disarmed by default and will not be repeating.

<timer\_interval> is an integer followed by units of time, for example, 5 seconds, 500 milliseconds.

The optional repeat clause specifies whether or not the timer is repeating. If omitted, the default value is false.

**tolerance  
{<maximum\_lag>|infinite}**

Indicates the maximum amount of time that a timer can fire late before it is considered to be an error. If this tolerance is exceeded, an alarm is raised.

<maximum\_lag> is an integer followed by units of time, for example, 5 days, 5 hours, 5 minutes, 5 seconds, or 500 milliseconds. The default value is infinite, which indicates that it is never considered an error regardless of how late the timer fires.

**resolution <accuracy>**

The accuracy requirement of the timer. A timer with a duration of 10 seconds will be expected to fire no later than (10 seconds + accuracy) after it is armed. For example, a timer with a duration of 10 seconds and a resolution of 10 milliseconds will be expected to fire 10 to 10.01 seconds after it is armed. <accuracy> is an integer followed by units of time, for example, 5 seconds, 500 milliseconds. The default value is 100 milliseconds.



**Example:**

For example:

```
# Define a timer that fires every minute with one second resolution
timer "MinuteTimer" arm 1 minutes repeat true resolution 1 seconds
PolicyGroup expr(Timer.MinuteTimer.Expired) all
{
  # Rules to evaluate every minute when the timer expires
}
```

## 3.3.1 Timer Arm

Use the timer arm action to arm a policy timer for a particular duration.

```
timer.<TimerName>.arm({duration:<time>})
```

Where:

**<TimerName>**

is the name of the timer

**<time>**

indicates how long to arm the non-repeating timer for. If the duration argument is specified, arms non-repeating timer for the duration time. If the duration argument is omitted, the timer is reset to its initial duration. A negative duration is treated the same as a duration of zero. (expires now).

## 3.3.2 Timer Clear

Use the timer clear action to disarm a policy timer. This action may be used on both repeating and non-repeating timers. After a timer has been disarmed, it may be restarted using its arm action.

```
timer.<TimerName>.clear()
```

## 3.4 Table Syntax

Tables are used to store state and build state machines in policy.

Tables in policy are similar to tables in a database. Tables have a name and a set of named columns. Each table must define at least one column. Default values can be assigned to columns. Rows are created when a column value is set in policy using a set action, or an increment action, or the Create function is used. Rows are deleted either explicitly using the `delete_row` action or when they time out if configured to do so.



### Warning:

Any change to the table schema deletes all rows from the table upon reload.

When the definition of a table in policy changes, for example because new columns are added, data types are changed, or you change the generation number, any existing data in the table is cleared when these changes are loaded. If you do not wish to lose the data, instead of deleting a column, just ignore it. Or, if you wish to add a column, try adding new tables to achieve the functionality that is desired without changing existing tables.

By default, a maximum of 1000 tables can be defined in policy.

There is a maximum amount of memory that can be used by table rows and a limit on the number of rows that can be created. An alarm is raised if either limit is exceeded, and attempts to create new rows will fail until the active usage falls below the maximum.

You can view information about tables by executing the CLI command `show policy tables`.

```
table "<Name>" (<type> "<ColumnName1>" {default <expression>}{sparse true|false},
  <type> "<ColumnName2>" {default <expression>}{sparse true|false}, ...)
timeout <interval> {reset timeout read|write|readwrite|none }{generation <number>}
unique by <unique_by_spec>
```

|                                   |  |
|-----------------------------------|--|
| <b>&lt;Name&gt;</b>               | <p>Name of the table, used to identify the table in the CLI and when referencing the table using SandScript.</p> <ul style="list-style-type: none"> <li>Must begin with a letter and can only contain letters, digits, and underscores.</li> <li>Is not case sensitive.</li> <li>Must be unique amongst tables.</li> <li>Determines the name of the SandScript fields (<code>Table.Name</code>) that are used to access rows in the table.</li> </ul>  |
| <b>&lt;type&gt;</b>               | <p>Type of the column. Supported types include: <code>string</code>, <code>string(N)</code> (string of length at most <code>N</code>), <code>float</code>, <code>integer</code> (use smaller integer types to use less memory: <code>integer8</code>, <code>integer16</code>, <code>integer32</code>, <code>unsigned8</code>, <code>unsigned16</code>, <code>unsigned32</code>), <code>timestamp</code>, <code>ipaddress</code>, <code>boolean</code>, and <code>Timer.TimerName</code>.</p> |
| <b>&lt;ColumnName&gt;</b>         | <p>Name of a column. Multiple columns can be specified. Column specifications (which include the name, the type, and column options) are separated by commas. The same rules that apply to the table name apply to the column names. The column names (along with the table name) determine the name of the SandScript fields that are used to read and write to the table (for example, <code>Table.Name.ColumnName</code>).</p>  |
| <b>default &lt;expression&gt;</b> | <p>The default value of the column when a row is created. Optional, if omitted, null is assumed. The expression must be assignment compatible with the type of the column.</p>   |
| <b>sparse {true false}</b>        | <p>Specifies if the table will optimize the memory used by the column under the assumption that the column usually holds a null value. This option should only be set to true for columns that are null most of the time or for timers that are unarmed most of the time. This option cannot be used for key columns or timer columns that are armed by default. Sparse is optional; if omitted the column is not sparse.</p>  |

|  |   |
|--|---|
| <b>timeout &lt;interval&gt;</b>                    | <p>Specifies how long after a row is created should it time out, after which it is deleted. &lt;interval&gt; is an integer followed by units of time. Examples are:</p> <pre>5 seconds 5 sec 500 milliseconds 500 ms</pre> <p>Use “none” to indicate that rows should never time out. Specifying a 0-length time interval has the same effect as using “none”. If using none, rows must be deleted using the delete_row action. Take care to ensure rows are deleted in a sufficient time such that resources are not exhausted.</p>                              |
| <b>reset_timeout<br/>read write readwrite none</b> | <p>Specifies when the timeout time of a row is reset. If omitted, the default value is readwrite.</p> <ul style="list-style-type: none"><li>• Read causes the timer to reset each time a value is read from the row.</li><li>• Write causes the timer to reset each time a value is written to the row.</li><li>• Readwrite causes the timer to reset each time a value is read from or written to the table.</li><li>• None causes the timer to never reset, the timer will start when the row is created and the row will be removed when it expires.</li></ul> |
| <b>generation &lt;number&gt;</b>                   | <p>A non-negative integer that specifies if a table should be cleared on policy reload. If the generation number changes, the table will be cleared upon reload.</p>  |
| <b>&lt;unique_by_spec&gt;</b>                      | <p>The key for the table, a separate row is created for each instance defined by the specification. It is defined as a SandScript expression list (a comma separated list of fields or expressions inside parenthesis), for example, “unique by (Flow)”. Note that columns of the table may be used in the unique by specification, accessed via their SandScript field, essentially creating a primary key for the table, for example, “unique by (Table.Name.ColumnName)”.</p>  |



**Example:**

For example:

```
# define the table
# this table can store a string per Flow
table "FlowState" (string "HttpHostname") timeout none unique by (Flow)
PolicyGroup expr(Flow.SessionProtocol = Protocol.Http) {
  # when HTTP flows end, delete their row from the table
  if expr(Flow.IsEnd) then delete_row Table.FlowState
  # for every HTTP flow, create a row and store the hostname
  if expr(Flow.Client.Stream.Http.Host is not null) then \
    set Table.FlowState.HttpHostname = \
      Flow.Client.Stream.Http.Host
}
```

## 3.4.1 Accessing Table Rows

Table rows can be accessed in one of two ways, either implicitly (the expressions in the unique\_by\_spec specify a particular row when evaluated) or explicitly using square bracket “[ ]” accessors.

For implicit access, tables are accessed via these SandScript fields:

Table.Name.ColumnName

For explicit access, tables are accessed via these SandScript fields:

Table.Name[<Key>].ColumnName

Note that if columns of the table are used in the `unique_by_spec`, only explicit access is supported. This is not strictly true if the table has a timer column, which defines a new timer context in which a row is accessed implicitly. For example:

```
timer "MyTimer"
table "MyTable" (integer "IntCol", \
                  string "StringCol", \
                  Timer.MyTimer "TimerCol") \
    timeout none unique by (Table.MyTable.IntCol)

PolicyGroup expr(Table.MyTable.TimerCol.Expired) {
    # Table.MyTable.IntCol and Table.MyTable.StringCol accessed implicitly in this timer expired
    context
}
```

Use the `delete_row` action with the `Table.Name` or `Table.Name[<Key>]` fields to delete rows from a table.

### 3.4.2 Table Clear

Use the table clear action to delete all rows in a SandScript table.

```
table.<TableName>.clear()
```

### 3.4.3 Table Decrement

Use the table decrement action to efficiently decrement multiple integer columns of a row in a SandScript table.

An attempt to increment a read-only column results in an error. Supported for integer columns only. If the row for that key is not in the table, a new row is created.

```
table.<TableName>[<Key>].decrement(ColumnName1:<expression>, ColumnName2:<expression>, ...)
```

Where `<expression>` is the amount by which to decrement the column.

### 3.4.4 Table Increment

Use the table increment action to efficiently increment multiple integer columns of a row in a SandScript table.

An attempt to increment a read-only column results in an error. Supported for integer columns only. If the row for that key is not in the table, a new row is created.

```
table.<TableName>[<Key>].increment(<ColumnName1>:<expression>, <ColumnName2>:<expression>, ...)
```

Where `<expression>` is the amount by which to increment the column.

### 3.4.5 Table Persistence

A policy table can be persisted using local persistency, where table rows are saved to the local disk, or replicated persistency, where table rows are saved to both local and remote disk.

Use replicated persistency in a high-availability (HA) cluster. Using persistency or replication on tables has a significant impact on table action performance, as it requires disk I/O and network communication.

A persisted or replicated table has these limitations:

- Table row timeout is not replicated, but is re-armed to failover.
- Timer columns cannot be defined as repeatable timers.

The syntax for local persistency is:

```
Table "Name" { \
  ... \
} persist true unique by ...
```

The syntax for replicated persistency is:

```
Table "Name" { \
  ... \
} persist true replicate true unique by ...
```

## 3.4.6 Table Reset

Use the table reset action to efficiently reset all writable columns of a row in a policy table to their default values.

```
table.<TableName>[<Key>].reset()
```

## 3.4.7 Table Set

Use the table set action to efficiently set multiple columns of a row in a SandScript table.

An attempt to set a read-only column results in an error. The order in which columns are set is not guaranteed. If order is important, multiple set actions should be used instead. If the row for that key is not in the table, a new row is created.

```
table.<TableName>[<Key>].set(<ColumnName1>:<expression>, <ColumnName2>:<expression>, ...)
```

Where <expression> is the value to set for the column.

## 3.4.8 Table Timer Arm

Use the table timer arm action to arm a timer column of a row in a policy table. The action only applies if <ColumnName> is a timer column, and the timer is only armed for the indexed row. Refer to the Timer arm action.

```
table.<TableName>[<Key>].<ColumnName>.arm({duration:time})
```

## 3.4.9 Table Timer Clear

Use the table timer clear action to disarm a timer column of a row in a policy table. The action only applies if <ColumnName> is a timer column, and the timer is only disarmed for the indexed row. Refer to the Timer clear action.

```
table.<TableName>[<Key>].<ColumnName>.clear()
```

# 3.5 User-defined Functions

Functions can be defined to group together commonly-executed rules, perhaps parameterized by a set of argument values.

Effective use of functions can assist with writing cleaner, more modular, and more maintainable SandScript policy. Functions can be invoked after they are defined, as either expressions or actions. A function cannot call itself or a function that is defined after it. If a function does not define a return value, then it returns null when used as an expression. The function body, however, is still executed.

```
function "<name>" ({writable} <type> "<ArgName1>" {default <expression>},
                  {writable} <type> "<ArgName1>" {default <expression>},...)
{
    rules and/or nested SandScript groups...
}
```

|                              |   |
|------------------------------|---|
| <b>&lt;name&gt;</b>          | Name used to invoke the function. Note that name: <ul style="list-style-type: none"> <li>• Must begin with a letter and can only contain letters, digits, and underscores.</li> <li>• Must not be a SandScript reserved keyword.</li> <li>• Is not case-sensitive.</li> <li>• Cannot conflict with other function names (user-defined and internal).</li> </ul> |
| <b>writable</b>              | Allows the function to modify the expression assigned to the argument when the function is invoked. The writable keyword may not be used if the default keyword is used.  |
| <b>default</b>               | Specifies a default value to use for the argument if the argument is not specified when the function is invoked. The default keyword may not be used if the writable keyword is used.   |
| <b>&lt;type&gt;</b>          | Type of argument or return value. Supported types include all those supported for local variables.  |
| <b>&lt;returnValName&gt;</b> | The return value name which can be set from within the function body. If the field is not set within the function body, the return value will be null.  |



#### Example:

This example invokes the user-defined function FuncName:

```
if <condition(s)> then FuncName(ArgName:<arg_expression>, ArgName:<arg_expression>, ...)
if expr(FuncName(ArgName:<arg_expression>, ArgName:<arg_expression>, ...)) then action(s)
if <condition(s)> then set field = FuncName(ArgName:<arg_expression>,
ArgName:<arg_expression>, ...)
```

Where:

- FuncName is the name of the function as provided in the definition.
- ArgName is the name of a function argument from the definition.
- <arg\_expression> is the expression to be assigned to the corresponding ArgName argument when the function body is executed. The expression must be assignment compatible with the type of the argument. If the corresponding ArgName argument is writable, then the expression must be writable and of the same type as the argument.



#### Example:

A simple summation example:

```
function "add" (integer "LeftHandSide", integer "RightHandSide") returns integer "sum" {
    if true then set sum = LeftHandSide + RightHandSide
}

integer "totalBytes"
if expr(Flow.IsEnd) then set totalBytes = add(Flow.Rx.TotalBytes, Flow.Tx.TotalBytes)
```

## 3.6 Foreach

Foreach loops can be used to efficiently evaluate rules for every row in a policy table.

