# logsnarf Documentation

***Release 0.0.1***

**David MacKinnon**

**Jun 28, 2023**

# CONTENTS

Contents:

CHAPTER

ONE

# CONFIGURING LOGSNARF

## 1.1 The Logsnarf configuration file

**Contents**



### 1.1.1 General

Logsnarf looks in XDG_CONFIG_DIRS for configuration files by default, named {RESOURCE_NAME}.ini with a default resource name of 'logsnarf'. So a user configuration will, by default be (on linux) ~/.config/logsnarf/logsnarf.ini

The same directories are searched for a logging.ini which should be a `logging.config.fileConfig()` configuration. If no file is found, `logging.basicConfig()` is called with no arguments.

Within the logsnarf config file there must be a logsnarf section with an apps entry. This should contain a list of sections where the various apps are configured.

An app is a combination of directories watched, and bigquery upload information. If you're configuring multiple apps, you can also use the [DEFAULT] section to provide defaults for these (e.g. upload credentials)

### 1.1.2 Example

An example configuration file:

```
[DEFAULT]
project_number=<your google project number>
service_email=<service account email address>
dataset=logging
keyfile=key.p12
default_domain=my.domain
max_buffer=2000
batchsize=400

[logsnarf]
apps=app1, app2
threadpool_size=20

[app1]
table_name_schema=syslog_{YEAR}{MONTH}{DAY}
directories=["/var/log/syslog/hosts"]
recursive=True
pattern=(syslog|.*\.log$)
schema=syslog_schema.json

[app2]
table_name_fmt=apache_{YEAR}{MONTH}{DAY}
directories=["/var/log/apache"]
recursive=False
pattern=(access|error)\.log$
```

In this example `schema_file` for app2 would be `app2_schema.json`. The state files would be `app1_state.json` and `app2_state.json` (using the defaults and ConfigParser interpolation.)

### 1.1.3 Configuration file fields

Where `%(value)s` is listed in a default, it uses `ConfigParser.SafeConfigParser` interpolation. `__name__` in this case refers to the section name.

#### logsnarf section

**apps**
: list of sections with app configurations

**threadpool_size**
: **default value: 30** size to set the twisted threadpool. Logsnarf does most uploads and some other table operations in threads.

### App sections

Fields in an app section are

### BigQuery-Service related

**dataset**
Dataset to upload to

**keyfile**
Path to a file with Service account key.

**project_number**
Your BigQuery project number

**schema_file**
**default value: %(__name__)s_schema.json**

Filename for a file with a json representation of the BigQuery fields This is loaded from the xdg user config directory.

**service email**
Service account email address

### Logfile related

**default_tz**
**default value: UTC** The default timezone to apply if none is available in the data.

**directories**
a JSON list of directories to watch for log files

**pattern**
**default value:** `.*\.log` regexp pattern that files must match to be watched.

**state_file**
**default value: %(app_section)s_state.json)** state file to store logfile names, inode and last seek position. This will be created if it doesn't exist.

### BigQuery uploader related

**batchsize**
**default value: 250** how many log entries to upload in a single request. max 500

**flush_interval**
**default value: 30** Normally the uploader will wait until batchsize log entries are queued before starting an upload, however it will wait at most flush_interval seconds.

**max_buffer**
**default value: 1000** how many log entries the uploader should buffer from the log watcher before pausing the log watcher. You will mainly hit this backprocessing files, while every effort is made to flush this buffer on exit,

Lines kept here are at risk of loss, since they are marked "read" by the log watcher when pushed to the uploader. Too low, and you'll be waiting on BigQuery inserts.

**table_name_fmt**
**default value: logs_{YEAR}{MONTH}{DAY}** When creating tables, this is used for naming, if the entries don't contain a 'table' field. The schema can include {YEAR} {MONTH} and {DATE} substitutions, that are taken from a time field in the data, or now if that field doesn't exist.

