SimpleMonitor

James Seward

Jun 11, 2023

GETTING STARTED

CHAPTER

ONE

INSTALLATION

SimpleMonitor is available via PyPi:

```
pip install simplemonitor
```

Tip: You may want to install it in a virtualenv, or you can use pipx which automatically manages virtualenvs for command-line tools.

Create the configuration files: `monitor.ini` and `monitors.ini`. See Configuration.

Warning: I know the configuration file names are dumb, sorry.

1.1 Running

Just run:

```
simplemonitor
```

SimpleMonitor does not fork. For best results, run it with a service management tool such as daemontools, supervisor, or systemd. You can find some sample configurations for this purpose on GitHub.

SimpleMonitor will look for its configuration files in the current working directory. You can specify a different configuration file using `-f`.

You can verify the configuration files syntax with `-t`.

By default, SimpleMonitor's output is limited to errors and other issues, and it emits a `.` character every two loops. Use `-H` to disable the latter, and `-v`, `-d` and `-q` (or `-l`) to control the former.

If you are using something like systemd or multilog which add their own timestamps to the start of the line, you may want `--no-timestamps` to avoid having unnecessary timestamps added.

1.2 Command Line Options Reference

General options

-h, --help show help message and exit

--version show version number and exit

Execution options

-p PIDFILE, --pidfile PIDFILE Write PID into this file

-N, --no-network Disable network listening socket (if enabled in config)

-f CONFIG, --config CONFIG configuration file (this is the main config; you also need monitors.ini (default filename)

-j THREADS, --threads THREADS number of threads to run for checking monitors (default is number of CPUs detected)

Output options

-v, --verbose Alias for `--log-level=info`

-q, --quiet Alias for `--log-level=critical`

-d, --debug Alias for `--log-level=debug`

-H, --no-heartbeat Omit printing the `.` character when running checks

-l LOGLEVEL, --log-level LOGLEVEL Log level: critical, error, warn, info, debug

-C, --no-colour, --no-color Do not colourise log output

--no-timestamps Do not prefix log output with timestamps

Testing options

-t, --test Test config and exit

These options are really for testing SimpleMonitor itself, and you probably don't need them.

-1, --one-shot Run the monitors once only, without alerting. Require monitors without "fail" in the name to succeed. Exit zero or non-zero accordingly.

--loops LOOPS Number of iterations to run before exiting

--dump-known-resources Print out loaded Monitor, Alerter and Logger types

CHAPTER

TWO

CONFIGURATION

The main configuration lives in `monitor.ini`. By default, SimpleMonitor will look for it in the working directory when launched. To specify a different file, use the `-f` option.

The format is fairly standard "INI"; section names are lowercase in `[square brackets]`, and values inside the sections are defined as `key=value`. You can use blank lines to space things out, and comments start with `#`.

Section names and option values, but not option names, support environment variable injection. To include the value of an environment variable, use `%env:VARIABLE%`, which will inject the value of `$VARAIBLE` from the environment. You can use this to share a common configuration file across multiple hosts, for example.

This main configuration file contains the global settings for SimpleMonitor, plus the logging and alerting configuration. A separate file, by default `monitors.ini`, contains the monitor configuration. You can specify a different monitors configuration file using a directive in the main configuration.

Warning: I know the configuration file names are dumb, sorry.

2.1 Configuration value types

Values which take **bool** accept `1`, `yes`, and `true` as truthy, and everything else as falsey.

Values which take **bytes** accept suffixes of `K`, `M`, or `G` for kibibytes, mibibytes or gibibytes, otherwise are just a number of bytes.

2.2 `monitor.ini`

This file must contain a `[monitor]` section, which must contain at least the `interval` setting.

2.2.1 `[monitor]` section

`interval`

Type integer

Required true

defines how many seconds to wait between running all the monitors. Note that the time taken to run the monitors is not subtracted from the interval, so the next iteration will run at interval + time_to_run_monitors seconds.

`monitors`

Type string

Required false

Default `monitors.ini`

the filename to load the monitors themselves from. Relative to the cwd, not the path of this configuration file.

`pidfile`

Type string

Required false

Default none

the path to write a pidfile to.

`remote`

Type bool

Required false

Default false

enables the listener for receiving data from remote instances. Can be overridden to disabled with `-N` command line option.

`remote_port`

Type integer

Required if `remote` is enabled

the TCP port to listen on for remote data

`key`

Type string

Required if `remote` is enabled

shared secret for validating data from remote instances.

`bind_host`

Type string

Required false

Default `0.0.0.0` (all interfaces)

the local IP address to listen on, if `remote` is enabled.

`hup_file`

Type string

Required false

Default none

a file to watch the modification time on. If the modification time increases, SimpleMonitor *reloads its configuration*.

Tip: SimpleMonitor will reload if it receives SIGHUP; this option is useful for platforms which don't have that.

bind_host

Type string

Required false

Default all interfaces

the local address to bind to for remote data

2.2.2 [reporting] section

loggers

Type comma-separated list of string

Required false

Default none

the names of the loggers you want to use. Each one must be a [section] in this configuration file.