A foreach loop is useful when it is necessary to extract information from or update information in many rows of a policy table, as it is more efficient than accessing rows individually. The foreach policy construct can be used with Diameter policy.

```
foreach "<IteratorName>" in Table.<TableName>{  
    rules and/or nested policy groups...  
}
```

**<IteratorName>** The name of the foreach iterator, which is of type Table.TableName:cursor. The iterator has scope only within the foreach construct and cannot be explicitly re-assigned.

- Must begin with a letter and can only contain letters, digits and underscores.
- Must not be a policy reserved keyword.
- Is not case sensitive.
- Cannot conflict with other local variables that are still in scope.

**<TableName>** The name of the policy table being accessed.



**Example:**

For example:

```
table "MyTable" (integer "key", integer "value") timeout none \  
    unique by (Table.MyTable.Key)  
foreach "row" in Table.MyTable  
{  
    # For all rows with even-numbered keys, increment the value  
    if expr((row.key % 2) = 0) then increment row.value  
}
```

## 3.7 Events

Events can be defined in policy to allow one application to raise an event that other applications can listen for and take action on.

When policy raises an event, all of the event's arguments are evaluated, and those values are available to any rules using fields of the event. All event arguments are optional. If they are not provided when the event is raised, event listeners observe a null value for those arguments.

There is no guarantee on when an event will be run, or on the relative ordering of events. There may be any number of rules using the event, or none. The `Event.Name` SandScript field is only evaluated when the event has been raised. Actions within an event listener cannot raise the same event that is being listened for, or an event that will cause the same event to be raised. Events cannot be raised in rules that are available in all policy scopes.

Events may be redefined as long as the types and names of all event arguments are the same in each definition. The order of the event argument definitions does not matter.

```
event "<Name>" (<type> "<ArgName1>", <type> "<ArgName2>", ...)
```

Where:

- **<Name>** is the name of the event. Note that **<Name>**:
  - Must begin with a letter and can only contain letters, digits, and underscores.
  - Must not be a policy reserved keyword.
  - Is not case sensitive.
- **<type>** is the type of the event argument. Supported types include: string, float, integer, boolean, ipaddress and `Record.RecordName`.

Events are raised as actions as:

```
if <condition(s)> then Event.<Name>(ArgName:<arg_expression>, ArgName:<arg_expression>, ...)
```

Where:

|                               |   |
|-------------------------------|---|
| <b>&lt;Name&gt;</b>           | The name of the event as provided in the definition.  |
| <b>&lt;ArgName&gt;</b>        | The name of an event argument from the definition   |
| <b>&lt;arg_expression&gt;</b> | Is assigned to the corresponding ArgName argument when the event is raised. The expression must be assignment compatible with the type of the argument. |

Policy can listen for an event. For example:

```
if expr(Event.Name) then action(s)
```

Where: <Name> is the name of the event as provided in the definition.



#### Example:

This example raises an event on every new flow. When the event is run, a row is inserted in the table. If the flow was an HTTP flow, the string HTTP is stored in the table, otherwise, NULL is stored in the table. The remainder of the policy evaluation happens before the event is executed, but there is no guarantee on how soon after the policy evaluation finishes that the event will be run.

```
table "ExampleTable" (integer "key", string "value") timeout none unique by
(Table.ExampleTable.Key)
event "ExampleEvent" (integer "IntegerArg", string "StringArg")

PolicyGroup expr(Flow.IsNew) {
  if expr(Flow.SessionProtocol = Protocol.Http) then Event.ExampleEvent(IntegerArg: Flow,
StringArg: "HTTP")
  if true then Event.ExampleEvent(IntegerArg: Flow)
}

if expr(Event.ExampleEvent) then set ExampleTable[Event.ExampleEvent.IntegerArg].Value =
Event.ExampleEvent.StringArg

if expr(Flow.IsEnd) then delete_row Table.ExampleTable[Flow]
```

## 3.8 Published Expressions

Published expressions facilitate custom reporting and provide a mechanism for reporting on the value of a SandScript expression over time.

The value of the SandScript expression is periodically sampled and stored in the database. These values can then be reported on via Network Demographics Server reports. To examine published expressions, create a Published Expressions report (in Network Demographics, choose **Network Characterization > Demographics > by Network Element > Published Expressions**).

The sampling period defaults to 15 minutes (900 seconds), but can be configured using the `set config service statistics log-interval published-expression <value>` CLI configuration command. Note that the sampling period is a global variable which affects all published expressions.

The SandScript expression being published must be a scalar, integer-valued expression. A scalar expression is one whose value is not tied to a particular flow, packet, or subscriber.

```
publish "<name>" <expression> {aggregate sum|max}
```

|                           |   |
|---------------------------|---|
| <b>&lt;name&gt;</b>       | The name assigned to the expression. The name can be any alphanumeric string, unique among publish statements. This name appears in Network Demographics reports. |
| <b>&lt;expression&gt;</b> | Any scalar, integer-valued expression.  |



**aggregate {sum|max}** The aggregation type of the published expression. This option must be specified if the expression being published does not have an aggregation type. Non-generic measurements have an implied aggregation type and therefore do not need this option. The aggregation types are:

- **sum** - The published expression is treated as a counter when its values are aggregated across modules and over time intervals in reports.
- **max** - The published expression is treated as a gauge where the peak value of the published expression is taken, across modules and over time intervals in reports.

on-scalar measurements can be published if they have a particular unique by specification and are either synchronized to the publishing interval or over infinity. For a non-scalar measurement to be publishable, it must be unique by one of:

- Subscriber.
- Subscriber and classifier.
- 1-4 classifiers.

When publishing unique by subscriber or subscriber and classifier, the number of published instances for any given subscriber is limited by the capacity of the SPB. Counters are defined as measurements which are aggregated using the sum operator and gauges are defined as measurements which are aggregated using the max operator. The limits are:

| SPB Release      | Counter Limit | Gauge Limit |
|------------------|---------------|-------------|
| 6.20 and later   | 64            | 12          |
| 6.00 and earlier | 32            | 6           |





# 4

## Measurements

- ["Measurements Overview" on page 44](#)
- ["Measurement Groups" on page 49](#)
- ["Histograms" on page 49](#)

## 4.1 Measurements Overview

Measurements are declared through policy to collect traffic and subscriber-related statistics.

Measurements are taken in the order that they are defined in policy, however, they are taken after all other applicable actions have been run. This does not apply to generic measurements which are modified explicitly via the set/increment/decrement actions.

```
measurement "blockedConnections" connections where expr(Flow.IsBlocked)
if expr(Flow.ApplicationProtocol = Protocol.ares) then block
```

A maximum of 1000 measurements can be defined in SandScript. The measurement types that are supported are:

| Measurement Type       | Description  |
|------------------------|--|
| Generic (user-defined) | The value of this type of measurement is controlled explicitly through policy using the set, increment, or decrement actions. It can be used to implement any kind of user-defined counter or statistic. |
| Expression             | A measurement can be used to expose the value of a SandScript expression.  |
| Top                    | Sums the value of an expression over time and finds the top instances.   |
| Sum                    | Aggregates the value of a SandScript expression over time.   |
| Histogram              | Counts observed values into bins.  |

Measurement values are available using the CLI command `show policy measurements`, so you can monitor the values in real-time.

Measurement values are also available in policy via SandScript fields and can be published and reported on in NDS, allowing a number of custom user-defined statistics and reports to be defined. For detailed information on SandScript fields, refer to the *PTS SandScript Reference*.

### 4.1.1 Measurements

Collects traffic and subscriber related data.

```
measurement "<Name>" <type> {over <period>} {where <condition(s)>} {unique by <unique_by_spec>}
{id <index>}
```

#### Name

The name of the measurement, used to identify it through the CLI and SandScript. The name:

- Must begin with a letter.
- Can only contain letters, digits, and underscores.
- Is not case sensitive.
- Must be unique amongst measurements and limiters (cannot have a measurement named "Protocols" and a limiter named "Protocols").

#### type

Specifies the measurement type. If no type is specified, it is a generic measurement. Otherwise, measurements come in these types:

```
connections
bitrate
hosts
sum
```

top  
expr  
histogram

### Period

The exact behavior depends on the measurement type. In general:

- Peak values are reset at the start of each period.
- With the exception of connection measurements, the set of unique instances being measured by unique-by measurements is cleared at the start of each period.
- Can be specified as a time in seconds, minutes, or hours (for example, “over 60 seconds”, “over 5 minutes”, “over 1 hour”).
- For generic and sum measurements, if unique\_by\_spec is specified as between 1 and 3 enumerated classifiers, period may be specified as infinity. The measurement can be published, but its values will never be reset.
- Can be synchronized with the statistics publishing interval by specifying publish\_interval.
- Unique-by measurements MUST be synchronized to the publishing interval to be published.
- Required for measurements where a unique-by\_spec is specified, optional for all others. The default behavior is to never reset the value and/or peak values of the measurement.

### Condition(s)

The condition(s) that indicate what to measure. The format of the condition(s) is the same as it is for rules. The condition is optional; the default value is **true**, and all traffic or hosts (depending on the measurement type) are measured.

### unique\_by\_spec

When this measurement is specified, it measures a value for each instance as defined in the specification. These values are acceptable”:

client-ip|server-ip|internal-ip|receiver-ip|sender-ip|client-subscriber|server-subscriber|subscriber

For example, “unique by client-ip”.

Or, can be a SandScript expression list (a comma separated list of fields or expressions inside parenthesis). For example:

```
unique by (Flow.Subscriber.Attribute.CMTS, Flow.Subscriber.Attribute.QAM)
unique by (Subscriber, Flow.Classifier.APN)
```

The total number of unique instances being measured across all measurements is limited according to platform:

- PTS 24000 supports 63 million.
- PTS 22000 supports 20 million.
- PTS 14000 supports 20 million.

### pt

An alarm is raised when the limit is exceeded. A published measurement can be unique by no more than four unique-by classifiers, or no more than one protocol and three classifiers.

### index

A static identifier for the measurement that persists through reloads and restarts. Use this identifier to poll the measurement via SNMP in the SANDVINE-MIB::svMeasurementsTable.

## 4.1.2 Generic Measurement

User-defined measurements are referred to as generic measurements.

```
measurement "<Name>" {over <period>} {unique by <unique_by_spec>} {aggregate sum|max}
```

**aggregate {sum|max}** is the aggregation type of the measurement. The aggregation types are:

- **sum**, (the default) the measurement is treated as a counter when its values are aggregated across modules and over time intervals in reports.
- **max**, the measurement is treated as a gauge where the peak value of the published expression is taken, across modules and over time intervals in reports.

Unlike other measurement types, simply defining the measurement does not provide any value. Policy rule(s) must be written to set the `Measurement.Name`, or `Measurement.Name.Current` SandScript fields which are initialized to 0. Setting generic measurements to NULL is equivalent to setting them to 0.

The measurement may or may not be over an interval. If it is not over an interval, then you can set the value of the defined measurement using the `Measurement.Name` SandScript field. If the measurement is over an interval, then the values of the measurement that are exposed to policy are:

- `Measurement.Name` for the value of the measurement in the previous interval.
- `Measurement.Name.Current` for the value of the measurement in the current interval.



**Note:**

For measurements over an interval, Sandvine recommends modifying the `Measurement.Name.Current` SandScript field. While the policy that sets, increments, or decrements the `Measurement.Name` SandScript field for measurements over an interval is valid, it has the same effect as performing the action on the `Measurement.Name.Current` SandScript field.



**Example:**

To set the value of this measurement over an interval using the set, increment, or decrement actions, use the `Measurement.<Name>.Current` SandScript field. For example:

```
measurement "NewHTTPFlows" over publish_interval
if expr(Flow.IsNew and Flow.ApplicationProtocol = Protocol.HTTP) \
then increment Measurement.NewHTTPFlows.Current
```

## 4.1.3 Sum Measurement

This type of measurement is used to aggregate the value of an expression over time. Typically sum measurements are used to count bytes or packets.

Syntax:

```
measurement "<Name>" sum(<expression>) {over <period>} {where <condition(s)>}
{unique by <unique_by_spec>}
```

The `unique_by_spec` is optional, if specified, period must also be specified.

If no `unique_by_spec` is specified, period controls how often the measurement value is reset to 0. If a `unique_by_spec` is specified, period controls how often the instances are flushed. In both cases, the peak values are also reset at the start of each period.

This example counts the total number of bytes for all HTTP connections. The value of the measurement is reset at the start of each publishing interval.

```
measurement "HttpBytes" sum(Flow.Tx.ProtocolBytes) over publish_interval \
where expr(Flow.ApplicationProtocol = Protocol.HTTP)
```

This example measures the total number of bytes per subscriber and protocol category (defined by a classifier) and resets the statistics each publish interval:

```
classifier "Category" values "Web" "P2P" "Other"
PolicyGroup {
  if expr(Flow.SessionProtocol = Protocol.Http) then \
    set Flow.Classifier.Category = Classifier.Category.Web
```

```
if expr(Flow.IsApplicationProtocol("ares","bittorrent","edonkey","gnutella") then \  
  set Flow.Classifier.Category = Classifier.Category.P2P  
if true then \  
  set Flow.Classifier.Category = Classifier.Category.Other  
}  
measurement "BytesPerSubPerCategory" sum(Flow.Tx.ProtocolBytes) over publish_interval \  
  unique by (Subscriber, Flow.Classifier.Category)
```

## 4.1.4 Top Measurement

Top measurements can be taken using an existing unique-by generic or sum measurement, or by using a statistical method to estimate top instances.

To identify the top instances of an existing unique-by generic or sum measurement, the syntax is:

```
measurement "<Name>" top(<n>, <measurement_reference>)  
measurement "<Name>" topPercent(<n>, <measurement_reference>)
```

Where:

|                                      |   |
|--------------------------------------|---|
| <b>Name</b>                          | Name of the measurement.  |
| <b>top &lt;n&gt;</b>                 | The number of top instances to identify as an absolute number. Must be a value between 1 and 1000.  |
| <b>topPercent &lt;n&gt;</b>          | The number of top instances to identify, as a percentage of the total number of instances. Must be a value between 1 and 100. If the percentage multiplied by the number of instances in the measurement being referenced is greater than 1000, only the top 1000 instances are identified. |
| <b>&lt;measurement_reference&gt;</b> | A reference to the value of a unique-by generic, sum, or other top measurement (Measurement.Name). The top n instances of this measurement are identified. Note that unique by subscriber measurements cannot be referenced in top measurements.  |

If the number of unique instances defined by the unique\_by\_spec is too high to count with a generic or sum measurement, use this syntax:

```
measurement "<Name>" top(<n>, sum(<expression>)) over <period> {where <condition(s)>} unique  
by <unique_by_spec>
```

The syntax is similar to that of the sum measurement, with the addition of the top operator.