### Other

**default_domain**
used by additional verifiers, to insert a domain on non-qualified hostnames in 'host', 'src.host', or 'dst.host' fields

## 1.2 Schema File

This file matches the schema.fields <https://cloud.google.com/bigquery/docs/reference/rest/v2/tables#TableFieldSchema> attribute listed in the Google BigQuery schema documentation. It should be a list of fields, as described by the API docs. An example, based very loosely on the CEE/Lumberjack schema follows.

```
[
    {
      "mode": "REQUIRED",
      "type": "STRING",
      "name": "msg"
    },
    {
      "mode": "REQUIRED",
      "type": "STRING",
      "name": "host"
    },
    {
      "mode": "REQUIRED",
      "type": "TIMESTAMP",
      "name": "timereported"
    },
    {
      "mode": "REQUIRED",
      "type": "TIMESTAMP",
      "name": "time"
    },
    {
      "type": "STRING",
      "name": "pname"
    },
    {
      "type": "INTEGER",
      "name": "pid"
    },
    {
      "type": "STRING",
      "name": "sev"
    },
    {
```

(continues on next page)

(continued from previous page)

```
      "type": "STRING",
      "name": "service"
    },
    {
      "type": "RECORD",
      "name": "syslog",
      "fields": [
        {
          "type": "STRING",
          "name": "fac"
        },
        {
          "type": "STRING",
          "name": "pri"
        }
      ]
    },
    {
      "type": "RECORD",
      "name": "action",
      "fields": [
        {
          "type": "STRING",
          "name": "method"
        },
        {
          "type": "STRING",
          "name": "status"
        },
        {
          "type": "STRING",
          "name": "type"
        }
      ]
    },
    {
      "type": "RECORD",
      "name": "src",
      "fields": [
        {
          "type": "STRING",
          "name": "ifname"
        },
        {
          "type": "STRING",
          "name": "hwaddr"
        },
        {
          "type": "STRING",
          "name": "host"
        },
        {
```

(continues on next page)

(continued from previous page)

```
          "type": "STRING",
          "name": "ipv4"
        },
        {
          "type": "STRING",
          "name": "ipv6"
        },
        {
          "type": "INTEGER",
          "name": "port"
        }
      ]
    },
    {
      "type": "RECORD",
      "name": "dst",
      "fields": [
        {
          "type": "STRING",
          "name": "ifname"
        },
        {
          "type": "STRING",
          "name": "hwaddr"
        },
        {
          "type": "STRING",
          "name": "host"
        },
        {
          "type": "STRING",
          "name": "ipv4"
        },
        {
          "type": "STRING",
          "name": "ipv6"
        },
        {
          "type": "INTEGER",
          "name": "port"
        }
      ]
    },
    {
      "type": "RECORD",
      "name": "user",
      "fields": [
        {
          "type": "STRING",
          "name": "name"
        },
        {
```

(continues on next page)

(continued from previous page)

```
          "type": "INTEGER",
          "name": "id"
        },
        {
          "type": "INTEGER",
          "name": "eid"
        },
        {
          "type": "STRING",
          "name": "euser"
        },
        {
          "type": "STRING",
          "name": "domain"
        }
      ]
    }
]
```

## 1.3 RSYSLOG

format more easily used by logsnarf, and BigQuery. No claims are made regarding the optimality of these configurations, but should serve as a starting point for those interested.

### 1.3.1 mm_normalize

This performs some feature extraction on ssh and dhcp logs. Populating the additional fields and subrecords.

```
# Rules for dhcpd
rule=: %action!method:word% from %src!hwaddr:word% via %src!ifname:word%
rule=: %action!method:word% on %src!ipv4:word% to %src!hwaddr:word% (%src!host:char-to:)
↪%)via %src!ifname:word%
rule=: %action!method:word% on %src!ipv4:word% to %src!hwaddr:word% via %src!ifname:word%
rule=: %action!method:word% for %src!ipv4:word% (%dst!ipv4:ipv4%) from %src!hwaddr:word%␣
↪(%src!host:char-to:)%)  via %src!ifname:word%
rule=: %action!method:word% for %src!ipv4:word% (%dst!ipv4:ipv4%) from %src!hwaddr:word%␣
↪ via %src!ifname:word%
rule=: %action!method:word% for %src!ipv4:word% from %src!hwaddr:word% (%src!host:char-
↪to:)%)  via %src!ifname:word%
rule=: %action!method:word% for %src!ipv4:word% from %src!hwaddr:word%  via %src!
↪ifname:word%
rule=: %action!method:word% of %src!ipv4:word% from %src!hwaddr:word% (%src!host:char-
↪to:)%) via %src!ifname:word% %-:rest%
rule=: %action!method:word% of %src!ipv4:word% from %src!hwaddr:word% via %src!
↪ifname:word% %-:rest%

# Rules for dhcpclient
rule=: %action!method:word% on %src!ifname:word% to %dst!ipv4:ipv4% port %dst!port:number
↪%%-:rest%
```