See Loggers for the common options and list of Alerters with their configurations.

alerters

Type comma-separated list of string

Required false

Default none

the names of the alerters you want to use. Each one must be a [section] in this configuration file.

See Alerters for the common options and list of Alerters with their configurations.

2.3 monitors.ini

This file only contains monitors. Each monitor is a [section] in the file, with the section name giving the monitor its name. The name defaults is reserved, and can be used to specify default values for options. Each monitor's individual configuration overrides the defaults.

See Monitors for the common options and list of Monitors with their configurations.

2.4 Example configuration

This is an example pair of configuration files to show what goes where. For more examples, see Config examples.

monitor.ini:

```
[monitor]
interval=60

[reporting]
loggers=logfile
alerters=email,sms
```

(continues on next page)

(continued from previous page)

```
# write a log file with the state of each monitor, each time
[logfile]
type=logfile
filename=monitor.log

# email me when monitors fail or succeed
[email]
type=email
host=mailserver.example.com
from=monitor@example.com
to=admin@example.com

# send me an SMS after a monitor has failed 10 times in a row
[sms]
type=bulksms
username=some-username
password=some-password
target=+447777123456
limit=10
```

monitors.ini:

```
# check the webserver pings
[www-ping]
type=ping
host=www.example.com

# check the webserver answers https; don't bother checking if it's not pinging
[www-http]
type=http
url=https://www.example.com
depend=www-ping

# check the root partition has at least 1GB of free space
[root-diskspace]
type=diskspace
partition=/
limit=1G
```

2.5 Reloading

You can send SimpleMonitor a SIGHUP to make it reload its configuration. On platforms which don't have that (e.g. Windows), you can specify a file to watch. If the modification time of the file changes, SimpleMonitor will reload its configration.

Reloading will pick up a change to `interval` but no other configuration in the `[monitor]` section. Monitors, Alerters and Loggers are reloaded. You can add and remove them, and change their configurations, but not change their types. (To change a type, first remove it from the configuration and reload, then add it back in.)

CHAPTER

THREE

MONITOR CONFIGURATION

Monitors are defined in (by default) `monitors.ini`. The monitor is named by its `[section]` heading. If you create a `[defaults]` section, the values are used as defaults for all the other monitors. Each monitor's configuration will override the values from the default.

Contents

3.1 Common options

These options are common to all monitor types.

`type`

Type string

Required true

the type of the monitor; one of those in the list below.

`runon`

Type string

Required false

Default

none

a hostname on which the monitor should run. If not set, always runs. You can use this to share one config file among many hosts. (The value which is compared to is that returned by Python's `socket.gethostname()`.)

`depend`

Type comma-separated list of string

Required false

Default none

the monitors on which this one depends. This monitor will run after those, unless one of them fails or is skipped, in which case this one will also skip. A skip does not trigger an alerter.

`tolerance`

Type integer

Required false

Default 1

the number of times a monitor can fail before it enters the failed state. Handy for things which intermittently fail, such as unreliable links. The number of times the monitor has actually failed, minus this number, is its "Virtual Failure Count". See also the *limit* option on Alerters.

`urgent`

Type boolean

Required false

Default true

if this monitor is "urgent" or not. Non-urgent monitors do not trigger urgent alerters (e.g. BulkSMS)

`gap`

Type integer

Required false

Default 0

the number of seconds this monitor should allow to pass before polling. Use it to make a monitor poll only once an hour (`3600`), for example. Setting this value lower than the `interval` will have no effect, and the monitor will run every loop like normal.

Some monitors default to a higher value when it doesn't make sense to run their check too frequently because the underlying data will not change that often or quickly, such as *pkgaudit*. You can override their default to a lower value as required.

Hint: Monitors which are in the failed state will poll every loop, regardless of this setting, in order to detect recovery as quickly as possible

---

`remote_alert`

Type boolean

Required false

Default false

set to true to have this monitor's alerting handled by a remote instance instead of the local one. If you're using the remote feature, this is a good candidate to put in the `[defaults]`.

`recover_command`

Type string

Required false

Default none

a command to execute once when this monitor enters the failed state. For example, it could attempt to restart a service.

`recovered_command`

Type string

Required false

Default none

a command to execute once when this monitor returns to the OK state. For example, it could restart a service which was affected by the failure of what this monitor checks.

`notify`

Type boolean

Required false

Default true

if this monitor should alert at all.

`group`

Type string

Required false

Default `default`

the group the monitor belongs to. Alerters and Loggers will only fire for monitors which appear in their groups.

`failure_doc`

Type string

Required false

Default none

information to include in alerts on failure (e.g. a URL to a runbook)

`gps`

Type string

Required no, unless you want to use the *html logger*'s map

comma-separated latitude and longitude of this monitor

3.2 Monitors

Note: The `type` of the monitor is the first word in its heading.

3.2.1 apcupsd - APC UPS status

Uses an existing and configured `apcupsd` installation to check the UPS status. Any status other than `ONLINE` is a failure.

path

> **Type** string
>
> **Required** false
>
> **Default** none

the path to the `apcaccess` binary. On Windows, defaults to `C:\apcupsd\bin`. On other platforms, looks in `$PATH`.