The statistical method used to estimate the top instances will introduce some error into the measured values. Each instance that is measured by the top measurement will be no less than 95% of its actual value. It is also possible that some instances near the tail of the list will incorrectly be reported as being one of the top instances. This uncertainty is relatively higher when the volume of traffic being measured is low.



### Example:

This example identifies the top 100 HTTP hostnames by connections:

```
measurement "TopHttp" top(100, sum(1)) over publish_interval \  
  unique by (Flow.Client.Stream.Http.Host)
```

This example demonstrates how to publish a top measurement and count the top URLs by bytes.

```
table "HttpFlowState" (string "Url") timeout none unique by (Flow)  
classifier "Url_Bytes" type string  
PolicyGroup expr(Flow.SessionProtocol=Protocol.Http) {  
  if expr(Flow.Client.Stream.Http.Host is not null) then \  
    set Classifier.Url_Bytes = Flow.Client.Stream.Http.Url and \  
    set Table.HttpFlowState.Url = Flow.Client.Stream.Http.Url  
  if expr(Table.HttpFlowState.Url is not null) then \  
    set Classifier.Url_Bytes = Table.HttpFlowState.Url  
}
```

```
measurement "TopBytesByUrl" \
  top(6, sum(Flow.Tx.ProtocolBytes)) \
  over publish_interval \
  where expr(Flow.SessionProtocol = Protocol.Http) \
  unique by (Classifier.Url_Bytes)
publish "TopBytesByUrl" Measurement.TopBytesByUrl
if expr(Flow.SessionProtocol=Protocol.Http and Flow.IsEnd) then delete_row
Table.HttpFlowState
```

## 4.1.5 Expression Measurement

Expression measurements measure the value of a SandScript expression

The syntax is:

```
measurement "<Name>" expr(<expression>) {over <period>}
```

Expression measurements can only be used with scalar fields. This type of measurement is most often used to perform some math on two or more other measurements. For example, two bitrate measurements that measure upstream and downstream bandwidth can be added together to measure total bandwidth:

```
measurement "Upstream" bitrate where source class "internal"
measurement "Downstream" bitrate where destination class "internal"
measurement "Total" expr(Measurement.Upstream + Measurement.Downstream)
```

In this example, average packet size is determined by dividing total bytes by total packets:

```
measurement "Bytes" sum(Flow.Tx.Bytes)
measurement "Packets" sum(Flow.Tx.Packets)
measurement "AvgPacketSize" expr(Measurement.Bytes / Measurement.Packets)
```

## 4.1.6 Histogram Measurement

Histogram measurements measure floating point values of a SandScript expression.

Syntax:

```
measurement "Name" histogram(<histogram_reference>, <expression>)
  {over <period>} {where <condition(s)>} {unique by <unique-by_spec>}
```

Where:

|                            |   |
|----------------------------|---|
| <b>histogram_reference</b> | A SandScript reference to a histogram that defines the ranges of the bins that measured values are counted into (Histogram.Name). |
| <b>expression</b>          | An integer or floating point valued expression to measure. Integer values are cast to floating point.                             |

Alternately, you can perform the measurement using the increment action and by specifying which histogram bin to increment. For example:

```
measurement "MyHistogramMeasurement" histogram(Histogram.MyHistogram)over publish_interval
if <condition> then increment Measurement.MyHistogramMeasurement bin <expression>
```

You can use the `by` parameter to the increment action to increment a histogram by more than 1. For example:

```
if <condition> then increment Measurement.MyHistogramMeasurement bin <expression> by 2
```



## 4.2 Measurement Groups

Measurement groups can be used to optimize policy by grouping measurements with similar conditions together, or to create a decision tree.

They may contain measurement definitions or nested measurement groups. The syntax and behavior is the same as a policy group except measurements instead of rules are declared within the group.

- The condition on the measurement group controls entry into the group. If the condition matches, the measurements are performed, otherwise they are not.
- If the keyword "all" is specified, all of the measurements in the measurement group that match their condition are performed. If "all" is not specified, only the first measurement that matches its condition is performed.
- Measurement groups can be nested.
- Measurement groups have an if-elseif-else syntax.

## 4.3 Histograms

Histograms are declared through policy to define bins for use with histogram measurements and controllers.

Histogram bins cover a continuous range of floating point numbers from -infinity to +infinity. A single histogram bin is represented by a lower and upper bound value, where the lower bound is exclusive, and the upper bound is inclusive.

Histogram types define how the bins are distributed over the range of floating point numbers. All histogram types have implied lower and upper outlier bins. The maximum number of bins is 200, not including the outlier bins.

The supported histogram types are:

- Linear - Evenly spaced bins defined over a range of floating point numbers.
- Custom - Consists of explicitly defined bin ranges.

Syntax:

```
histogram "<name>" <type> <parameters>
```

Where:

|                           |   |
|---------------------------|---|
| <b>&lt;name&gt;</b>       | The name of the histogram. The name is used to identify the histogram through CLI and SandScript. <ul style="list-style-type: none"><li>• Must begin with a letter and can only contain letters, digits, and underscores.</li><li>• Must not be a reserved SandScript keyword.</li><li>• Is not case sensitive.</li></ul> |
| <b>&lt;type&gt;</b>       | Specifies the histogram type. Refer to the details about specific histogram types that follow.  |
| <b>&lt;parameters&gt;</b> | Type-specific parameters for the histogram.   |

### 4.3.1 Linear Histogram

Linear histograms are defined by a minimum value, maximum value, and number of bins.

Syntax:

```
histogram "<name>" linear min <min_val> max <max_val> bins <num_bins>
```

Where:

- `min_val` is the lower bound of the first bin
- `max_val` is the upper bound of the final bin
- `num_bins` is the number of bins in the histogram.

Bins are of width  $(\text{max} - \text{min})/\text{bins}$ , with all of the bins covering the range  $\{\text{min}, \text{max}\}$ . The lower and upper outlier bins are defined over the ranges  $\{-\infty, \text{min}\}$ , and  $\{\text{max}, +\infty\}$  respectively.

## 4.3.2 Custom Histograms

Custom histograms are defined by a consecutive list of floating point values. The number of bins is one less than the number of values given (plus the two implied outlier bins).

Syntax

```
histogram "<name>" custom (<value1>, ... , <valueN>)
```

Where: `value1..valueN` are consecutive floating point values indicating the bin ranges for the histogram.

The range of each individual bin is defined by two consecutive values in the list, where the first value is the lower bound and the second value is the upper bound of the bin. The lower and upper outlier bins are defined over the ranges  $\{-\infty, \text{first}\}$ , and  $\{\text{last}, +\infty\}$ , where `first` and `last` are the first and last values in the list of floating point values.



### Example:

This example defines a linear set of histogram bins that divide the range 0-100 into 100 bins of the same size.

```
histogram "LinearHistogram" linear min 0 max 100 bins 100
```



### Example:

This example defines a custom set of histogram bins from 0-100 with finer granularity over numbers closer to zero.

```
histogram "CustomHistogram" custom (0, 0.5, 1, 1.5, 2, 3, 5, 10, 20, 30, 50, 70, 100)
```



### Example:

Using the histogram distributions defined above, this example keeps a running count of the entire network's round-trip time (RTT) measured into a custom histogram.

```
measurement "MeasureRttIntoCustom" histogram(\
    Histogram.CustomHistogram, \
    ToMilliseconds(Flow.Subscriber.HandshakeRTT)) \
where expr(Flow.Subscriber.HandshakeRTT is not null)
```



### Example:

Using the histogram distributions defined above, measure and publish the RTT per protocol into a linearly distributed histogram:

```
measurement "MeasureRttIntoLinear" histogram(\
    Histogram.LinearHistogram, \
    ToMilliseconds(Flow.Subscriber.HandshakeRTT)) \
over publish_interval \
where expr(Flow.Subscriber.HandshakeRTT is not null) \
unique by (Flow.ApplicationProtocol.StatsProtocol)
publish "RttByProtocol" measurement.MeasureRttIntoLinear
```



### Example:

Measure and publish a histogram of the number of bytes per flow unique by protocol. Use a histogram distribution with bins of size 500 bytes over the range 0-100000.

```
histogram "Bytes" linear min 0 max 100000 bins 200
measurement "FlowBytesPerProtocol" \
    histogram(histogram.Bytes, Flow.Rx.TotalBytes + Flow.Tx.TotalBytes) \
over publish_interval \
```

```
where expr(Flow.IsEnd) \  
unique by (Flow.ApplicationProtocol.StatsProtocol)  
publish "FlowBytesPerProtocol" measurement.FlowBytesPerProtocol
```



**Example:**

Measure flow duration in seconds into a custom histogram. The bins are distributed such that flow durations of a few seconds up to an entire day can be measured.

```
histogram "Seconds" custom(0, 1, 2, 3, 4, 5, 10, 20, 30, 60, 300, 600, 1800, 3600, 21600,  
43200, 86400)  
measurement "FlowAge" histogram(histogram.Seconds, Flow.Age) \  
over publish_interval \  
where expr(Flow.IsEnd)  
publish "FlowDuration" measurement.FlowAge
```





# 5

## Subscriber to IP Mapping

- ["Subscriber Aware SandScript Policy" on page 54](#)
- ["Subscriber Database Configuration" on page 55](#)
- ["Disabling Secure Tunnels Between the SPB and a PTS Element" on page 56](#)
- ["Troubleshooting Subscriber Mapping" on page 57](#)

## 5.1 Subscriber Aware SandScript Policy

Subscriber awareness is achieved through an IP address to subscriber mapping, and requires an SPB to be configured and connected to the PTS or SDE.

The ability to map a subscriber is a pre-requisite for any subscriber-aware policies, such as subscriber statistics collection or Top Talkers. An active IP address and optional site number to subscriber mapping is called a session for that subscriber. To identify a session, use the field `Session.Id`. This field returns an integer representing a unique identifier for this subscriber session or NULL if no subscriber session exists.

Attributes are type-value pairs, such as "servicelevel"="gold", that can be attached to a particular subscriber or to any of that subscriber's sessions. Each subscriber, or session, may have any number of attributes. These attributes may be used in conditions and typically represent the type of SandScript and type of service the subscriber will receive (for example, collect statistics, shape traffic, allow/deny and so on).

The SDE maintains a subscriber session as long as SDE SandScript continues to use the session.

### 5.1.1 Updating SandScript to Uniquely Identify Subscribers

You must update SandScript policy that uses a subscriber's IP address to uniquely identify a session to use the `Session.ID` field instead.

A common use-case in legacy custom SandScript is to use a subscriber's IP address as a unique handle for a subscriber session, for example in unique-by measurements, or as a table key. In cases where subscriber IP addresses overlap, a subscriber's IP address is no longer unique. Therefore, these policies must be updated to use the field `Session.ID` to uniquely identify a subscriber's session. The `Session.ID` field returns an integer identifier that is unique per session.

It is safe to use this field even if session qualifiers are not enabled, so Sandvine recommends updating any policies that use unique-by IP address to use `Session.ID` instead.

### 5.1.2 Subscriber Lookup

Subscriber to IP address mappings and all subscriber or session attributes are maintained by the SPB, which pushes notifications to, and accepts queries from, the PTS or SDE elements as needed. When traffic from an unknown subscriber IP address is received by the PTS or SDE, the subscriber who owns that IP address and all attributes of that subscriber are looked up.

To reduce network congestion, the PTS and SDE each use an algorithm which may apply a delay before either element's first look up request is sent to the SPB for any newly-discovered subscriber IP address. The algorithm operates one of these modes:

- Static mode (the default) has a pre-configured delay that does not change. If the static mode is enabled and the delay set to 0, there is no delay incurred on any lookup request sent by the PTS.
- Dynamic mode adjusts the size of delay based on the number of unnecessary lookups. An unnecessary lookup is one where a look up was requested but the IP becomes mapped by a push notification before the lookup response arrives. If there are too many unnecessary lookups, the delay is increased and when the percentage of unnecessary lookups drops, the delay is decreased.

These CLI commands can be used to manage subscriber lookup:

| CLI commands                             | Description   | Allowed | Default |
|--|---|---------|---------|
| show service subscriber-management stats | Displays the current delay time before the PTS will send a lookup request to the SPB. | N/A     | N/A     |

| CLI commands   | Description  | Allowed        | Default |
|--|--|----------------|---------|
| set config service<br>subscriber-management lookup<br>initial-delay mode         | Sets the lookup mode.  | dynamic static | static  |
| set config service<br>subscriber-management lookup<br>initial-delay static delay | If configured, bypasses dynamic lookup and sets a delay time for all lookups. Units of measure for configuration is milliseconds, therefore, the maximum is equivalent to 100 seconds. | 0-100000       | 0       |

### 5.1.3 Populating Subscriber to IP Address Mappings

Subscriber-IP mapping can be populated dynamically using DHCP, RADIUS, or GTP-C, or statically using a script.

In situations where Remote Authentication Dial-In User Service (RADIUS) Accounting is deployed, the mapping can be obtained from the accounting records. Where DHCP is used, the mapping can be obtained by intercepting DHCP traffic. Sandvine employs a highly flexible, SandScript-based application called Subscriber Mapping that can be used in a wide variety of environments to populate subscriber to IP address mappings.

For detailed information on configuring these methods for subscriber-IP mapping, including instructions for the PopulateSubIpMap script, refer to the *Subscriber Mapping User Guide*.

## 5.2 Subscriber Database Configuration

The database query behavior and subscriber management strategies such as timing out subscribers may need to be configured, using the appropriate CLI configuration command.