(continues on next page)

(continued from previous page)

```
rule:= %action!method:word for %src!ipv4:ipv4% from %dst!hwaddr:word% via %src!
↪ifname:word%
rule:= %action!method:word on %src!ipv4:ipv4% to %src!hwaddr:word% via %src!ifname:word%

# Rules for sshd
rule=auth,success: Accepted %action!method:word% for %user!name:char-to:@%@%user!
↪domain:word% from %src!ipv4:ipv4% port %src!port:number% %-:rest%
rule=auth,success: Accepted %action!method:word% for %user!name:char-to:@%@%user!
↪domain:word% from %src!ipv6:word% port %src!port:number% %-:rest%
rule=auth,success: Accepted %action!method:word% for %user!name:word% from %src!ipv4:ipv4
↪% port %src!port:number% %-:rest%
rule=auth,success: Accepted %action!method:word% for %user!name:word% from %src!ipv6:word
↪% port %src!port:number% %-:rest%
rule=auth,failure: Failed %action!method:word% for %user!name:char-to:@%@%user!
↪domain:word% from %src!ipv4:ipv4% port %src!port:number% %-:rest%
rule=auth,failure: Failed %action!method:word% for %user!name:char-to:@%@%user!
↪domain:word% from %src!ipv6:word% port %src!port:number% %-:rest%
rule=auth,failure: Failed %action!method:word% for %user!name:word% from %src!ipv4:ipv4%␣
↪port %src!port:number% %-:rest%
rule=auth,failure: Failed %action!method:word% for %user!name:word% from %src!ipv6:word%␣
↪port %src!port:number% %-:rest%

# Rules for pam
rule=auth,success: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪user %user!name:word% authenticated as %user!name:char-to:@%@%user!domain:word%
rule=auth,success: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪user %user!name:word% authenticated as %user!name:word%
rule=: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%): session
↪%action!status:word% for user %user!euser:word% by %user!name:char-to:(%(uid=%user!
↪id:number%)
rule=: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%): session
↪%action!status:word% for user %user!name:char-to:@%@%user!domain:word% %user!name:word
↪% by (uid=%user!id:number%)
rule=: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%): session
↪%action!status:word% for user %user!name:char-to:@%@%user!domain:word% %user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid=%user!id:number% euid=%user!
↪eid:number% tty=%-:word% ruser=%user!name:word% rhost=%src!ipv4:word% user=%user!
↪name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid=%user!id:number% euid=%user!
↪eid:number% tty=%-:word% ruser=%user!name:word% rhost=%src!ipv4:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid=%user!id:number% euid=%user!
↪eid:number% tty=%-:word% ruser=%user!name:word% rhost= user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid=%user!id:number% euid=%user!
↪eid:number% tty=%-:word% ruser=%user!name:word% rhost=
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid=%user!id:number% euid=%user!
↪eid:number% tty=%-:word% ruser= rhost=%src!ipv4:word% user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
```

(continues on next page)

(continued from previous page)

```
↪authentication failure; logname=%user!name:word% uid=%user!id:number% euid=%user!
↪eid:number% tty=%-:word% ruser= rhost=%src!ipv4:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid=%user!id:number% euid=%user!
↪eid:number% tty=%-:word% ruser= rhost= user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid=%user!id:number% euid=%user!
↪eid:number% tty=%-:word% ruser= rhost=
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid= euid=%user!eid:number% tty=%-
↪:word% ruser=%user!name:word% rhost=%src!ipv4:word% user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid= euid=%user!eid:number% tty=%-
↪:word% ruser=%user!name:word% rhost=%src!ipv4:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid= euid=%user!eid:number% tty=%-
↪:word% ruser=%user!name:word% rhost= user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid= euid=%user!eid:number% tty=%-
↪:word% ruser=%user!name:word% rhost=
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid= euid=%user!eid:number% tty=%-
↪:word% ruser= rhost=%src!ipv4:word% user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid= euid=%user!eid:number% tty=%-
↪:word% ruser= rhost=%src!ipv4:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid= euid=%user!eid:number% tty=%-
↪:word% ruser= rhost= user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname=%user!name:word% uid= euid=%user!eid:number% tty=%-
↪:word% ruser= rhost=
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid=%user!id:number% euid=%user!eid:number% tty=%-
↪:word% ruser=%user!name:word% rhost=%src!ipv4:word% user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid=%user!id:number% euid=%user!eid:number% tty=%-
↪:word% ruser=%user!name:word% rhost=%src!ipv4:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid=%user!id:number% euid=%user!eid:number% tty=%-
↪:word% ruser=%user!name:word% rhost= user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid=%user!id:number% euid=%user!eid:number% tty=%-
↪:word% ruser=%user!name:word% rhost=
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid=%user!id:number% euid=%user!eid:number% tty=%-
↪:word% ruser= rhost=%src!ipv4:word% user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid=%user!id:number% euid=%user!eid:number% tty=%-
↪:word% ruser= rhost=%src!ipv4:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid=%user!id:number% euid=%user!eid:number% tty=%-
```