3.2.2 arlo_camera - Arlo camera battery level

Checks Arlo camera battery level is high enough.

username

> **Type** string
>
> **Required** true

Arlo username

password

> **Type** string
>
> **Required** true

Arlo password

device_name

> **Type** string
>
> **Required** true

the device to check (e.g. `Front Camera`)

base_station_id

> **Type** integer
>
> **Required** false
>
> **Default** `0`

the number of your base station. Only required if you have more than one. It's an array index, but figuring out which is which is an exercise left to the reader.

3.2.3 command - run an external command

Run a command, and optionally verify its output. If the command exits non-zero, this monitor fails.

`command`

Type string

Required true

the command to run.

`result_regexp`

Type string (regular expression)

Required false

Default none

if supplied, the output of the command must match else the monitor fails.

`result_max`

Type integer

Required false

if supplied, the output of the command is evaluated as an integer and if greater than this, the monitor fails. If the output cannot be converted to an integer, the monitor fails.

3.2.4 compound - combine monitors

Combine (logical-and) multiple monitors. By default, if any monitor in the list is OK, this monitor is OK. If they all fail, this monitor fails. To change this limit use the `min_fail` setting.

> **Warning:** Do not specify the other monitors in this monitor's `depends` setting. The dependency handling for compound monitors is a special case and done for you.

`monitors`

Type comma-separated list of string

Required true

the monitors to combine

`min_fail`

Type integer

Required false

Default the number of monitors in the list

the number of monitors from the list which should be failed for this monitor to fail. The default is that all the monitors must fail.

3.2.5 diskspace - free disk space

Checks the free space on the given partition/drive.

partition

Type string

Required true

the partition/drive to check. On Windows, give the drive letter (e.g. `C:`). Otherwise, give the mountpoint (e.g. `/usr`).

limit

Type *bytes*

Required true

the minimum allowed amount of free space.

3.2.6 dns - resolve record

Attempts to resolve the DNS record, and optionally checks the result. Requires `dig` to be installed and on the PATH.

record

Type string

Required true

the DNS name to resolve

record_type

Type string

Required false

Default `A`

the type of record to request

desired_val

Type string

Required false

if not given, this monitor simply checks the record resolves.

Give the special value `NXDOMAIN` to check the record **does not** resolve.

If you need to check a multivalue response (e.g. MX records), format them like this (note the leading spaces on the continuation lines):

```
desired_val=10 a.mx.domain.com
  20 b.mx.domain.com
  30 c.mx.domain.com
```

server

Type string

Required false

the server to send the request to. If not given, uses the system default.

port

Type integer

Required false

Default `53`

the port on the DNS server to use

3.2.7 eximqueue - Exim queue size

Checks the output of `exigrep` to make sure the queue isn't too big.

max_length

Type integer

Required false

Default `1`

the maximum acceptable queue length

path

Type string

Required false

Default `/usr/local/sbin`

the path containing the `exigrep` binary

3.2.8 fail - alawys fails

This monitor fails 5 times in a row, then succeeds once. Use for testing. See the *null* monitor for the inverse.

3.2.9 filestat - file size and age

Examines a file's size and age. If neither of the age/size values are given, simply checks the file exists.

filename

Type string

Required true

the path of the file to monitor.

maxage

Type integer

Required false

the maximum allowed age of the file in seconds. If not given, not checked.

minsize

Type *bytes*

Required false

the minimum allowed size of the file in bytes. If not given, not checked.

3.2.10 hass_sensor - Home Automation Sensors

This monitor checks for the existence of a home automation sensor.

url

Type string

Required true

API URL for the monitor

sensor

Type string

Required true

the name of the sensor

token

Type string

Required true

API token for the sensor

timeout

Type int

Required false

Default 5

Timeout for HTTP request to HASS

3.2.11 host - ping a host

Check a host is pingable.

Tip: This monitor relies on executing the `ping` command provided by your OS. It has known issues on non-English locales on Windows. You should use the *ping* monitor instead. The only reason to use this one is that it does not require SimpleMonitor to run as root.

---

host

Type string

Required true

the hostname/IP to ping

ping_regexp

Type regexp

Required false

Default automatic

the regexp which matches a successful ping. You may need to set this to use this monitor in a non-English locale.

`time_regexp`

Type regexp

Required false

Default automatic

the regexp which matches the ping time in the output. Must set a match group named `ms`. You may need to set this as above.

3.2.12 http - fetch and verify a URL

Attempts to fetch a URL and makes sure the HTTP return code is (by default) 200/OK. Can also match the content of the page to a regular expression.