Normally, subscriber mapping behavior should not need to be changed from the default. Any changes should be done under the direction of Sandvine Customer Support as inappropriate configuration can have a detrimental effect on element and database performance.


#### Subscriber Database Query Configuration

These CLI configuration commands are used to define the subscriber database query behavior:

| CLI Configuration Command      | Description  | Allowed  | Default                      |
|--------------------------------|--|--|------------------------------|
| set config service spb version | Configures the SPB version that the PTS or SDE is connected to. Select the major version of the SPB when configuring. For example, choose "5.60" if connected to an SPB running any maintenance release of 5.60.   | 6.20, 6.00, 5.60, 5.51, 5.40                                   | Newest supported SPB version |
| set config service spb servers | Contains a space-separated list of hostnames and/or IP addresses of SPB devices the PTS or SDE connects to. The PTS or SDE connects to only one of the servers in this list, but if the connection fails it will fallback to other servers. By default, the PTS or SDE uses an ssl-encrypted connection; if any server is configured with a tcp:// prefix, the PTS or SDE will use an un-encrypted TCP connection. | Any valid host name or IP address, with optional tcp:// prefix | Empty                        |

### Subscriber Management Configuration

These CLI configuration commands are used to define the subscriber management behavior.

| CLI Configuration Command   | Description   | Allowed                        | Default |
|---|---|--------------------------------|---------|
| set config service subscriber-management timeout inactivity               | Sets the timeout length (in seconds) after which a subscriber-IP mapping expires if no traffic is received for that IP address. When traffic is again seen for that IP, a lookup will be issued.  | 1 to 1209600 seconds (14 days) | 1200    |
| set config service subscriber-management timeout lookup initial           | Sets the initial delay between subscriber IP lookup requests. The delay between requests doubles with each request.   | 1 - 86400 seconds (24 hours)   | 120     |
| set config service subscriber-management timeout lookup max               | Sets the maximum possible delay between subscriber IP lookup requests.  | 1 - 86400 (24 hours)           | 32400   |
| set config service subscriber-management timeout late                     | Sets the maximum delay between the initial lookup request for a subscriber IP and that IP becoming mapped, after which the mapping is considered "late".<br>Related alarm is Alarm Model 30: Subscriber mapping on PTS is delayed.  | 1 - 120                        | 30      |
| set config service subscriber-management login-events handle-notification | Configures the PTS or SDE to process IP assignment notifications from the SPB   | true false                     | true    |
| set config service subscriber-management lookup max-attempts              | Maximum number of unsuccessful subscriber-IP lookup attempts the PTS or SDE will do before giving up. Set to 0 for no limit.  | 0 - 255                        | 0       |
| set config service subscriber-management lookup on-receive                | Configures the PTS or SDE to perform subscriber lookups when the subscriber IP receives unidirectional data.<br><br> <b>Note:</b><br>The PTS or SDE always does subscriber lookups when the subscriber IP is sending data. | true false                     | true    |

## 5.3 Disabling Secure Tunnels Between the SPB and a PTS Element

By default, communication between the SPB and a PTS element is encrypted using a secure tunnel.

In situations where messaging performance is critical, such as when the rate of subscriber mappings is higher than 4000 mappings/sec, or the volume of statistics published is high, disabling security tunnels will improve messaging performance on the PTS.

To disable security tunnels, prefix the SPB element's IP address with `tcp://` when entering addresses in the Quickstart menu, or in the CLI configuration mode. For example, using `tcp://10.10.10.1` creates a connection bypassing the security tunnels.

To verify that the SPB is properly connected, at the CLI prompt enter `show spb connections`. Sample output:

```
Connections
=====
Id      AdminStatus  OperStatus  Type
-----
11863   [up]         [connected] Statistics database
```



| APIVersion | ConfiguredURI    | ConnectedURI     | Failures | LastTimeConnected |
|------------|------------------|------------------|----------|-------------------|
| [2]        | tcp://10.10.10.1 | tcp://10.10.10.1 | 1        | 20100325_153719   |

The `show spb connections` CLI command does not show the IP of the SPB until a connection is established. Instead it shows 127.0.0.1.

To revert to the high-security configuration, remove the `tcp://` prefix from the SPB IP address.

## 5.4 Troubleshooting Subscriber Mapping

You can monitor the subscriber-to-IP-address mappings cache using the `show service subscriber-management dashboard` CLI command.

The dashboard runs continuously, displaying running totals for variables such as mapping and un-mapping rates, and the current number of mapped and un-mapped subscribers in the system. Some statistics of particular interest include:

- `LookupsTimedOut` counter, indicates that there was a delay from the SPB during a subscriber lookup which could indicate the SPB is becoming overloaded.
- `IpNotInSubscriberClass` counts subscribers found in a network class defined in `subnets.txt` but that have not been defined in `subscriber_classes` in `policy.conf`.

For more information about this command, see the *Sandvine Operations Reference Guide*. Also see the *Subscriber Mapping User Guide*.





# 6

## Maps

- ["Map Overview" on page 60](#)
- ["String Maps" on page 60](#)
- ["Hostname Maps" on page 62](#)
- ["URL Maps" on page 63](#)
- ["IP Address Maps" on page 64](#)
- ["Loading Map Contents" on page 65](#)
- ["Map File Format" on page 65](#)

## 6.1 Map Overview

Maps define a set of items and provides the ability to test for an item's existence within the set.

Each item in the map can be associated with an optional label, which can be accessed when looking up a specific item in policy. Items defined within the map can contain a glob match character (\*) which gives extra flexibility when defining items in a map.

The different map types are:

- String map allows for a map to contain any string.
- Host map provides options to perform normalized match of hostnames.
- URL map provides option to perform normalized match of URLs. For use cases where maps contain only full URLs.
- IP Address map is optimized to match IP addresses. For use cases where maps contain only IP addresses.

You can view the currently defined maps using the `show policy map` CLI command which gives aggregate, real-time statistics that exposes information such as number of total queries against the map, number of total matches, and more.

When using maps, note these limitations:

- The PTS supports a maximum of 100,000 map entries across all maps in the system. SandScript policy will not load if you exceed this limit.
- The PTS supports a maximum of 10,000 entries with the glob-match character across all maps.
- The PTS 24000 and 22000 provides 80Mb of memory to be used for maps and the PTS 14000 and 8210 provide 40Mb of memory. Use the `show policy map` CLI command to monitor resource usage.

## 6.2 String Maps

The syntax is:

```
map "<name>" string {label_type string|integer} {case_sensitive true|false}
{escaping true|false} files <file_list>|values <value_list>
```

Where:

|                       |  |
|-----------------------|--|
| <b>name</b>           | A unique name for the map. Note that name: <ul style="list-style-type: none"> <li>• Must begin with a letter and can only contain letters, digits, and underscores.</li> <li>• Must not be a policy reserved keyword.</li> <li>• Is not case sensitive.</li> <li>• Cannot conflict with other function names (user-defined and internal).</li> </ul> |
| <b>label_type</b>     | Specifies optional labels along with the entries. Defaults to string.  |
| <b>case_sensitive</b> | Indicates if case sensitive match should be used. Defaults to false.   |
| <b>escaping</b>       | Enables escaping certain characters in the patterns of policy maps. Defaults to false if not specified. Valid escape sequences are:  |

| Escape sequence | Meaning |
|-----------------|---------|
| \#              | hash    |
| \               | space   |
| \t              | tab     |

| Escape sequence | Meaning         |
|-----------------|-----------------|
| \r              | carriage return |
| \n              | newline         |
| \\              | backslash       |

**files <file\_list>** A list of quoted file names separated by spaces. These files will be read to populate the map. They can be absolute path, or relative to `/usr/local/sandvine/etc/`. This is the recommended source for a string map whenever possible.

**values <value\_list>** A list of quoted strings separated by white space that define the contents of the map.

## 6.2.1 SandScript Functions for a String Map

Use functions to operate on the map.

Each map function has this syntax:

`Map.<Name>.<Function>`

Where:

- **Name:** is the unique name given to the map in the map definition.
- **Function:** each SandScript function transforms input and finds a match according to optional parameters specified in the map declaration. The functions that are provided by a string map are:

| Function name                    | Description   |
|----------------------------------|---|
| <code>Contains(string)</code>    | Returns true if <i>string</i> is matched by a key/pattern in the map, compared as specified by <i>case_sensitive</i> parameter in the map declaration.  |
| <code>FindLabel(string)</code>   | Returns the label associated with a key/pattern, if <i>string</i> is matched by a key/pattern in the map, compared as specified by the <i>case_sensitive</i> parameter in the map declaration. The return type is an integer/string as specified by <i>label_type</i> parameter in the map declaration. |
| <code>FindPattern(string)</code> | Returns the key/pattern associated with a key/pattern, if <i>string</i> is matched by a key/pattern in the map, compared as specified by the <i>case_sensitive</i> parameter in the map declaration.  |



### Example:

This example uses a map file called `subscriberlist.txt`:

```
00:30:48:30:0d:8a
00:30:48:30:0d:8b
00:30:48:30:0d:8c
```

To access the map file:

```
map "MyMap" string files "subscriberlist.txt"
```

We can use "MyMap" with the pre-defined measurement "Matches":

```
measurement "Matches"
if expr(Map.MyMap.Contains(Flow.Client.Subscriber.Name) then \
increment measurement.Matches
```

## 6.3 Hostname Maps

The hostname map allows for some extra functionality and optimization when the keys/patterns in a map are all host names.

The syntax is:

```
map "<name>" hostname {label_type string|integer} {case_sensitive true|false}
{normalize true|false} files <file_list>|values <value_list>
```

The same parameters as string maps apply, plus `normalize` performs normalized match if set to true.

Normalization allows identifying two syntactically different URLs that represent the same resource. Policy maps support normalizing URLs before doing a match so that two such different URLs can be matched. Normalization support includes removing percent-encoding and dot-segments for a URL before matching URL lists. For example, equivalent URLs are:

| Unnormalized URL   | Normalized URL   |
|--|--|
| <code>http://www.example.com/./a/b/./c/./d.html</code>           | <code>http://www.example.com/a/c/d.html</code>                 |
| <code>http://en.wikipedia.org/wiki/URL-%2FPercentencoding</code> | <code>http://en.wikipedia.org/wiki/URL/Percent-encoding</code> |

### 6.3.1 SandScript Functions for Hostname Maps

The SandScript functions that are provided for a hostname map are:

| Function name                      | Description   |
|------------------------------------|---|
| <code>Contains(hostname)</code>    | Returns true if <i>hostname</i> is matched by a key/pattern in the map, compared as specified by <i>case_sensitive</i> parameter in the map declaration.  |
| <code>FindLabel(hostname)</code>   | Returns the label associated with a key/pattern, if <i>hostname</i> is matched by a key/pattern in the map, compared as specified by the <i>case_sensitive</i> parameter in the map declaration. The return type is an integer/string as specified by <i>label_type</i> parameter in the map declaration. |
| <code>FindPattern(hostname)</code> | Returns the key/pattern associated with a key/pattern, if <i>hostname</i> is matched by a key/pattern in the map, compared as specified by the <i>case_sensitive</i> parameter in the map declaration.  |
| <code>FindHost(hostname)</code>    | Returns the host, if <i>hostname</i> is matched by a key/pattern in the map, as specified by <i>normalize</i> and <i>case_sensitive</i> parameters in the map declaration, or null if the <i>hostname</i> isn't found.  |



**Example:**

This example uses a map file called `hostnamelist.txt`:

```
www.example.com
www.sandvine.com
```

The policy to access the map file is:

```
map "MyMap" hostname normalize true case_sensitive false files "hostnamelist.txt"
```

We can use "MyMap" with the pre-defined measurement "Matches":

```
measurement "Matches"  
if expr(Map.MyMap.Contains(Flow.Client.Stream.Http.Host) then \  
increment measurement.Matches
```

## 6.4 URL Maps

The syntax is:

```
map "<name>" url {{case_sensitive true|false} {normalize true|false}  
{resource_param_match_order exact|any}} {{files <file_list>}}{{values <value_list>}}
```

The same parameters as for string maps apply. In addition, `resource_param_match_order` defines how parameters contained in URLs should be checked when attempting to determine if an input URL is contained in a map. If `exact` (the default), the map attempts to match all text following the "?" character in a URL as if it were a string. If `any`, it attempts to match URL parameters in any order in which they may appear.

For the input "www.sandvine.com/index.html?a=1&b=2", the matches are:

| Map entry                           | Resource_param_match_order | Match? |
|-------------------------------------|----------------------------|--------|
| www.sandvine.com/index.html?a=1&b=2 | exact                      | Yes    |
| www.sandvine.com/index.html?a=1&b=2 | any                        | Yes    |
| www.sandvine.com/index.html?b=2&a=1 | exact                      | No     |
| www.sandvine.com/index.html?b=2&a=1 | any                        | Yes    |

### 6.4.1 SandScript Functions for URL Maps

A URL map type has a set of policy functions used to operate on the map in policy.