(continues on next page)

(continued from previous page)

```
↪:word% ruser= rhost= user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid=%user!id:number% euid=%user!eid:number% tty=%-
↪:word% ruser= rhost=
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid= euid=%user!eid:number% tty=%-:word% ruser=%user!
↪name:word% rhost=%src!ipv4:word% user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid= euid=%user!eid:number% tty=%-:word% ruser=%user!
↪name:word% rhost=%src!ipv4:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid= euid=%user!eid:number% tty=%-:word% ruser=%user!
↪name:word% rhost= user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid= euid=%user!eid:number% tty=%-:word% ruser=%user!
↪name:word% rhost=
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid= euid=%user!eid:number% tty=%-:word% ruser= rhost=
↪%src!ipv4:word% user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid= euid=%user!eid:number% tty=%-:word% ruser= rhost=
↪%src!ipv4:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid= euid=%user!eid:number% tty=%-:word% ruser=␣
↪rhost= user=%user!name:word%
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪authentication failure; logname= uid= euid=%user!eid:number% tty=%-:word% ruser= rhost=
rule=auth,failure: pam_%action!method:char-to:(%(%-:char-to::%:%action!type:char-to:)%):␣
↪received for user %user!name:char-to::%: %-:number% (%msg:char-to:)%)

# Rules for su
rule=authorize,success: Successful su for %user!euser:word% by %user!name:word%


annotate=success:+action.status="success"
annotate=failure:+action.status="failure"
annotate=auth:+action.type="auth"
annotate=authorize:+action.type="authz"
```

### 1.3.2 rsyslog

This configuration fragment provides a ruleset that runs incoming logs through the mm_normalize module, performs some sanity checking on log times (largely for devices without an internal RTC, who always provide bad times on boot, until time is synchronized), generates a table name for logsnarf, and populates some of the JSON fields manually from syslog fields. The easiest way to use this is to assign the ruleset to an input then initializing the input with the rulebase. For example:

```
input(type="imudp" port="514" ruleset="remote")
```

This needs to be declared after the rulebase, as per normal, however.

```
$template JSONDyna,"/srv/log/json/%$YEAR%/%$MONTH%/%$DAY%.log"

# Templates to generate the table name.
template(name="table-index-gen"
  type="list") {
    constant(value="home_logs_")
    property(name="timegenerated" dateFormat="rfc3339" position.from="1" position.to="4")
    property(name="timegenerated" dateFormat="rfc3339" position.from="6" position.to="7")
    property(name="timegenerated" dateFormat="rfc3339" position.from="9" position.to="10
↪")
}

template(name="table-index-rep"
  type="list") {
    constant(value="home_logs_")
    property(name="timereported" dateFormat="rfc3339" position.from="1" position.to="4")
    property(name="timereported" dateFormat="rfc3339" position.from="6" position.to="7")
    property(name="timereported" dateFormat="rfc3339" position.from="9" position.to="10")
}

# Templates for unix timestamps with subsecond accuracy.
template(name="precise-unix-reported" type="list") {
  property(name="timereported" dateFormat="unixtimestamp")
  constant(value=".")
  property(name="timereported" dateFormat="subseconds")
}

template(name="precise-unix-generated" type="list") {
  property(name="timegenerated" dateFormat="unixtimestamp")
  constant(value=".")
  property(name="timegenerated" dateFormat="subseconds")
}

template(name="json_out" type="list") {
  property(name="$!all-json")
  constant(value="\n")
}

ruleset(name="remote") {
    *.* action(type="mmnormalize" ruleBase="/etc/lognorm/lognorm.rulebase")
    unset $!event.tags;
    unset $!originalmsg;
    unset $!unparsed-data;

    set $!time = exec_template("precise-unix-generated");
    set $!timereported = exec_template("precise-unix-reported");
    set $!host = $hostname;
    set $!sev = $syslogseverity-text;
    set $!syslog!fac = $syslogfacility-text;
    set $!syslog!pri = $syslogpriority-text;
    if ($programname == "") and ($procid contains "[") then {
      set $!pid = re_extract($syslogtag, "([0-9]+)]:", 0, 1, "0");
      set $!pname = re_extract($syslogtag, "\\[?([a-z0-9A-Z]+)", 0, 1, "unknown");
```