`url`

Type string

Required true

the URL to open

`regexp`

Type regexp

Required false

Default none

the regexp to look for in the body of the response

`allowed_codes`

Type comma-separated list of integer

Required false

Default *200*

a list of acceptable HTTP status codes

`allow_redirects`

Type bool

Required false

Default *true*

Follow redirects

`username`

Type str

Required false

Default none

Username for http basic auth

`password`

Type str

Required false

Default none

Password for http basic auth

`verify_hostname`

Type boolean

Required false

Default true

set to false to disable SSL hostname verification (e.g. with self-signed certificates)

`timeout`

Type integer

Required false

Default `5`

the timeout in seconds for the HTTP request to complete

`headers`

Type JSON map as string

Required false

Default `{}`

JSON map of HTTP header names and values to add to the request

3.2.13 loadavg - load average

Check the load average on the host.

`which`

Type integer

Required false

Default `1`

the load average to monitor. `0` = 1min, `1` = 5min, `2` = 15min

`max`

Type float

Required false

Default `1.00`

the maximum acceptable load average

3.2.14 memory - free memory percent

Check free memory percentage.

percent_free

Type int

Required true

the minimum percent of available (as per psutils' definition) memory

3.2.15 null - always passes

Monitor which always passes. Use for testing. See the *fail* monitor for the inverse.

This monitor has no additional parameters.

3.2.16 ping - ping a host

Pings a host to make sure it's up. Uses a Python ping module instead of calling out to an external app, but needs to be run as root.

host

Type string

Required true

the hostname or IP to ping

timeout

Type int

Required false

Default 5

the timeout for the ping in seconds

3.2.17 pkgaudit - FreeBSD pkg audit

Fails if `pkg audit` reports any vulnerable packages installed.

path

Type string

Required false

Default /usr/local/sbin/pkg

the path to the `pkg` binary

3.2.18 portaudit - FreeBSD port audit

Fails if `portaudit` reports any vulnerable ports installed.

`path`

Type string

Required false

Default `/usr/local/sbin/portaudit`

the path to the `portaudit` binary

3.2.19 process - running process

Check for a running process.

`process_name`

Type string

Required true

the process name to check for

`min_count`

Type integer

Required false

Default 1

the minimum number of matching processes

`max_count`

Type integer

Required false

Default infinity

the maximum number of matching processes

`username`

Type string

Required false

Default any user

limit matches to processes owned by this user.

3.2.20 rc - FreeBSD rc service

Checks a FreeBSD-style service is running, by running its rc script (in /usr/local/etc/rc.d) with the status command.

Tip: You may want the *unix_service* monitor for a more generic check.

service

Type string

Required true

the name of the service to check. Should be the name of the rc.d script in `/usr/local/etc/rc.d`. Any trailing `.sh` is optional and added if needed.

path

Type string

Required false

Default `/usr/local/etc/rc.d`

the path of the folder containing the rc script.

return_code

Type integer

Required false

Default `0`

the required return code from the script

3.2.21 ring_doorbell - Ring doorbell battery

Check the battery level of a Ring doorbell.

device_name

Type string

Required true

the name of the Ring Doorbell to monitor.

minimum_battery

Type integer

Required false

Default `25`

the minimum battery percent allowed.

username

Type string

Required true

your Ring username (e.g. email address). Accounts using MFA are not supported. You can create a separate user for API access.

`password`

Type string

Required true

your Ring password.

Warning: Do not commit credentials to source control!

`device_type`

Type string

Required false

Default `doorbell`

the device type. Acceptable values are `doorbell` or `camera`.

3.2.22 service - Windows Service

Checks a Windows service to make sure it's in the correct state.

`service`

Type string

Required true

the short name of the service to monitor (this is the "Service Name" on the General tab of the service Properties in the Services MMC snap-in).

`want_state`

Type string

Required false

Default `RUNNING`

the required status for the service. One of:

- `RUNNING`
- `STOPPED`
- `PAUSED`
- `START_PENDING`
- `PAUSE_PENDING`
- `CONTINUE_PENDING`
- `STOP_PENDING`

Tip: version 1.9 and earlier had a **host** parameter, which is no longer used.

3.2.23 svc - daemontools service

Checks a daemontools `supervise`-managed service is running.

`path`

Type string

Required true

the path to the service's directory (e.g. `/var/service/something`)

3.2.24 swap - available swap space

Checks for available swap space.

`percent_free`

Type integer

Required true

minimum acceptable free swap percent

3.2.25 systemd-unit - systemd unit check

Monitors a systemd unit status, via dbus. You may want the *unix_service* monitor instead if you just want to ensure a service is running.

`name`

Type string

Required true

the name of the unit to monitor

`load_states`

Type comma-separated list of string

Required false

Default `loaded`

desired load states for the unit

`active_states`

Type comma-separated list of string

Required false

Default `active,reloading`

desired active states for the unit

`sub_states`

Type comma-separated list of string

Required false

Default none

desired sub states for the service

3.2.26 tcp - open TCP port

Checks a TCP port is connectible. Doesn't care what happens after the connection is opened.

host

Type string

Required true

the name/IP of the host to connect to

port

Type integer

Required true

the port number to connect to.

3.2.27 tls_expiry - TLS cert expiration

Checks an SSL/TLS certificate is not due to expire/has expired.

Note: This monitor's *gap* defaults to 12 hours.