The syntax is:

```
Map.Name.Function(hostname, resource)
```

Where:

- **Name**—Is the unique name given to the map in the map definition.
- **Function**—Includes any of these functions:

| Function name                                | Description   |
|--|---|
| <code>Contains(hostname, resource)</code>    | Returns true if <i>hostname</i> matches a key/pattern in the map, as specified by the <i>case_sensitive</i> parameter in the map declaration.   |
| <code>FindLabel(hostname, resource)</code>   | Returns the label associated with a key/pattern, if <i>hostname</i> matches a key/pattern in the map, as specified in the <i>case_sensitive</i> parameter in the map declaration. The return type is an integer or string, as specified by <i>label_type</i> parameter in the map declaration. Returns null if no label is found. |
| <code>FindPattern(hostname, resource)</code> | Returns the key/pattern associated with a key/pattern, if <i>hostname</i> matches a key/pattern in the map, as specified in the <i>case_sensitive</i> parameter in the map declaration. Returns null if there is no match.  |

| Function name                                     | Description   |
|---|---|
| FindHost( <i>hostname</i> , <i>resource</i> )     | Returns the host, if <i>hostname</i> matches a key/pattern in the map, as specified by the <i>normalize</i> , <i>case_sensitive</i> , and <i>resource_param_match_order</i> parameters in the map declaration. Returns null if there is no match. |
| FindResource( <i>hostname</i> , <i>resource</i> ) | Returns the resource, if <i>resource</i> matches a key/pattern in the map, as specified by <i>normalize</i> , <i>case_sensitive</i> , and <i>resource_param_match_order</i> parameters in the map declaration. Returns null if there is no match. |
| FindUrl( <i>hostname</i> , <i>resource</i> )      | Returns the URL, if <i>URL</i> matches a key/pattern in the map, as specified by <i>normalize</i> , <i>case_sensitive</i> , and <i>resource_param_match_order</i> parameters in the map declaration. Returns null if there is no match.           |



#### Example:

For example:

We have `urllist1.txt`

```
www.example.com/
www.test.com/req=sample1
```

And `urllist2.txt`

```
www.testsample.com/test/*
www.testsample.com/request*
http://www.sandvine.com/example.html?a=1&b=2
http://www.example2.com/*
```

The policy to access these map files is:

```
map "MyMap" url normalize true case_sensitive false resource_param_match_order exact files
    "urllist1.txt" "urllist2.txt"
```

We can use "MyMap" with the pre-defined measurement "Matches":

```
measurement "Matches"
if expr (Map.MyMap.Contains(
    Flow.Client.Stream.Http.Host, \
    Flow.Client.Stream.Http.Resource)) then \
    increment measurement.Matches
```

## 6.5 IP Address Maps

This map is optimized to match IP addresses, for use cases where maps contain only IP addresses (IPv4 and IPv6).

The syntax is:

```
map "<name>" ipaddress {label_type string|integer}
    {escaping true|false}{{files <file_list>}}{{values <value_list>}}
```

Use functions to operate on the map in SandScript. Each function has the syntax:

```
Map.<Name>.<Function>
```

Where:

- Name - Is the unique name given to the map in the map definition.
- Function - Transforms input and finds a match according to optional parameters specified in the map declaration.

The functions provided for IP address maps are:



| Function name                 | Description   |
|-------------------------------|---|
| Contains( <i>ipaddress</i> )  | Returns true if <i>ipaddress</i> is matched by an IP address in the map.  |
| FindLabel( <i>ipaddress</i> ) | Returns the label associated with an IP address, if <i>ipaddress</i> is matched by an IP address in the map. The return type is an integer/string as specified by <i>label_type</i> parameter in the map declaration. |



**Example:**

For example, to increment the IP address map:

```
map "MyMap" ipaddress files "subscriberIPs.txt"
if expr (Map.MyMap.Contains (Flow.Internet.IPAddress)) then \
increment measurement.Matches
```

To use the FindLabel() function with IP Address maps:

```
map "MyMap" ipaddress label_type string files "subscriberIPs.txt"
string "MyString" = Map.MyMap.FindLabel (IPAddress ("140.211.166.64"))
if (MyString is not null)
{
    # use MyString here
}
```

## 6.6 Loading Map Contents

Whenever the properties of a map are changed (map type, data source, and so on) a full policy reload must be performed.

Additionally, if the source type for the map is “values”, a full policy reload must also be performed if the values in the map are to be changed. However, if the source type of the map is “files”, the content of the maps can be modified without a full reload, meaning that while the new map data is being loaded, policy continues to be applied until the new content is loaded, at which point the new map content is swapped seamlessly.

To perform a reload of maps contents when the source type is “files” at the command line, run:

```
SDE> svreload -maps
```

If this reload fail for any reason, policy continues to be applied with the content of the map that existed before the reload was issued.

## 6.7 Map File Format

When the source of map contents is defined as files, each file is read in one line at a time.

Each line should be of the format: <key> <optional\_label>. Where:

- key is the item that will be stored in the map. It may contain one or more occurrences of the glob-match character (\*). Note that the validity of a key depends on the type of map.

- For URL maps, <key> should be a valid URL and can optionally start with `http://`.
- For hostname maps, <key> should be a valid host name, it can optionally start with `http://`.
- For string maps, <key> can be any string.
- optional\_label is an integer or string value that is associated with the value in the map. While adding a label is optional, if you specify an integer label, then you must include the value. A string label will default to an empty string. All optional\_labels in the file must be the same type as the label\_type parameter in the map declaration.



**Example:**

For example:

```
www.example.com 1 www.sandvine.com 2
```

## 6.7.1 Glob Matching

The key defined within maps may contain the glob match character (\*) to give added flexibility when defining keys within a map.

One or more glob match characters may be used in a key. For example, if this key is specified within a map:

```
www.example*.com
```

Then any of these inputs will be matched by the map:

```
www.example1.com
www.exampleAAA.com
www.example.com
```

If you wish a particular pattern to be matched before all others, put it into its own map to be searched first. For example, say we have these patterns:

```
www.exam*.com
www.example*.com
```

Because of the glob match character, both these patterns will match the string "http://www.exampleAAA.com". To match the more specific `www.example*.com` first, put it into its own map file, called `MapSpecific`, and the more general keys into `MapGeneral`. Then create policy to search the more specific map first:

```
Classifier "URL count" type string
if true then set Flow.Classifier.URL_count = \
    Map.MapSpecific.FindLabel(Flow.Client.Stream.Http.URL)
if expr(Flow.Classifier.URL_count is null) then set Flow.Classifier.URL_count = \
    Map.MapGeneral.FindLabel(Flow.Client.Stream.Http.URL)
```

## 6.7.2 Normalization for Hostname and URL Maps

Normalization allows identifying two syntactically different URLs that represent the same resource. Policy maps support normalizing URLs before doing a match so that two such different URLs can be matched. Normalization support includes removing percent-encoding and dot-segments for a URL before matching URL lists. For example, equivalent URLs are:

| Unnormalized URL   | Normalized URL   |
|--|--|
| <code>http://www.example.com/./a/b/./c/./d.html</code>           | <code>http://www.example.com/a/c/d.html</code>                 |
| <code>http://en.wikipedia.org/wiki/URL-%2FPercentencoding</code> | <code>http://en.wikipedia.org/wiki/URL/Percent-encoding</code> |



# 7

## Diameter Stack Configuration

- ["The Diameter Stack" on page 68](#)
- ["Diameter Stack Base Protocol Messages" on page 68](#)

## 7.1 The Diameter Stack

The Diameter stack is enabled by default when Diameter peers are correctly configured.

The information in this guide is for advanced configuration, debugging and informational purposes, and is therefore not required information for a typical deployment.

The Diameter peer configuration file is /usr/local/sandvine/etc/diam\_peer\_config.xml. Policy exists to handle Diameter messaging; see [Diameter](#) on page 23.

## 7.2 Diameter Stack Base Protocol Messages

A session begins when the Diameter stack sends a Capabilities Exchange Request (CER) to one of its configured Diameter peer servers.

If an acceptable Capabilities Exchange Answer (CEA) is received, the stack moves into the connected state and begins to send periodic Device Watchdog Request (DWR) and expects to receive Device Watchdog Answer (DWA) Messages in response. Any DWR's received by the Diameter stack are responded to by an appropriate DWA. Disconnect Peer Requests (DPR) are currently handled by closing the connection, no Disconnect Peer Answer (DPA) is sent. The Diameter stack does not currently generate DPR requests and thus does not handle DPA messages.

For outgoing message definitions, mandatory AVPs are included with every outgoing message. Optional AVPs may or may not be included with the outgoing message, depending on the stack configuration.

For incoming message definitions, mandatory AVPs are required with every incoming message. Optional-C AVPs are used by the dictionary, but incoming messages can include any number of instances (including zero) in the message. Optional AVPs are listed in the RFC, but the diameter stack does not explicitly check or use them.

Further information is available in:

- RFC 3588 - Section 2.5 Connections vs. Sessions
- RFC 3588 - Section 3.2 Command Code ABNF specification

### 7.2.1 CER Messages

A Capabilities-Exchange-Request (CER) is sent by a diameter peer to initiate a connection to another diameter peer.

#### Outgoing CER Messages

The Diameter stack initiates a session by sending a CER message with this format:

| AVP                | Category  | Details                                 |
|--------------------|-----------|---|
| Origin-Host        | Mandatory | Configurable via changing LocalIdentify |
| Origin-Realm       | Mandatory | Configurable via changing LocalRealm    |
| Host-IP-Address    | Mandatory | Configurable via changing LocalIp       |
| Vendor-Id          | Mandatory | Configurable via VendorId               |
| Product-Name       | Mandatory | Configurable via ProductName            |
| Origin-State-Id    | Mandatory | Not configurable                        |
| Inband-Security-Id | Mandatory | Not configurable                        |

| AVP                             | Category  | Details                                |
|---------------------------------|-----------|--|
| Firmware-Revision               | Mandatory | Not configurable                       |
| Supported-Vendor-Id             | Optional  | Configurable via dictionary definition |
| Auth-Application-Id             | Optional  | Configurable via dictionary definition |
| Acct-Application-Id             | Optional  | Configurable via dictionary definition |
| Vendor-Specific- Application-Id | Optional  | Configurable via dictionary definition |

#### Incoming CER messages

The Diameter stack drops any incoming CER messages.

## 7.2.2 CEA Messages

A Capabilities-Exchange-Answer (CEA) is sent by a Diameter peer in response to a CER message.

#### Outgoing CEA Messages

Since the Diameter stacks drops any incoming CER messages, no CEA messages are sent.

#### Incoming CEA messages

The AVPs listed in the table are mandatory. If any are missing in an incoming CEA, the Diameter stack will close the connection. If there is no set of common supported Application Ids, the connection is also closed. The Diameter stack expects any incoming CEA messages to be of the format described in the table.

| AVP                            | Category   | Details                                    |
|--------------------------------|------------|--|
| Result-Code                    | Mandatory  | Must be DIAMETER_SUCCESS                   |
| Origin-Host                    | Mandatory  |  |
| Origin-Realm                   | Mandatory  |  |
| Vendor-Specific-Application-ID | Optional-C | Used to determine negotiated applications. |
| Auth-Application-Id            | Optional-C | Used to determine negotiated applications. |
| Acct-Application-Id            | Optional-C | Used to determine negotiated applications. |
| Host-IP-Address                | Optional   |  |
| Vendor-IP                      | Optional   |  |
| Product-Name                   | Optional   |  |

## 7.2.3 DWR Messages

A Diameter peer may send a Device-Watchdog-Request (DWR) if no messages have been exchanged for some time.

#### Outgoing DWR Messages

The Diameter stack periodically sends DWR messages with this format:

| AVP             | Category  | Details                                  |
|-----------------|-----------|--|
| Origin-Host     | Mandatory | Configurable via changing LocalIdentify. |
| Origin-Realm    | Mandatory | Configurable via changing LocalRealm.    |
| Origin-State-Id | Mandatory | Not configurable.                        |

#### Incoming DWR Messages

The incoming DWR message format is:

| AVP             | Category | Details |
|-----------------|----------|---------|
| Origin-Host     | Optional |         |
| Origin-Realm    | Optional |         |
| Origin-State-Id | Optional |         |

## 7.2.4 DWA Messages

A peer that receives a DWR message must respond with a Device-Watchdog-Answer (DWA).

#### Outgoing DWA Messages

The Diameter stack responds to an incoming DWR message with a DWA message with this format:

| AVP             | Category  | Details   |
|-----------------|-----------|---|
| Origin-Host     | Mandatory | Configurable via changing LocalIdentify           |
| Origin-Realm    | Mandatory | Configurable via changing LocalRealm              |
| Origin-State-Id | Mandatory | Not configurable                                  |
| Result-Code     | Mandatory | Not configurable. Always set to DIAMETER_SUCCESS. |

#### Incoming DWA Messages

Upon reception of a DWA, the Diameter stack updates its connection timeout. The format of the incoming DWA must only have a valid header and command code. Thus, the Diameter stack expects any incoming DWA message to be of this format:

| AVP               | Category | Details |
|-------------------|----------|---------|
| Result-Code       | O        |         |
| Origin-Host       | O        |         |
| Origin-Realm      | O        |         |
| Error-Messages    | O        |         |
| Failed-AVP        | O        |         |
| Original-State-Id | O        |         |

## 7.2.5 Outgoing DPR Messages

Upon reception of a Disconnect-Peer-Request (DPR) message, the connection is immediately closed, no outgoing DPA message is sent in response. The format of the incoming DPR must only have a valid header and command code. Thus, the Diameter stack expects any incoming DPR message to be of the format:

| AVP              | Category | Details |
|------------------|----------|---------|
| Origin-Host      | O        |         |
| Origin-Realm     | O        |         |
| Disconnect-Cause | O        |         |







# 8

## DHCP Actions

- ["DHCP Actions" on page 74](#)
- ["LeaseQueryByIp Syntax" on page 74](#)
- ["DHCPv6.LeaseQueryByClientID Syntax" on page 74](#)
- ["DHCPv4.LeaseQueryByMac Syntax" on page 75](#)

## 8.1 DHCP Actions

Use DHCP actions to discover information concerning subscriber IP addresses from a DHCP server.

SandScript provides support for DHCPv4 and DHCPv6.

When using the Subscriber Mapping application for DHCP-based mappings, Sandvine recommends using the Subscriber Mapping's native DHCP LEASEQUERY reconciliation mechanism.