(continues on next page)

(continued from previous page)

```
    } else {
      set $!pname = $programname;
      set $!pid = $procid;
    }

    set $!msg = $msg;
    set $.timediff = cnum(field($!time, 46, 1)) - cnum(field($!timereported, 46, 1));

    # If the difference in times is greater than one day, trust the generated time more.
    if ($.timediff > 86400) or ($.timediff < -86400) then {
      set $!table = exec_template("table-index-gen");
    }
    else {
      set $!table = exec_template("table-index-rep");
    }
    *.* action(type="omfile" dynaFile="JSONDyna" template="json_out")
}
```

CHAPTER

TWO

# LOGSNARF PACKAGE

## 2.1 Module contents

This package provides classes for streaming log files to BigQuery.

Currently only JSON schema are supported, and the log files must, likewise contain valid line separated JSON. You can however hook into post-decode hooks to conveniently munge or backfill fields based on the log data.

## 2.2 Submodules

### 2.2.1 logsnarf.app module

Logsnarf application.

Application code to connect the various modules to do something useful..

**class** logsnarf.app.**App**(*cfg*, *section_name*)

Bases: object

Logsnarf application class.

Currently this is just a convenient way to encapsulate the setup of the various components. It minimizes the use of the configuration object to make for cleaner implementation classes.

Currently a logsnarf.App contains three main components.

**logsnarf.snarf.Logsnarf - Monitors directories for log updates, feeds them**
linewise to the next in the chain.

**logsnarf.uploader.BigQueryUploader - Receieves lines from Logsnarf,**
pushes them through the logsnarf.schema.Schema object for munging/verification, adds an insertId to every row, batches them into groups, which get a transaction id (internal use only), and passes them on to the BigQueryService. Partial failures (only some rows) are handled here. Full request failures are generally handled by the BigQueryService. Currently this is where flush-on-exit is handled.

**logsnarf.service.BigQueryService - Encapsulates a standard googleapi**
BigQueryService along with our project/dataset information and credentials.

**Parameters**

- **cfg** (logsnarf.config.Config) – config object
- **section_name** (*str*) – section name from the config that tells us what to do

start()

class logsnarf.app.Options

Bases: Options

optParameters = [['resource_name', 'n', 'logsnarf', 'Resource name'], ['config_file', 'f', None, 'Config file']]

logsnarf.app.install_custom_verifiers(*sch*, *default_domain*)

logsnarf.app.install_schema_load_hook(*sch*)

logsnarf.app.main()

### 2.2.2 logsnarf.config module

Logsnarf configuration.

See *configuration* for details on how to configure logsnarf.

class logsnarf.config.Config(*resource_name=None*, *config_file=None*)

Bases: Mapping

Class to contain configuration information for Logsnarf.

This class wraps loading of the configparser config file(s), initializing logging (from a logging.ini), and proxies some methods from xdg.BaseDirectory module for creating/opening config or data files in appropriate places.

It also provides a dict-like read-only interface to the config file, with a vague attempt at type casting through ConfigSection objects.

**Parameters**

- **resource_name** (*str*) – The xdg resource name to use. Defaults to logsnarf
- **config_file** (*str*) – absolute path to a configuration file

load()

Initializes logging and loads the configs.

loadConfigPaths()

Returns the configuration paths for the resource.

Returns an iterator which gives each directory named 'logsnarf' in the configuration search path. Information provided by earlier directories should take precedence over later ones, and the user-specific config dir comes first.

loadConfigs()

Loads all configurations

loadLoggingConfig()

Initializes logging.

The twisted PythonLoggingObserver is initialized, then the config paths are searched for a ${RESOURCE_NAME}/logging.ini, failing that, logging.basicConfig() is called with no arguments so that some sort of logging occurs.