Warning: Due to a limitation of the underlying Python modules in use, this does not currently support TLS 1.3.

host

Type string

Required true

the hostname to connect to

port

Type integer

Required false

Default `443`

the port number to connect on

min_days

Type integer

Required false

Default `7`

the minimum allowable number of days until expiry

sni

Type string

Required false

the hostname to send during TLS handshake for SNI. Use if you are serving multiple certificates from the same host/port. If empty, will just get the default certificate from the server

3.2.28 unifi_failover - USG failover WAN status

Checks a Unifi Security Gateway for failover WAN status. Connects via SSH; the USG must be in your `known_hosts` file. Requires the specified interface to have the carrier up, a gateway, and not be in the `failover` state.

`router_address`

Type string

Required true

the address of the USG

`router_username`

Type string

Required true

the username to log in as

`router_password`

Type string

Required conditional

the password to log in with. Required if not using `ssh_key`.

`ssh_key`

Type string

Required conditional

path to the SSH private key to log in with. Required if not using `router_password`.

`check_interface`

Type string

Required false

Default `eth2`

the interface which should be ready for failover.

3.2.29 unifi_watchdog - USG failover watchdog

Checks a Unifi Security Gateway to make sure the failover WAN is healthy. Connects via SSH; the USG must be in your `known_hosts` file. Requires the specified interface to have status `Running` and the ping target to be `REACHABLE`.

`router_address`

Type string

Required true

the address of the USG

`router_username`

Type string

Required true

the username to log in as

`router_password`

Type string

Required conditional

the password to log in with. Required if not using `ssh_key`.

`ssh_key`

Type string

Required conditional

path to the SSH private key to log in with. Required if not using `router_password`.

`primary_interface`

Type string

Required false

Default `pppoe0`

the primary WAN interface

`secondary_interface`

Type string

Required false

Default `eth2`

the secondary (failover) WAN interface

3.2.30 unix_service - generic UNIX service

Generic UNIX service check, by running `service ... status`.

`service`

Type string

Required true

the name of the service to check

`state`

Type string

Required false

Default `running`

the state of the service; either `running` (status command exits 0) or `stopped` (status command exits 1).

CHAPTER

FOUR

ALERTER CONFIGURATION

Alerters send one-off alerts when a monitor fails. They can also send an alert when it succeeds again.

An alerter knows if it is urgent or not; if a monitor defined as non-urgent fails, an urgent alerter will not trigger for it. This means you can avoid receiving SMS alerts for things which don't require your immediate attention.

Alerters can also have a time configuration for hours when they are or are not allowed to alert. They can also send an alert at the end of the silence period for any monitors which are currently failed.

Alerters are defined in the main configuration file, which by default is `monitor.ini`. The section name is the name of your alerter, which you should then add to the `alerters` configuration value.

Contents

4.1 Common options

These options are common to all alerter types.

`type`

Type string

Required true

the type of the alerter; one of those in the list below.

`depend`

Type comma-separated list of string

Required false

Default none

a list of monitors this alerter depends on. If any of them fail, no attempt will be made to send the alert.

`limit`

Type integer

Required false

Default 1

the (virtual) number of times a monitor must have failed before this alerter fires for it. You can use this to escalate an alert to another email address or text messaging, for example. See the *tolerance* Monitor configuration option.

`dry_run`

Type boolean

Required false

Default `false`

makes an alerter do everything except actually send the message, and instead will print some information about what it would do.

`ooh_success`

Type boolean

Required false

Default `false`

makes an alerter trigger its success action even if out of hours

`groups`

Type comma-separated list of string

Required false

Default `default`

list of monitor groups this alerter should fire for. See the *group* setting for monitors.

`only_failures`

Type boolean

Required false

Default `false`

if true, only send alerts for failures (or catchups)

`tz`

Type string

Required false

Default `UTC`

the timezone to use in alert messages. See also *times_tz*.

`repeat`

Type boolean

Required false

Default `false`

fire this alerter (for a failed monitor) every iteration

`urgent`

Type boolean

Required false

if the alerter should be urgent or not. The default varies from alerter to alerter. Typically, those which send "page" style alerts such as SMS default to urgent. You can use this option to override that in e.g. the case of the SNS alerter, which could be urgent if sending SMSes, but non-urgent if sending emails.

4.2 Time restrictions

All alerters accept time period configuration. By default, an alerter is active at all times, so you will always immediately receive an alert at the point where a monitor has failed enough (more times than the limit). To set limits on when an alerter can send, use the configuration values below.

Note that the *times_type* option sets the timezone all the values are interpreted as. The default is the local timezone of the host evaluating the logic.

day

Type comma-separated list of integer

Required false

Default all days

which days an alerter can operate on. `0` is Monday, `6` is Sunday.

times_type

Type string

Required false

Default `always`

one of `always`, `only`, or `not`. `only` means that the limits specify the period the alerter is allowed to operate in. `not` means the specify the period it isn't, and outside of that time it is allowed.

time_lower

Type string

Required when *times_type* is not `always`

the lower end of the time range. Must be lower than *time_upper*. The format is `HH:mm` in 24-hour clock.

time_upper

Type string

Required when *times_type* is not `always`

the upper end of the time range. Must be lower than *time_lower*. The format is `HH:mm` in 24-hour clock.

times_tz

Type string

Required false

Default the host's local time

the timezone for *day*, *time_lower* and *time_upper* to be interpreted in.

delay

Type boolean

Required false

Default `false`

set to true to have the alerter send a "catch-up" alert about a failed monitor if it failed during a time the alerter was not allowed to send, and is still failed as the alerter enters the time it is allowed to send. If the monitor fails and recovers during the not-allowed time, no alert is sent either way.