## 8.2 LeaseQueryByIp Syntax

This action sends a lease query by IP address to the specified DHCP server. It returns true if the request was sent; false otherwise.

```
[DHCPv4|DHCPv6].LeaseQueryByIp({Status: <status>}, Peer: <peer>, \
  IPAddress: <address>)
```

Where:

|                        |   |
|------------------------|---|
| <b>[DHCPv4 DHCPv6]</b> | specifies the version of the DHCP server.   |
| <b>Status</b>          | is an optional output parameter that indicates the action status. The error codes are: <ul style="list-style-type: none"> <li>• 0 - request was sent.</li> <li>• 1 E_PEER_UNCONFIGURED - peer does not exist in configuration.</li> <li>• 100 E_INVALID_PEER - invalid peer parameter.</li> <li>• 201 E_INVALID_PARAM - invalid IpAddress parameter.</li> </ul> |
| <b>Peer</b>            | is the IP address of the configured DHCP server to send the lease query to. Use an IPv4 address with a DHCPv4 server and an IPv6 address with a DHCPv6 server.  |
| <b>IpAddress</b>       | is the client IP address to query. Use an IPv4 address with a DHCPv4 server and an IPv6 address with a DHCPv6 server.   |



### Example:

For example, to make a IPv4 lease query:

```
if true then \
set rc=DHCPv4.LeaseQueryByIp( \
  Status: st, \
  Peer: StringToIp("10.10.10.1"), \
  IPAddress: StringToIp("10.10.10.2"))
```



### Example:

To make a IPv6 lease query:

```
if true then set rc=DHCPv6.LeaseQueryByIp( \
  Status: st, \
  Peer: StringToIp("10:10:10:10:10:10:10:10"), \
  IPAddress: StringToIp("10:10:10:10:10:10:10:11"))
```

## 8.3 DHCPv6.LeaseQueryByClientID Syntax

Sends a lease query by Client ID to the specified DHCPv6 server.

```
DHCPv6.LeaseQueryByClientID({Status: <status>}, Peer: <peer>, ClientID: <id>)
```

Where:

**Status** is an optional output parameter that indicates the action status. The error codes are:

- 0 - request was sent.
- 1 E\_PEER\_UNCONFIGURED - peer does not exist in configuration.
- 100 E\_INVALID\_PEER - invalid peer parameter.
- 201 E\_INVALID\_PARAM - invalid ClientID parameter.

**Peer** is the IPv6 address of the configured DHCP server to send the lease query to.

**ClientID** is the client ID to query.



**Example:**

For example:

```
if true then set rc=DHCPv6.LeaseQueryByIp( \  
  Status: st, \  
  Peer: StringToIp("10:10:10:10:10:10:10:10"), \  
  ClientId: HexToOctets("0054FEDE"))
```

## 8.4 DHCPv4.LeaseQueryByMac Syntax

This action sends a lease query by MAC address to the specified DHCP server. It returns true if the request was sent; false otherwise.

```
DHCPv4.LeaseQueryByMac({Status: <status>,} Peer: <peer>, Mac: <address>)
```

Where:

**Status** is an optional output parameter that indicates the action status. The error codes are:

- 0 - request was sent.
- 1 E\_PEER\_UNCONFIGURED - peer does not exist in configuration.
- 100 E\_INVALID\_PEER - invalid peer parameter.
- 201 E\_INVALID\_PARAM - invalid Mac parameter.

**Peer** is the IPv4 address of the configured DHCP server to send the lease query to.

**Mac** is the client MAC address to query.



**Example:**

For example:

```
if (true) then set rc=DHCPv4.LeaseQueryByIp( \  
  Status: st, \  
  Peer: StringToIp("10.10.10.1"), \  
  Mac: StringToMac("11:11:11:11:11:11"))
```





# 9

## Logging

- ["Logging" on page 78](#)
- ["Logging.<LogId>.<DestinationId>.Write Syntax" on page 78](#)
- ["Logging.<LogId>.<DestinationId>.Flush Syntax" on page 78](#)
- ["Logging.<LogId>.<DestinationId>.Close Syntax" on page 79](#)
- ["Logging Example" on page 79](#)

## 9.1 Logging

SandScript logging actions are used to log information to the SDE logging subsystem, based on the log structure defined in `logging_dictionary.xml`.

You can use the logging actions to log information, such as usage data, or to log debug information. For more information about the dictionary file, the mandatory parameters to log, and how to configure the logging subsystem, see the *SDE User Guide*.

## 9.2 Logging.<LogId>.<DestinationId>.Write Syntax

Writes log data to the specified destination.

Returns an error code for the operation. If error code is less than 100 then the operation succeeded. Possible error codes are:

- 2 - successfully wrote to destination file.
- 101 - failed to write to destination file.
- 102 - failed to open destination file.

```
Logging.<LogId>.<DestinationId>.Write( \  
  {{metadataName1: <name1>, metadataValue1: <value1>}}\  
  {, metadataName2: <name2>, metadataValue2: <value2>} ...}  
  fieldName1: <name3>, fieldValue1: <value3>\  
  {, fieldName2: <name4>, fieldValue2: <value4>, ...} )
```

Where:

|                      |   |
|----------------------|---|
| <b>LogId</b>         | is the name of the log structure which is defined at <code>logging_dictionary.xml</code> .  |
| <b>DestinationId</b> | is the name of the destination under the LogId which defined at <code>logging_dictionary.xml</code>   |
| <b>metadataName</b>  | is the metadata parameter which is embedded inside the <code>fileName</code> , <code>fileExtension</code> and <code>subPath</code> tags (one or more) of destination's file tag at <code>logging_dictionary.xml</code> . Metadata parameters are used to modify those parameters dynamically from the policy engine so that the path for the output file can be defined on the fly. It is required to set all metadata parameters which are defined for the given <code>destinationId</code> within <code>logging_dictionary.xml</code> . |
| <b>metadataValue</b> | is the values to set for the corresponding metadata parameter.  |
| <b>fieldName</b>     | is the field to be set. All fields are declared under the LogId tag at <code>logging_dictionary.xml</code> . All mandatory fields must be assigned.   |
| <b>fieldValue</b>    | is the values to set for the corresponding field  |

## 9.3 Logging.<LogId>.<DestinationId>.Flush Syntax

Flushes log data from the specified destination.

Returns true if destination's buffer was flushed; false otherwise.

```
Logging.<LogId>.<DestinationId>.Flush( \  
  {{metadataName1: <name1>, metadataValue1: <value1>}} \  
  {, metadataName2: <name2>, metadataValue2: <value2>} ...})
```

Where:

|                      |  |
|----------------------|--|
| <b>LogId</b>         | is the name of the log structure which is defined at logging_dictionary.xml.   |
| <b>DestinationId</b> | is the name of the destination under the LogId which defined at logging_dictionary.xml   |
| <b>metadataName</b>  | is the metadata parameter which is embedded inside the fileName, fileExtention and subPath tags (one or more) of destination's file tag at logging_dictionary.xml. Metadata parameters are used to modify those parameters dynamically from the policy engine so that the path for the output file can be defined on the fly. It is required to set all metadata parameters which are defined for the given destinationId within logging_dictionary.xml. |
| <b>metadataValue</b> | is the values to set for the corresponding metadata parameter.   |

## 9.4 Logging.<LogId>.<DestinationId>.Close Syntax

Performs a rotation on the log file for the specified destination. Rotation is when the open file is closed and moved to the target folder and a new empty file is opened for the destination.

Returns true if destination's file was rotated; false otherwise.

```
Logging.<LogId>.<DestinationId>.Close(\
  {{metadataName1: <name1>, metadataValue1: <value1>}}\
  {, metadataName2: <name2>, metadataValue2: <value2>} ...})
```

Where:

|                      |  |
|----------------------|--|
| <b>LogId</b>         | is the name of the log structure which is defined at logging_dictionary.xml.   |
| <b>DestinationId</b> | is the name of the destination under the LogId which defined at logging_dictionary.xml   |
| <b>metadataName</b>  | is the metadata parameter which is embedded inside the fileName, fileExtention and subPath tags (one or more) of destination's file tag at logging_dictionary.xml. Metadata parameters are used to modify those parameters dynamically from the policy engine so that the path for the output file can be defined on the fly. It is required to set all metadata parameters which are defined for the given destinationId within logging_dictionary.xml. |
| <b>metadataValue</b> | is the values to set for the corresponding metadata parameter.   |

## 9.5 Logging Example

This simple example illustrates the relationship between logging policy and the logging dictionary.

The policy is:

```
# local file will be created under
# /usr/local/sandvine/var/spool/logging/local/tmp/local/
if true then \
  Logging.SoapTest.Request.Write(sub_id: "sub_name", \
                                usage: 100)
```

And the logging dictionary is:

```
<Logs>
  <ConfVersion>6.00.0001</ConfVersion>

  <Log>
```

```
<!-- Matches policy Logging.SoapTest -->
<LogId>SoapTest</LogId>
<LogType>Text</LogType>

<Destinations>
  <File>
    <!-- Matches policy Logging.SoapTest.Request -->
    <DestinationId>Request</DestinationId>
    <FileName>SanityTest</FileName>
    <FileExtension>log</FileExtension>
    <FileSuffix>TimeStampWithIndex</FileSuffix>
    <!--
    Note:
    ====
    <LocalRootPath>/< TargetRootPath> tags are RELATIVE paths to the
    actual local/target files location:
    /usr/local/sandvine/var/spool/logging/local/<LocalRootPath>/<SubPath>
    /usr/local/sandvine/var/spool/logging/target/<TargetRootPath>/<SubPath>
    -->
    <LocalRootPath>/tmp/local</LocalRootPath>
    <TargetRootPath>/tmp/target</TargetRootPath>
    <SubPath></SubPath>
    <CharacterEncoding>Ascii</CharacterEncoding>
    <EOR>Unix</EOR>
    <RecordsBufferSize>1</RecordsBufferSize>
    <RotateOnFileSize>1024</RotateOnFileSize>
    <RotateOnRecordsNumber>3</RotateOnRecordsNumber>
  </File>
</Destinations>

<Record>
  <Field>
    <!-- First parameter passed to the policy action -->
    <Name>sub_id</Name>
    <DataType>String</DataType>
    <Type>Variable</Type>
    <Length>20</Length>
    <Alignment>Left</Alignment>
    <Padding>Spaces</Padding>
    <Mandatory>Yes</Mandatory>
  </Field>

  <Field>
    <!-- Second parameter passed to the policy action -->
    <Name>usage</Name>
    <DataType>Integer</DataType>
    <Type>Variable</Type>
    <Length>8</Length>
    <Alignment>Right</Alignment>
    <Padding>Zeros</Padding>
    <Mandatory>Yes</Mandatory>
  </Field>
</Record>

</Log>
</Logs>
```





# 10

## **RADIUS Actions**

- ["RADIUS Actions" on page 82](#)
- ["RADIUS Server Action" on page 82](#)
- ["RADIUS Client Action" on page 82](#)

## 10.1 RADIUS Actions

SandScript provides actions for both RADIUS servers and clients.

## 10.2 RADIUS Server Action

Sends a RADIUS reply to the specified RADIUS client. Usually the reply is sent as an answer for arrived RADIUS request.

The return code is true if the request was sent; false otherwise.

If 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 must be surrounded by single quotes.

```
RADIUS.<CommandName>(Peer-IP-Address, Peer-IP-Port, \
  Request-Authenticator, Id, {AVPName, ...})
```

Where:

|                              |  |
|------------------------------|--|
| <b>CommandName</b>           | is the RADIUS command to send.   |
| <b>Peer-IP-Address</b>       | is the IPv4 client address to reply to. Usually this value is obtained from original RADIUS request. |
| <b>Request-Authenticator</b> | is the RADIUS request authenticator. Usually this value obtained from original RADIUS request.       |
| <b>Id</b>                    | is the RADIUS request ID. Usually this value obtained from original RADIUS request.                  |
| <b>AVPName</b>               | is an optional AVP name from the dictionary.   |



### Example:

For example:

```
if <condition> then set rc=RADIUS.'Access-Accept'(\
  'Peer-IP-Address': RADIUS.'Access-Request'.Peer-IP-Address, \
  'Peer-IP-Port': RADIUS.'Access-Request'.Peer-IP-Port, \
  'Id': RADIUS.'Access-Request'.Id, \
  'Request-Authenticator': RADIUS.'Access-Request'.Auth, \
  'Framed-IP-Address': StringToIP("10.10.10.1"))
```

## 10.3 RADIUS Client Action

This action sends a RADIUS command to the specified RADIUS server.

The return code is true if the request was sent; false otherwise.

If 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 must be surrounded by single quotes.

```
RADIUS.<CommandName>(Peer, {AVPName, ...})
```

Where:

|                    |   |
|--------------------|---|
| <b>CommandName</b> | is the RADIUS command to send.                                |
| <b>Peer</b>        | is the name of configured RADIUS node to send the command to. |

**AVPName** is an optional AVP name from the dictionary.



**Example:**

For example:

```
if <condition> then set rc=RADIUS.'Access-Request( \
'Peer': "myself", \
'User-Name': "bob", \
'User-Password': "bob's password", \
'Framed-IP-Address': StringToIP("10.10.10.1"))
```





# 11

## Provisioning

- ["Provisioning.Session.ProvisionSubscriber\(\)" on page 86](#)
- ["Provisioning.Nat.Map or Unmap" on page 87](#)
- [" Provisioning.Subscriber.Get\(\) " on page 88](#)
- ["Provisioning.Subscriber.Delete" on page 89](#)

## 11.1 Provisioning.Session.ProvisionSubscriber()

Provisions a subscriber with the values of the passed in parameters. Returns an integer to identify the transaction ID of the request; or 0 to indicate failure.

```
Provisioning.Session.ProvisionSubscriber()
```

If only one IP address is used, the form "IPv4.Address" may be used as a synonym for "IPv4[0].Address". The same goes for "IPv6.Address". The index must be a literal integer, not an expression.

Each address must have a matching prefix. If both the address and prefix are NULL the pair is ignored, but if one of the pair is NULL the action fails.

Mandatory arguments are:

|  |   |
|--|---|
| <b>IPv4[i].Address</b>                     | is the subscriber's IPv4 address (ipaddress). A NULL value is treated as if the argument had been omitted.  |
| <b>IPv4[i].Prefix</b>                      | is the IPv4 prefix length (integer). A NULL value is treated as if the argument had been omitted.<br><br>The action will fail if the value is a negative integer, or an integer larger than 32.   |
| <b>IPv4[i].SessionQualifier.SiteNumber</b> | is the IPv4 site number (integer). The site number can be used to disambiguate the sessions when the same IP address belongs to more than a single subscriber within a single deployment (IP address space overlap).  |
| <b>IPv6[i].Address</b>                     | is subscriber's IPv6 address (ipaddress). A NULL value is treated as if the argument had been omitted.  |
| <b>IPv6[i].Prefix</b>                      | is the IPv6 prefix length (integer). A NULL value is treated as if the argument had been omitted.<br><br>The action will fail if the value is a negative integer, or an integer larger than 128.  |
| <b>SubscriberID</b>                        | a string identifying the subscriber. This parameter cannot be an empty string or NULL.  |
| <b>Timestamp</b>                           | is the time of the mapping, in milliseconds. This time is sent in the request to the SPB.   |
| <b>Presence</b>                            | indicates the type of the request, login or logout (integer). Allowed values are: <ul style="list-style-type: none"> <li>• Provision.PRESENCE_LOGIN: Subscriber logged in, or update.</li> <li>• Provision.PRESENCE_LOGOUT: Subscriber logged out.</li> </ul> |



**Note:**

If a value outside of the list is provided, the action will fail.