**openConfigFile**(*name*, *mode*)

Opens a file in the user config directory, with the given mode.

**Parameters**

- **name** (*str*) – the name of the config file to open
- **mode** (*str*) – mode to open file with

**Returns**
file object opened with the config file

**Return type**
file

**openDataFile**(*name*, *mode*, **args*, ***kwargs*)

Opens a file in the user data directory, with the given mode.

**Parameters**

- **name** (*str*) – the name of the data file to open
- **mode** (*str*) – mode to open file with

**Returns**
file object opened with the data file

**Return type**
file

**saveConfigPath**(*name=''*)

Ensures the user save config path exists, and returns the path.

Optionally also provide a filename, to receive a full path to that file in the data directory.

Ensure `$XDG_CONFIG_HOME/logsnarf/` exists, and return its path. Use this when saving or updating application configuration.

**Parameters**
**name** (*str*) – the name of the file

**Returns**
path to the directory, or, if given, file

**Return type**
str

**saveDataPath**(*name=''*)

Ensures the user save data path exists, and returns the path to it.

Optionally also provide a filename, to receive a full path to that file in the data directory.

Ensure `$XDG_DATA_HOME/logsnarf/` exists, and return its path. Use this when saving or updating application data.

**Parameters**
**name** (*str*) – the name of the file

**Returns**
path to the directory, or, if given, file

**Return type**
str

class logsnarf.config.**ConfigSection**(*name*, *cfg*)

Bases: Mapping

Class representing a configparser section.

Provides a mapping interface that attempts to do some vaguely sane type casting. No attempt to be smart, cache results etc is done.

**Parameters**

- **name** (*str*) – section name
- **cfg** (*configparser.ConfigParser*) – configparser object

### 2.2.3 logsnarf.errors module

One line description here.

Longer description here.

exception logsnarf.errors.**ConfigError**

Bases: *Error*

Configuration Error.

exception logsnarf.errors.**Error**

Bases: Exception

Base exception class.

exception logsnarf.errors.**MissingConfigValue**(*section*, *item*, *msg=''*)

Bases: *ConfigError*

exception logsnarf.errors.**ServiceError**

Bases: *Error*

Error raised by BigQueryService.

exception logsnarf.errors.**ValidationError**

Bases: ValueError, *Error*

Validation Error.

exception logsnarf.errors.**ValidatorError**

Bases: *Error*

Invalid validator.

### 2.2.4 logsnarf.schema module

class logsnarf.schema.**Schema**(*schema_file*, *default_tz=<UTC>*)

Bases: object

The Schema class represents a BigQuery JSON schema.

Objects of this class are able to

- load and verify schema files which should contain a JSON representation of a list of fields as defined by https://cloud.google.com/bigquery/docs/reference/rest/v2/tables#TableFieldSchema

- parse JSON strings, cooercing where able and appropriate fields to appropriate types as defined by the schema.
- validate python objects against the BigQuery schema.

**Parameters**

- **schema_file** (*file*) – File-like object containing the BigQuery JSON schema.
- **default_tz** (*datetime.tzinfo*) – Timezone to use on date strings that don't contain TZ information.

**Raises**

ValueError – if the schema file doesn't contain valid JSON

`clearPostprocessors()`

Removes all post processors.

`ignore_fields = ['table', '_sha1']`

Fields in this list are permitted, even if they aren't part of the schema. In Logsnarf we use this for the tables field, which tells us which table this log line belongs in, and we remove it from the entry before upload.

`loads(json_string)`

Deserialize json_string into a python object.

This applies all schema checks and post-processors.

**Parameters**

**json_string** (*string|bytes*) – utf-8 encoded string containing a JSON document.

**Returns**

The JSON document as a python object

**Return type**

dict or list or integer or float or unicode

**Raises**

- *logsnarf.errors.ValidationError* – if the JSON is valid, but does not contain a document that conforms to the BigQuery schema.
- ValueError – if the string does not contain a valid JSON document.

`registerPostprocessor(fn)`

Register a post processor.

Registers a function to be called on the result of every JSON object decoded by the Schema object.

**Parameters**

**fn** (*callable*) – A callable that takes on argument, the decoded JSON object, and returns the new version of that object.

`setFieldValidator(field_name, fn)`

Override the validator for a particular field in the schema.