4.2.1 Time examples

These snippets omit the alerter-specific configuration values.

Don't trigger during the hours I'm in the office (8:30am to 5:30pm, Monday to Friday):

```
[out_of_hours]
type=some-alerter-type
times_type=not
time_lower=08:30
time_upper_17:30
days=0,1,2,3,4
```

Don't send at antisocial times, but let me know later if something broke and hasn't recovered yet:

```
[polite_alerter]
type=some-alerter-type
times_type=only
time_lower=07:30
time_upper=22:00
delay=1
```

4.3 Alerters

Note: The `type` of the alerter is the first word in its heading.

4.3.1 46elks - 46elks notifications

Warning: Do not commit your credentials to a public repo!

You will need to register for an account at 46elks.

`username`

Type string

Required true

your 46wlks username

`password`

Type string

Required true

your 46wlks password

`target`

Type string

Required true

46elks target value

`sender`

Type string

Required false

Default `SmplMntr`

your SMS sender field. Start with a + if using a phone number.

`api_host`

Type string

Required false

Default `api.46elks.com`

API endpoint to use

`timeout`

Type int

Required false

Default `5`

Timeout for HTTP request

4.3.2 bulksms - SMS via BulkSMS

Warning: Do not commit your credentials to a public repo!

`sender`

Type string

Required false

Default `SmplMntr`

who the SMS should appear to be from. Max 11 chars, and best to stick to alphanumerics.

`username`

Type string

Required true

your BulkSMS username

`password`

Type string

Required true

your BulkSMS password

`target`

Type string

Required true

the number to send the SMS to. Specify using country code and number, with no + or international prefix. For example, `447777123456` for a UK mobile.

`timeout`

Type int

Required false

Default `5`

Timeout for HTTP request

4.3.3 email - send via SMTP

Warning: Do not commit your credentials to a public repo!

`host`

Type string

Required true

the email server to connect to

`port`

Type integer

Required false

Default `25`

the port to connect on

`from`

Type string

Required true

the email address to give as the sender

`to`

Type string

Required true

the email address to send to. You can specify multiple addresses by separating with `;`.

`cc`

Type string

Required false

the email address to cc to. You can specify multiple addresses by separating with `;`.

`username`

Type string

Required false

the username to log in to the SMTP server with

`password`

Type string

Required false

the password to log in to the SMTP server with

`ssl`

Type string

Required false

specify `starttls` to use StartTLS. Specify ``yes`` to use SMTP SSL. Otherwise, no SSL is used at all.

4.3.4 execute - run external command

`fail_command`

Type string

Required false

command to execute when a monitor fails

`success_command`

Type string

Required false

command to execute when a montior recovers

`catchup_command`

Type string

Required false

command to execute when exiting a time period when the alerter couldn't fire, a monitor failed during that time, and hasn't recovered yet. (See the *delay* configuration option.) If you specify the literal string `fail_command`, this will share the *fail_command* configuration value.

You can specify the following variable inside `{curly brackets}` to have them substituted when the command is executed:

- `hostname`: the host the monitor is running on
- `name`: the monitor's name
- `days`, `hours`, `minutes`, and `seconds`: the monitor's downtime

- `failed_at`: the date and time the monitor failed
- `vitual_fail_count`: the monitor's virtual failure count (number of failed checks - *`tolerance`*)
- `info`: the additional information the monitor recorded about its status
- `description`: description of what the monitor is checking

You will probably need to quote parameters to the command. For example:

```
fail_command=say "Oh no, monitor {name} has failed at {failed_at}"
```

The commands are executed directly by Python. If you require shell features, such as piping and redirection, you should use something like `bash -c "..."`. For example:

```
fail_command=/bin/bash -c "/usr/bin/printf \"The simplemonitor for {name} has failed on
↪{hostname}.\n\nTime: {failed_at}\nInfo: {info}\n\" | /usr/bin/mailx -A gmail -s \
↪"PROBLEM: simplemonitor {name} has failed on {hostname}.\" email@address"
```

4.3.5 nc - macOS notifications

Publishes alerts to the macOS Notification Center. Only for macOS.

No configuration options.

4.3.6 nextcloud_notification - notifications

> **Warning:** Do not commit your credentials to a public repo!

Send a notification to a Nextcloud server.

`token`

Type string

Required true

your nextcloud token

`user`

Type string

Required true

the admin user name

`server`

Type string

Required true

the nextcloud server

`server`

Type string

Required true

the user id who should receive the notification

timeout

Type int

Required false

Default 5

Timeout for HTTP request

4.3.7 pushbullet - push notifications

Warning: Do not commit your credentials to a public repo!