Optional arguments are:

|                           |   |
|---------------------------|---|
| <b>SessionQualifier</b>   | is a string to identify session. If not specified, the default "", or NULL, is used.  |
| <b>Realm</b>              | is an integer to identify the session realm. If not specified, the default 0, or NULL is used.  |
| <b>ReconcileID</b>        | is an integer indicating the reconcile subsystem ID. If not specified, the default 0, or NULL is used.  |
| <b>RetryOnFailure</b>     | is a boolean type that indicates if session processing should be retried in the case of failure. Defaults to TRUE.  |
| <b>SingleIpAssignment</b> | is a boolean that enables or disables subscriber single IP mode. Single IP mode ensures that the IP assignment unassigns all other IP assignments the subscriber may have. Defaults to FALSE. |
| <b>AcknowledgeRequest</b> | is a boolean indicating that the policy application is expecting an IsComplete event when processing completes. Defaults to TRUE.   |

|                             |   |
|-----------------------------|---|
| <b>TTL</b>                  | is an integer indicating session time to live inside the SDE's cache from the assign time (in seconds). Valid values are: 0 - infinity. Defaults to 900.<br><br>A 0 value indicates that no reconcile event should be generated for this session.   |
| <b>SubscriberAutoCreate</b> | is a boolean indicating if auto create is enabled or disabled. Defaults to NULL.<br><br>By default, the ProvisionSubscriber action requires at least one IP address argument. When SubscriberAutoCreate is given but NULL, no IP address argument is required, which permits the policy author to set attributes for a subscriber without an IP session. If the subscriber does not exist in the SPB and SubscriberAutoCreate is true, then the subscriber will be created in the SPB. If the subscriber does not exist in the SPB and SubscriberAutoCreate is false, then the SPB will reject the request. |
| <b>'OpaqueData[N]'</b>      | is any opaque data (string) kept on the SDE cache that will be used in the response Provision.Session.IsComplete. N can be from 0 to 19.  |

Optional subscriber / session attributes are:

|   |   |
|---|---|
| <b>'Attribute.&lt;name&gt;': &lt;value&gt;</b>        | the name must match the name in the attribute definitions. The type of the attribute must match the declared type of the attribute (see subscriber attribute above). Single quotes are mandatory around the argument name.<br><br>If the SPB is not running in "in memory" mode, session attributes cannot be set, but the mapping/unmapping action will be successful. To determine if in-memory mode is enabled, on the SPB run the <code>show service subscriber-management status</code> CLI command. |
| <b>'Attribute.&lt;name&gt;.Expire': &lt;value&gt;</b> | is the length of the time from now until the expiration occurs. Value must be an integer. Measured in seconds. 0 indicates that the attribute never expires. Default is 0.  |

## 11.2 Provisioning.Nat.Map or Unmap

Maps or unmaps the subscriber's IP address NAT mapping on the SDE. Returns an integer to identify the transaction Id of the request, or 0 to indicate failure.

```
Provisioning.Nat.Map()  
Provisioning.Nat.Unmap()
```

Mandatory parameters are:

|                          |  |
|--------------------------|--|
| <b>privateIP</b>         | The private IP address (ipaddress) to map or unmap.  |
| <b>privatePrefix</b>     | The IPv4 prefix length (integer). A NULL value is treated as if the argument had been omitted. The action will fail if the value is a negative integer, or larger than 32 when privateIP is an IPv4 address, or larger than 128 when privateIP is an IPv6 address.                                       |
| <b>privateSiteNumber</b> | The IPv4 site number (integer). The site number can be used to disambiguate the sessions when the same IP address belongs to more than a single subscriber within a single deployment (IP address space overlap). If privateIP is an IPv6 address this argument must not be set, or must be set to NULL. |
| <b>publicIP</b>          | The public IPv4 address (ipaddress) that was allocated. This argument must be set to an IPv4 address.  |
| <b>lowPort</b>           | The low port block number (integer) that was allocated. A NULL value is treated as if the argument was set to 0.   |
| <b>highPort</b>          | The high port block number (integer) that was allocated. A NULL value is treated as if the argument was set to 65535.  |
| <b>Timestamp</b>         | The time of the mapping, in milliseconds. This time is sent in the request to the SPB.   |

Optional parameters are:

|                           |   |
|---------------------------|---|
| <b>AcknowledgeRequest</b> | A boolean indicating that the policy application is expecting an IsComplete event when processing completes. Default is FALSE.  |
| <b>RetryOnFailure</b>     | A boolean that indicates if a transaction should be retried in the case of failure. Default is TRUE.  |
| <b>'OpaqueData[N]'</b>    | Any opaque data (string type) kept on the SDE cache to be used in the response Provision.Nat.(Map/Unmap).IsComplete. N can be from 0 to 19.   |
| <b>TTL</b>                | An integer indicating session time to live inside the SDE's cache from the assign time (in seconds). Valid values are: 0 - infinity. Defaults to 900. A 0 value indicates that no reconcile event should be generated for this session. |

## 11.3 Provisioning.Subscriber.Get()

The SandScript action fetches a subscriber profile by name from the SPB Database and returns a transaction ID to the Get request, or returns "0" in case of the Get request failure.

`Provisioning.Subscriber.Get()`

Mandatory arguments are:

**Name** The subscriber name in the SPB table. The data type is string and the default value is set to NULL.

Optional arguments are:

|                       |   |
|-----------------------|---|
| <b>RetryOnFailure</b> | The provisioning subsystem should retry sending the Get request upon failure. The data type is boolean and the default value is set to true.  |
| <b>OpaqueData[N]</b>  | Information (string data type) fetched from the Provision.Subscriber.Get SandScript request and stored on the SDE. This information is later used by the Provision.Subscriber.Get.IsComplete event to execute the Provision.Subscriber.Get operation completely. <ul style="list-style-type: none"> <li>Where N is an integer, with value range from 0 to 19.</li> <li>Opaque data type is string.</li> <li>Default value is true.</li> </ul> |

Sample SandScript:



```
# set an attribute for a subscriber
attribute "Tier" type string

PolicyGroup expr(Policy.Reloaded) all
{
    if true then \
        Provision.Session.ProvisionSubscriber(Presence: Provision.PRESENCE_LOGIN , Timestamp : NowMs , \
                                                Name: "JohnDoe", SubscriberAutoCreate: true, 'Attribute.Tier':
"Gold")
}

# setting the attribute was successful, no we look it up
PolicyGroup expr(Provision.Session.IsComplete) all
{
    if true then \
        Provision.Subscriber.Get(Name: "JohnDoe", OpaqueData: "tr1234")
}

# log the lookup answer, should be: 'Profile', 'JohnDoe', 'tr1234', 'hello'
PolicyGroup expr(Provision.Subscriber.Get.IsComplete) all
{
    if true then \
        log values ("Provision.Subscriber.Get.IsComplete received") and \
        log values ("Subscriber Name: ", Provision.Subscriber.Get.Name, "Opaque
Data:", Provision.Subscriber.Get.OpaqueData) and \
        log values ("Subscriber Attr Tier:", Provision.Subscriber.Get.Attribute.Tier) and \
        log values ("Subscriber TID:", Provision.Subscriber.Get.TransactionID) and \
        log values ("State:", Provision.Subscriber.Get.State)

        if expr(Provision.Subscriber.Get.State = Provision.STATE_GET_SUCCESS) then log values
("Provision.Provision.STATE_GET_SUCCESS")
        if expr(Provision.Subscriber.Get.State = Provision.STATE_GET_SUBSCRIBER_NOT_FOUND) then log values
("Provision.Provision.Provision.STATE_GET_SUBSCRIBER_NOT_FOUND")
        if expr(Provision.Subscriber.Get.State = Provision.STATE_GET_PROCESSING_FAILED) then log values
("Provision.STATE_GET_PROCESSING_FAILED")
}
```

Optional subscriber or session attributes:

**Attribute.<name>** The attribute that defines a subscriber. The name must match the name in the attribute definitions.  
The type of the attribute must match the declared type of the attribute.  
For example: Attribute.Tier: "Gold"

## 11.4 Provisioning.Subscriber.Delete

Provisioning.Subscriber.Delete, deletes subscriber by name from the SPB.

For example:

```
if expr(Provision.Subscriber.Get.State = Provision.STATE_OPERATION_PROCESSING_FAILED) \
then log values ("Provision.STATE_OPERATION_PROCESSING_FAILED")
```

**IsComplete** Returns an integer to identify the transaction ID of the request. Returning 0 indicates failure. The data type is boolean.

**TransactionID** The transaction ID issued by the SPB for the Get request. The data type is integer.

**Name** Indicates the state of the Get request. The data type is integer. Possible values are:

- Provision.STATE\_GET\_SUCCESS
- Provision.STATE\_GET\_SUBSCRIBER\_NOT\_FOUND
- Provision.STATE\_GET\_PROCESSING\_FAILED

**OpaqueData [N]** Any opaque data that was set by the Provisioning.Subscriber.Delete action. N can be integer expression. The value of N need to be from 0 to 19. The data type is string.





# 12

## ID Allocation

The ID allocation subsystem allows the SDE to communicate with the SPB to map a given identifier to a value which is used uniquely across the elements in the Sandvine deployment. The SDE is capable of creating, retrieving, and deleting these values from the SPB where they are stored. Some SDE applications require this functionality for centralized mapping between the names and identifiers, where the names and identifiers are unique.

The ID allocation subsystem provides a SandScript API that performs:

- A query to retrieve an identifier by name.
- A query to retrieve a name by identifier.
- Identifier generation—When querying a name that does not exist, the SDE communicates to the SPB to generate an identifier for that name and store it.
- Identifier to name map deletion—Deletes the map based on the identifier or the name.

See the *SDE SandScript Reference Guide* for more information on ID Allocation SandScript policies.

• ["IdAllocation.Lookup \(\) " on page 92](#)

• ["IdAllocation.Delete\(\) " on page 93](#)

## 12.1 IdAllocation.Lookup ()

The IdAllocation.Lookup() operation performs lookup in the SPB table for bulk request either by name or ID. The lookup is done using name or ID, if the name or ID exist in the SPB table, then the SPB returns name and ID for the lookup query.

If a lookup is done using a name and the name does not exist in the SPB table, then the SPB inserts this name and a unique ID to the name.

In case the lookup is queried using ID and the ID does not exist then the lookup request fails.

Sample SandScript:

```
PolicyGroup expr(Policy.Reloaded) all
{
    integer "transactionID"
    if true then set transactionID = IdAllocation.Lookup( \
        'Name[2]':"Account", \
        'Name[3]':"Member", \
        'Name[7]':"AnotherName", \
        'Id':123, \
        'Id[4]':456, \
        'Id[5]':789, \
        'AutoCreate':false, \
        'RetryOnFailure':false)
    if true then log values("Lookup request sent. TransactionID:", transactionID)
}

PolicyGroup expr(IdAllocation.Lookup.IsComplete) all
{
    if true then log values("IdAllocation.TransactionID", IdAllocation.TransactionID)
    if true then log values("IdAllocation.AutoCreate ", IdAllocation.AutoCreate)
    if true then log values("IdAllocation.Name[0]", IdAllocation.Name[0])
    if true then log values("IdAllocation.Id[0]", IdAllocation.Id[0])
    if true then log values("IdAllocation.Name[0].Result", IdAllocation.Name[0].Result)
    if true then log values("IdAllocation.Id[0].Result", IdAllocation.Id[0].Result)
    if true then log values("IdAllocation.Name.Created", IdAllocation.Name.Created)
    if true then log values("IdAllocation.Id.Created", IdAllocation.Id.Created)
    if true then log values("IdAllocation.State", IdAllocation.State)
}
```

Mandatory arguments are:

**Name[N]** The Name[N] is the name in the SPB table. The data type is string, with value range for [N] is from 0 to 9. The maximum length of a name is 1024 characters.

**Id[N]** The Id[N] is the identifying ID in the SPB table. The data is integer, with value range for [N] is from 0 to 9.



**Note:**

- At least one name or ID must be defined, where NULL names are ignored.
- The ID(s)' and Name(s)' indices cannot collide, e.g. if Name[0] is defined in the lookup request, Id[0] cannot be defined.
- The action will return an integer identifying the transaction ID of the request, when 0 indicates failure.

Optional arguments are:

**RetryOnFailure** The ID allocation subsystem should retry sending the lookup request upon failure. The data type is boolean and the default value is set to true.

**AutoCreate** If the lookup is queried by name and if that name does not exist in the SPB table, then the SPB adds this name (must be unique in the SPB table) and assigns a unique ID to that name. The data type is boolean and the default value is set to true.

**OpaqueData[N]** Any opaque data (string type) kept on SDE cache that is used in the corresponding IsComplete event.

- Where N is an integer, with value range from 0 to 19.
- Opaque data type is string.
- Default value is NULL.

## 12.2 IdAllocation.Delete()

The IdAllocation.Delete() operation deletes SPB table entries. A table entry contains a name and an ID. The action deletes the corresponding entry to the given ID or name.