**Parameters**

- **field_name** (*str*) – The field name to replace the validator for. If referring to a field of a subrecord, use dotted notation. e.g. recordfield.subrecord.item
- **fn** (*callable*) – A callable that recieves the root object, and the current value of the field, and returns the new value. In the case where the value is invalid, it should raise errors.ValidationError

setObjectLoadHook(*fn*)

Set the object load hook used by json.loads.

**Parameters**
**fn** (*callable*) – A callable that takes a non-literal, decoded json object, and returns an updated version of that object.

toUnixTimestamp(*_parent*, *value*)

Validator for TIMESTAMP fields.

**Parameters**

- **_parent** (*dict*) – Parent of the value.
- **value** (*str or integer or float*) – The value to validate.

**Returns**
validated value

**Return type**
float

**Raises**
*logsnarf.errors.ValidationError* – if value is not, or can not be converted to, a unix timestamp.

validateJSON(*root_obj*)

Validate that an object matches the BigQuery schema.

**This involves**

- ensuring all fields in the object are known
- all required fields are present.
- running the field validators on each field

**Parameters**
**root_obj** (*dict*) – the object (dict) to validate against the schema.

**Returns**
validated object

**Return type**
dict

**Raises**
*logsnarf.errors.ValidationError* – if the object is not valid against the schema

validateSchema()

Validate that the JSON document we loaded as schema, is valid.

static validateSchemaField(*field*)

Validate a field of a schema.

For clarity this is implemented with asserts. During normal schema validation this is wrapped in a ValidationError in validateSchema

**Parameters**
**field** (*dict*) – The field to validate.

**Raises**
AssertionError – if the field is invalid.

### 2.2.5 logsnarf.service module

BigQuery service.

**class** logsnarf.service.**BigQueryService**(*project_id*, *dataset*, *creds*, *reactor=None*, *debug=False*)

Bases: object

A fairly basic wrapper around the google BigQuery API.

**Parameters**

- **project_id** (*int*) – Numeric project id
- **dataset** (*str*) – Dataset name
- **creds** (*oauth2client.Credentials*) – oauth2 credentials
- **reactor** (twisted.internet.reactor) – twisted reactor object

**createTable**(*name*, *table_schema*)

Create a BigQuery table

**Parameters**

- **name** (*str*) – name of the table to create
- **table_schema** (*list*) – a list of fields representing the schema for the new table

**Raises**

- **googleapiclient.errors.HttpError** – if a HTTP error occurs
- **ssl.SSLError** – if an SSL based error occurs

**property http**

Creates and authorizes a httplib2.Http() instance

**insertAll**(*table*, *table_schema*, *data*, *upload_id=None*)

Insert rows into a table. Create the table if necessary.

This is typically called in the main thread.

**Parameters**

- **table** (*str*) – table to insert data to.
- **table_schema** (*list*) – list of fields, representing the table schema this can be ommitted if the table already exists.
- **data** (*list(dict)*) – the rows to be intersted. usually a list of dicts.
- **upload_id** (*str*) – unique identifier for this upload, only used internally for logging. one is generated if not supplied

**Returns**

a deferred for the results, the form of which is described at https://cloud.google.com/bigquery/docs/reference/rest/v2/tabledata/insertAll#response-body

**Return type**

twisted.internet.defer.Deferred

**insertAll_s**(*table*, *table_schema*, *data*, *upload_id=None*)

Synchronous version of *insertAll()*.

**Parameters**

- **table** (*str*) – table to insert data to.
- **table_schema** (*list*) – list of fields, representing the table schema this can be ommitted if the table already exists.
- **data** (*list(dict)*) – the rows to be intersted. usually a list of dicts.
- **upload_id** (*str*) – unique identifier for this upload, only used internally for logging. one is generated if not supplied

**Returns**
Results of the insert.

**Return type**
https://cloud.google.com/bigquery/docs/reference/rest/v2/tabledata/insertAll#response-body

**property service**

Creates a BigQuery service.

**updateTableList()**

Update our internal cache of tables.

### 2.2.6 logsnarf.snarf module

Log directory watcher and reader.

This class is responsible for watching a set of directories for updates to files matching a specific pattern. When those updates happen, it reads those changes and passes it to the consumer given to it at initialization time.

Updates are sent to the consumer as complete lines (including newlines). Infile progress is tracked via a persistent state object, which tracks the inode and file offset.

**class** logsnarf.snarf.**LogSnarf**(*state_obj*, *consumer*, *reactor=None*)

Bases: object

Main logsnarf class.

implements twisted.internet.interfaces.IPushProducer

Initialize a LogSnarf object.