You will need to be registered at pushbullet.

token

Type string

Required true

your pushbullet token

timeout

Type int

Required false

Default 5

Timeout for HTTP request

4.3.8 pushover - notifications

Warning: Do not commit your credentials to a public repo!

You will need to be registered at pushover.

user

Type string

Required true

your pushover user key

token

Type string

Required true

your pushover app token

timeout

Type int

Required false

Default 5

Timeout for HTTP request

4.3.9 ses - email via Amazon Simple Email Service

> **Warning:** Do not commit your credentials to a public repo!

If you have AWS credentials configured elsewhere (e.g. in `~/.aws/credentials`), or in the environment, this will use those and you do not need to specifiy credentials in your configuration file.

As a best practice, use an IAM User/Role which is only allowed to access the resources in use.

You will need to verify an address or domain.

`from`

Type string

Required true

the email address to send from

`to`

Type string

Required true

the email address to send to

`aws_region`

Type string

Required false

the AWS region to use (e.g. `eu-west-1`)

`aws_access_key`

Type string

Required false

the AWS access key to use

`aws_secret_access_key`

Type string

Required false

the AWS secret access key to use

4.3.10 slack - Slack webhook

> **Warning:** Do not commit your credentials to a public repo!

First, set up a webhook for this to use.

- Go to https://slack.com/apps/manage
- Add a new webhook
- Configure it to taste (channel, name, icon)
- Copy the webhook URL for your configuration below

`url`

Type string

Required true

the Slack webhook URL

`channel`

Type string

Required false

Default the channel configured on the webhook

the channel to send to

`username`

Type string

Required false

Default a username to send to

`timeout`

Type int

Required false

Default `5`

Timeout for HTTP request to Slack

4.3.11 sms77 - SMS via sms77

> **Warning:** Do not commit your credentials to a public repo!

Send SMSes via the SMS77 service.

`api_key`

Type string

Required true

your API key for SMS77

`target`

Type string

Required true

the target number to send to

`sender`

Type string

Required false

Default `SmplMntr`

the sender to use for the SMS

`timeout`

Type int

Required false

Default `5`

Timeout for HTTP request

4.3.12 sns - Amazon Simple Notification Service

Warning: Do not commit your credentials to a public repo!

If you have AWS credentials configured elsewhere (e.g. in `~/.aws/credentials`), or in the environment, this will use those and you do not need to specifiy credentials in your configuration file.

As a best practice, use an IAM User/Role which is only allowed to access the resources in use.

Note that not all regions with SNS also support sending SMS.

`topic`

Type string

Required yes, if `number` is not given

the ARN of the SNS topic to publish to. Specify this, or `number`, but not both.

`number`

Type string

Required yes, if `topic` is not given

the phone number to SMS. Give the number as country code then number, without a + or other international access code. For example, `447777123456` for a UK mobile. Specify this, or `topic`, but not both.

`aws_region`

Type string

Required false

the AWS region to use (e.g. `eu-west-1`)

`aws_access_key`

Type string

Required false

the AWS access key to use

`aws_secret_access_key`

Type string

Required false

the AWS secret access key to use

4.3.13 syslog - send to syslog

Syslog alerters have no additional configuration.

4.3.14 telegram - send to a chat

Warning: Do not commit your credentials to a public repo!

`token`

Type string

Required true

the token to access Telegram

`chat_id`

Type string

Required true

the chat id to send to

`timeout`

Type int

Required false

Default 5

Timeout for HTTP request to Telegram

4.3.15 twilio_sms - SMS via Twilio

> **Warning:** Do not commit your credentials to a public repo!

Send SMSes via the Twilio service.

`account_sid`

Type string

Required true

your account SID for Twilio

`auth_token`

Type string

Required true

your auth token for Twilio

`target`

Type string

Required true

the target number to send to. Format should be + followed by a country code and then the phone number

`sender`

Type string

Required false

Default `SmplMntr`

the sender to use for the SMS. Should be a number in the same format as the target parameter, or you may be able to use an alphanumberic ID.

CHAPTER

FIVE

LOGGER CONFIGURATION

Loggers record the state of every monitor after each interval.

Loggers are defined in the main configuration file, which by default is `monitor.ini`. The section name is the name of your logger, which you should then add to the `loggers` configuration value.

Contents

5.1 Common options

These options are common to all logger types.

`type`

Type string

Required true

the type of the logger; one of those in the list below.

`depend`

Type comma-separated list of string

Required false

Default none

a list of monitors this logger depends on. If any of them fail, no attempt will be made to log.

`groups`

Type comma-separated list of string

Required false

Default `default`

list of monitor groups this logger should record. Use the special value `_all` to match all groups. See the *group* setting for monitors.