Sample SandScript:

```
PolicyGroup expr(Policy.Reloaded) all
{
    integer "transactionID"
    # The invocation itself
    if true then set transactionID = IdAllocation.Delete('Id[3]':"HelloWorld")
    # Print to the log the transactionID we have just recieved. The transaction ID
    # will be used as correlation between the operation and the response.
    if true then log values("Lookup request sent. TransactionID:", transactionID)
}

PolicyGroup expr(IdAllocation.Delete.IsComplete) all
{
    if true then log values("IdAllocation.TransactionID", IdAllocation.TransactionID)
    if true then log values("IdAllocation.Name[3]", IdAllocation.Name[0])
    if true then log values("IdAllocation.Id[0]", IdAllocation.Id[0])
    if true then log values("IdAllocation.Name[0].Result", IdAllocation.Name[0].Result)
    if true then log values("IdAllocation.Id[0].Result", IdAllocation.Id[0].Result)
    if true then log values("IdAllocation.Name.Created", IdAllocation.Name.Created)
    if true then log values("IdAllocation.Id.Created", IdAllocation.Id.Created)
    if true then log values("IdAllocation.State", IdAllocation.State)
}
```

Mandatory arguments are:

- Name[N]** The Name[N] is the name in the SPB table. The data type is string, with value range for [N] is from 0 to 9. The maximum length of a name is 1024 characters.
- Id[N]** The Id[N] is the identifying ID in the SPB table. The data is integer, with value range for [N] is from 0 to 9.



**Note:**

- At least one name or ID must be defined, where NULL names are ignored.
- The ID(s)' and Name(s)' indices cannot collide, e.g. if Name[0] is defined in the lookup request, Id[0] cannot be defined.
- The action will return an integer identifying the transaction ID of the request, when 0 indicates failure.

Optional arguments are:

- RetryOnFailure** The ID allocation subsystem should retry sending the lookup request upon failure. The data type is boolean and the default value is set to true.
- OpaqueData[N]** Any opaque data (string type) kept on SDE cache that is used in the corresponding IsComplete event.
- Where N is an integer, with value range from 0 to 19.
  - Opaque data type is string.
  - Default value is NULL.





# 13

## LDAP

This section explains about the SandScripts applicable for the LDAP subsystem.

- ["Ldap.GetAttributes \(\) " on page 96](#)
- ["Ldap.Get\(\) " on page 96](#)
- ["Ldap.Search\(\) " on page 97](#)

## 13.1 Ldap.GetAttributes ()

The Ldap.GetAttributes () operations retrieves list of objects matching to a query attribute or value. This SandScript also obtains Distinguished Name (DN) and non-unique attribute or value and lists all the matching values for the requested list of attributes.

Sample SandScript:

```
Ldap.GetAttributes (Name:"LdapServer", DN:"DC=sandvine,DC=com", Key: "cn", Value: "Sandvine" , Attributes:
"*)
if expr (Ldap.GetAttributesResponse.ResultCode = 0) then \
log values (Ldap.GetAttributesResponse.ResultCode ) and \
log values (Ldap.GetAttributesResponse.dn ) and \
log values (Ldap.GetAttributesResponse.mail )

if expr (Ldap.GetAttributesResponse.ResultCode != 0) then \
log values (Ldap.GetAttributesResponse.ResultCode)
```

Arguments are:

|                   |  |
|-------------------|--|
| <b>Name</b>       | The LDAP server name.  |
| <b>Key</b>        | It is a filter condition, like, Given name, Common name, and Application.    |
| <b>Value</b>      | Information provided for a Key is the value.                                 |
| <b>Attributes</b> | It is an identity like company, department, mail, userprincipalname, and cn. |

## 13.2 Ldap.Get()

The Ldap.Get() operation retrieves one or many attributes or values for a single object based on a unique identifier.

Sample SandScript:

```
Ldap.Get (Name:"LdapServer", DN:"DC=sandvine,DC=com", Key: "cn", Value: "Sandvine" , Attributes: "*)
if expr (Ldap.GetResponse.ResultCode != 0 && Ldap.GetResponse.ResultCode != 4) then \
log values (Ldap.GetResponse.ResultCode ) and \
log values (Ldap.GetResponse.dn and \
log values (Ldap.GetResponse.mail )

if expr (Ldap.GetResponse.ResultCode = 0 || Ldap.GetResponse.ResultCode = 4) then \
log values (Ldap.GetResponse.ResultCode )

Looping for all responses using for each:
PolicyGroup expr (Ldap.GetResponse.ResultCode = 0) all
{
    foreach "my_it" in Ldap.GetResponse
    {
        if true then \
        log values (my_it.dn ) and \
        log values (my_it.mail)
    }
}
```

|                   |   |
|-------------------|---|
| <b>Name</b>       | The LDAP server name.   |
| <b>DN</b>         | Represents Distinguished Name, like an organization or a domain.              |
| <b>Key</b>        | It is a filter condition, like, Given name, Common name, and Application.     |
| <b>Value</b>      | Information provided for a Key is the value.                                  |
| <b>Attributes</b> | It is an identity like, company, department, mail, userprincipalname, and cn. |



## 13.3 Ldap.Search()

The Ldap.Search() operation performs a complex search on LDAP Server based on filters conditions to obtain the list of attributes or values. This function obtains Distinguished Name (DN) and non-unique attribute or value and lists all the matching values requested.

Sample SandScript:

```
Ldap.Search(Name:"LdapServer", DN:"DC=sandvine,DC=com", Filter: "(&(cn=Sandvine)(givenName=Sandvine)",  
Attributes: "**")  
if expr(Ldap.SearchResponse.ResultCode = 0) then \  
log values (Ldap.SearchResponse.ResultCode ) and \  
log values (Ldap.SearchResponse.dn )and \  
log values (Ldap.SearchResponse.mail)  
if expr(Ldap.SearchResponse.ResultCode != 0) then \  
log values (Ldap.SearchResponse.ResultCode)
```

```
Looping for all responses using for each:  
PolicyGroup expr(Ldap.SearchResponse.ResultCode = 0) all  
{  
    foreach "my_it" in Ldap.SearchResponse  
    {  
        if true then \  
        log values (my_it.dn ) and \  
        log values (my_it.mail)  
    }  
}
```

|                   |   |
|-------------------|---|
| <b>Name</b>       | The LDAP server name.   |
| <b>DN</b>         | Represents Distinguished Name, like an organization or a domain.              |
| <b>Filter</b>     | Represents a search criteria.   |
| <b>Attributes</b> | It is an identity like, company, department, mail, userprincipalname, and cn. |





# 14

## External Commands

- ["External Commands" on page 100](#)
- ["External Command Syntax" on page 100](#)

## 14.1 External Commands

The External Command subsystem allows policy to execute an external command on the system, such as a Executable and Linkable Format (ELF) or a script, so you can use third party utilities.

The commands are executed with the user-id of "sv" user.

If the options to an external command are changed and policy is reloaded, existing outstanding commands are treated in this manner:

- If the `max_concurrent` parameter is changed, but other parameters are the same, the system treats it as if it were the same command and the new value of `max_concurrent` is applied immediately on reload success.  
Increasing `max_concurrent` allows queued commands (if any) to be executed subject to the new limit. Decreasing `max_concurrent` does not impact the commands already invoked, but is applied to the queued and new commands. PDB statistics are not reset for this `CmdName`.
- If the `cmd_path` and/or output mode parameters are changed, it is treated as a new command. The commands that have been executed or queued before the reload will continue to be invoked. However, their `OUTPUT`, `ERROROUTPUT` and `END` events will not be sent to policy.

## 14.2 External Command Syntax

Invokes an external command, such as an ELF or script.

```
externalcommand "name" command "cmd_path" \  
  max_concurrent <concurrent_count> \  
  {output none|line}
```

Where:

|                         |   |
|-------------------------|---|
| <b>"name"</b>           | is the name of the external command. The name is used to identify the external command in the CLI and when referring to the external command using SandScript expressions. The name is case insensitive and must be unique.   |
| <b>"cmd_path"</b>       | is the external command with an absolute or a relative path.  |
| <b>concurrent_count</b> | is the number of outstanding commands allowed for the external command name. The value should be non zero positive integer. This limit is enforced per SDE.   |
| <b>output</b>           | <p>is the output mode to use. The output modes are:</p> <ul style="list-style-type: none"><li>• none - no pipe is opened to the child process. The output of external command is not passed to SandScript. This is the default value.</li><li>• line - the output of the external command is passed to SandScript as and when it is received and is split by one newline. The newline character is not passed to SandScript.</li></ul> <p>The output types <code>stdout/stderr</code> are sent to SandScript in separate events. The ordering between <code>stdout</code> and <code>stderr</code> cannot be guaranteed.</p> |



# 15

## Troubleshooting SandScript

- ["Tips for Using Unique By" on page 102](#)
- ["Message Logs" on page 102](#)
- ["Informative CLI Commands" on page 102](#)
- ["Error Handling for LDAP Subsystem when Input Queue is Full " on page 102](#)
- ["Error Handling for LDAP Subsystem when Output Queue is Full " on page 103](#)

## 15.1 Tips for Using Unique By

If your policy is over counting, over shaping or over limiting (doing too much of any of these actions), then it may be that one or more of your unique by clauses are not unique.

Using shapers, limiters or measurements that are unique by either a string field or by multiple expression fields may result in incorrect behavior. In such cases, different unique by instances can collide and result in incorrect values being returned by policy. The uniqueness of a unique by clause is guaranteed when the clause contains:

- a single integer expression.
- a single boolean expression.
- a subscriber expression and a classifier expression.
- a protocol expression and one or two classifier expressions.
- one to four classifier expressions.

If a unique by clause is used in policy that does not follow the above rules that guarantee uniqueness, then a warning will be raised when the policy is reloaded.

## 15.2 Message Logs

Most error messages are sent to syslog.

The default location for is `/var/log/svlog`. For more information on log messages, refer to the *PTS Operations Reference Guide*.

## 15.3 Informative CLI Commands

Sandvine provides system information through CLI commands.

| CLI command                        | Description   |
|------------------------------------|---|
| <code>show system overview</code>  | Shows an overview of what the PTS or SDE is doing.  |
| <code>show system resources</code> | Shows a list of system resources for the system, a specific resource ID, controller or module. System resources include hard disk space, memory, flows, and other resources that, if exhausted, will impact the proper functioning of the system. |

## 15.4 Error Handling for LDAP Subsystem when Input Queue is Full

When a LDAP subsystem input queue is full with SandScript requests and it cannot accept any new SandScript requests then this error condition occurs. In this case LDAP subsystem returns a "0" indicating the LDAP subsystem input queue is full. SandScript author can check for this transaction ID value "0" before sending a new request to the LDAP sub-system



**Note:**

The default input queue size is 100,000 messages each.

```
table "LdapTiD_table" (integer "LdapTiD", string "serverName", string "SubscriberName") timeout none unique
by (Table.LdapTiD_table.LdapTiD)
PolicyGroup all
{
    string "SubscriberName"
    string "serverName"
    integer "LdapTiD"
    if true then \
        set SubscriberName = "*" and \
        set serverName = "myLdap1" and \
        set LdapTiD = Ldap.Get(Name:serverName , Key: "cn", Value:SubscriberName, DN:
"DC=sandvine,DC=com", Attributes: "**")
        PolicyGroup
        {
            if expr(LdapTiD = 0) then \
                log values("message dropped LdapTiD=",LdapTiD)
            if true then \
                set Table.LdapTiD_table[LdapTiD].serverName = serverName and \
                set Table.LdapTiD_table[LdapTiD].SubscriberName = SubscriberName and \
                log values("LdapTiD=",LdapTiD) and \
                set LdapTiD = Ldap.Get(Name:serverName , Key: "cn", Value:SubscriberName, DN:
"DC=sandvine,DC=com", Attributes: "**") and \
                set Table.LdapTiD_table[LdapTiD].serverName = serverName and \
                set Table.LdapTiD_table[LdapTiD].SubscriberName = SubscriberName and \
                log values("LdapTiD=",LdapTiD)
        }
    }
}
```

#### Troubleshooting information:

If the Input queue is full all the SandScript requests are dropped by the LDAP subsystem

1. To check current queue usage:

```
show service ldap-client stats
```

2. Run this CLI command to check the current configuration for Input queue size:

```
show config service ldap-client in-queue-size
```

3. Run this CLI command to change the input queue size:

```
configure
```

4. Commit the changes made to the input queue:

```
commit
```

## 15.5 Error Handling for LDAP Subsystem when Output Queue is Full

When a LDAP subsystem sends a SandScript request to the LDAP server successfully, obtains a response from the LDAP server, and then the LDAP subsystem tries to put this SandScript response to output queue, but by then the output queue is full. In this case the LDAP subsystem issues an error “message dropped” with a result code 116 to SandScript.



**Note:**

The default output queue size is 100,000 messages each.

```
PolicyGroup expr(Ldap.GetResponse.ResultCode >= 0 ) all
{
    string "SubscriberName"
    string "plan"
    string "category"
    PolicyGroup
    {
        if expr( Ldap.GetResponse.ResultCode = 116 ) then \
            log values ("message with trx id" , Ldap.GetResponse.TransactionId, "dropped
due to queue full")
        if true then \
            set SubscriberName = String(Ldap.GetResponse[0].cn) and \
            set plan = String(Ldap.GetResponse[0].sn) and \
            set category = String(Ldap.GetResponse[0].givenName) and \
            log values ("Ldap.GetResponse.ResultCode=", Ldap.GetResponse.ResultCode)
and \
            log values ("Ldap.GetResponse.TransactionId=", Ldap.GetResponse.TransactionId)
and \
            log values
("Ldap.GetResponse.ResultDescription=", Ldap.GetResponse.ResultDescription) and \
            log values
("Table.LdapTiD_table[Ldap.GetResponse.TransactionId].serverName", Table.LdapTiD_table[Ldap.GetResponse.TransactionId].serverName)
and \
            log values
("Table.LdapTiD_table[Ldap.GetResponse.TransactionId].SubscriberName", Table.LdapTiD_table[Ldap.GetResponse.TransactionId].SubscriberName)
    }
}
```

#### Troubleshooting information:

**1. To check current output queue usage:**

```
show service ldap-client stats
```

**2. Run this CLI command to check the default output queue size:**

```
show config service ldap-client out-queue-size
```

**3. Run this CLI command to change the output queue size:**


```
configure
set config service ldap-client out-queue-size <value>
```

**4. Commit the changes made to the input queue:**

```
commit
```







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)