**Parameters**

- **state_obj** (logsnarf.state.State) – current state
- **consumer** (implements(twisted.internet.interfaces.IConsumer)) – a consumer object

**checkPattern**(*path*)

Check a path against our pattern.

**Parameters**
**path** (*twisted.python.filepath.FilePath*) – The path to check

**Returns bool**
true if the path matches.

**cleanState()**

Remove invalid entries from the state file.

**doRead**(*path*)

read from a file.

**Parameters**
**path** (twisted.python.filepath.FilePath) – the file to read

**pauseProducing**()

twisted.internet.interfaces.IPushProducer method

Pause producing.

**resumeProducing**()

twisted.internet.interfaces.IPushProducer method

Resume producing.

**setCallback**(*callback*)

Add a callback function.

The callback should have a signature of f(line), accepting one line (including newline) of text.

**start**()

Start watching.

**watch**(*path*, *pattern=None*, *recursive=True*)

Add a watch to a given directory.

**Parameters**

- **path** (string or twisted.python.filepath.FilePath) – directory to watch for changes in
- **pattern** (*re.RegexObject or None*) – regular expression that files much match
- **recursive** (*bool*) – If true, also watch subdirectories.

### 2.2.7 logsnarf.state module

Persistent state

Basically just a wrapper around a dict that saves on mutate. This doesn't change often, so is fine for our needs at the moment.

**class** logsnarf.state.**State**(*state_path*)

Bases: MutableMapping

A persistent dictionary.

State is initially loaded from a json encoded file. State is saved to that file on every mutate.

Create the state object from a file.

**Parameters**
**state_path** (*string*) – Path the JSON encoded state file.

**save**()

Save current state.

### 2.2.8 logsnarf.uploader module

BigQuery uploader

A class that implements twisted.internet.interfaces.IConsumer for uploading logs to BigQuery.

**class** logsnarf.uploader.**BigQueryUploader**(*schema_obj*, *svc*, *table_name_schema*, *reactor=None*)

Bases: _ConsumerMixin

**Parameters**

- **schema_obj** (logsnarf.schema.Schema) – A BigQuery schema
- **svc** (*logsnarf.service.BigQuery*) – a Bigquery service
- **table_name_schema** (*str*) – format to create table names. see *Configuring Logsnarf* for more details.
- **reactor** (twisted.internet.reactor) – twisted reactor

**addData**(*data*)

This expects valid dicts to upload.

**flush**()

Flush our buffer.

**pauseConsuming**()

Pause consuming, by pausing our producer.

**registerProducer**(*producer*, *streaming*)

twisted.internet.interfaces.IConsumer method

Registers a producer with the uploader.

**Parameters**

- **producer** (twisted.internet.interfaces.IProducer) – a producer
- **streaming** (*bool*) – true if the producer is a streaming producer

**resumeConsuming**()

Resume consuming, by resuming our producer.

**setBatchSize**(*n*)

Set the number of log entries to batch in an upload.

This is both a maximum batch size, minimum batch size. Barring uploads that occur at an interval set by *setFlushInterval()*

**Parameters**

**n** (*int*) – size to set batchsize to

**setDefaultTZ**(*tz*)

Set the default timezone

**Parameters**

**tz** (*str or datetime.tzinfo*) – timezone to set as default

**setFlushInterval**(*n*)

Set the flush interval for the uploader.

Uploads will occur at least every flush interval seconds, as long as there is anything to upload.

**Parameters**
**n** (*int*) – flush interval

`setMaxBuffer`(*n*)

Set the max buffer size for uploads.

Once this is reached, the uploader will ask the producer to pause until the buffer is below this limit.

**Parameters**
**n** (*int*) – max buffer size

`start`()

Start the uploader.

Start periodic tasks, at a trigger to flush our buffer on system shutdown. Make sure the producer is set to produce.

`startWriting`()

Required by _ConsumerMixin.

`upload`(*flush=False*)

Potentially insert some table data.

**Parameters**
**flush** (*bool*) – if this is part of flushing our buffer

`write`(*data*)

twisted.internet.interfaces.IConsumer method

Process data and add it to our buffer. It's preferable, but not required that full log lines are taken here.

**Parameters**
**data** (*str*) – text containing line separated JSON

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## l

# INDEX

## O



## P



## R



## S



## T



## U



## V



## W