`tz`

Type string

Required false

Default `UTC`

the timezone to convert date/times to

`dateformat`

Type string

Required false

Default `timestamp`

the date format to write for log lines. (Note that the timezone is controlled by the *tz* configuration value.) Accepted values are:

- `timestamp` (UNIX timestamp)
- `iso8601` (`YYYY-MM-DDTHH:MM:SS`)

`heartbeat`

Type bool

Required false

Default `false`

if set, the logger only logs for monitors which executed on an iteration. Intended to be combined with the *gap* property of a Monitor.

5.2 Loggers

Note: The `type` of the logger is the first word in its heading.

5.2.1 db - sqlite log of results

Logs results to a SQLite database. The results are written to a table named `results`.

If you want to have a SQLite snapshot of the current state of the monitors (not a log of results), see the *dbstatus* logger.

Automatically create the database schema.

`path`

Type string

Required true

the path to the database file to use

5.2.2 dbstatus - sqlite status snapshot

Stores a snapshot of monitor status in a SQLite database. The statuses are written to a table named `status`.

If you want to have a SQLite log of results (not a snapshot), see the *db* logger.

Automatically creates the database schema.

`path`

Type string

Required true

the path to the database file to use

5.2.3 html - HTML status page

> **Warning:** Do not commit your credentials to a public repo!

Writes an HTML status page. Can optionally display a map.

The supplied template includes JavaScript to notify you if the page either doesn't auto-refresh, or if SimpleMonitor has stopped updating it. This requires your machine running SimpleMonitor and the machine you are browsing from to agree on what the time is (timezone doesn't matter)! The template is written using Jinja2.

You can use the `upload_command` setting to specify a command to push the generated files to another location (e.g. a web server, an S3 bucket etc). I'd suggest putting the commands in a script and just specifying that script as the value for this setting.

`filename`

Type string

Required true

the html file to output. Will be stored in the `folder`

`folder`

Type string

Required false

Default `html`

the folder to write the output file(s) to. Must exist.

`copy_resources`

Type boolean

Required false

Default true

if true, copy supporting files (CSS, images, etc) to the `folder`

`source_folder`

Type string

Required false

the path to find the template and supporting files in. Defaults to those contained in the package. (In the package source, they are in `simplemonitor/html/`.)

`upload_command`

Type string

Required false

if set, a command to execute each time the output is updated to e.g. upload the files to an external webserver

`map`

Type boolean

Required false

set to true to enable the map display instead of the table. You must set the *gps* value on your Monitors for them to show up!

`map_start`

Type comma-separated list of float

Required false

three comma-separated values: the latitude the map display should start at, the longitude, and the zoom level. A good starting value for the zoom is probably between 10 and 15.

`map_token`

Type string

Required yes, if using the map

an API token for mapbox.com in order to make the map work

5.2.4 json - write JSON status file

Writes the status of monitors to a JSON file.

`filename`

Type string

Required true

the filename to write to

5.2.5 logfile - write a logfile

Writes a log file of the status of monitors.

The logfile format is:

```
datetime monitor-name: status; VFC=vfc (message) (execution-time)
```

where the fields have the following meanings:

datetime the datetime of the entry. Format is controlled by the `dateformat` configuration option.

monitor-name the name of the monitor

status either `ok` if the monitor succeeded, or `failed since YYYY-MM-DD HH:MM:SS`

vfc the virtual failure count: the number of failures of the monitor beyond its *tolerance*. Not present for **ok** lines.

message the message the monitor recorded as the reason for failure. Not present for **ok** lines.

execution-time the time the monitor took to execute its check

`filename`

Type string

Required true

the filename to write to. Rotating this file underneath SimpleMonitor will likely result to breakage. If you would like the logfile to rotate automatically based on size or age, see the *logfileng* logger.

`buffered`

Type boolean

Required false

Default true

disable to use unbuffered writes to the logfile, allowing it to be watched in real time. Otherwise, you will find that updates don't appear in the file immediately.

`only_failures`

Type boolean

Required false

Default false

set to have only monitor failures written to the log file (almost, but not quite, turning it into an alerter)

5.2.6 logfileng - write a logfile with rotation

Writes a log file of the status of monitors. Rotates and deletes old log files based on size or age.

The logfile format is:

```
datetime monitor-name: status; VFC=vfc (message) (execution-time)
```

where the fields have the following meanings:

datetime the datetime of the entry. Format is controlled by the `dateformat` configuration option.

monitor-name the name of the monitor

status either `ok` if the monitor succeeded, or `failed since YYYY-MM-DD HH:MM:SS`

vfc the virtual failure count: the number of failures of the monitor beyond its *tolerance*. Not present for **ok** lines.

message the message the monitor recorded as the reason for failure. Not present for **ok** lines.

execution-time the time the monitor took to execute its check

`filename`

Type string

Required true

the filename to write to. Rotated logs have either `.N` (where N is an incrementing number) or the date/time appended to the filename, depending on the rotation mode.

`rotation_type`

Type string

Required true

one of `time` or `size`

`when`

Type string

Required false

Default `h`

Only for rotation based on time. The units represented by `interval`. One of `s` for seconds, `m` for minutes, `h` for hours, or `d` for days

`interval`

Type integer

Required false

Default `1`

Only for rotation based on time. The number of `when` between file rotations.

`max_bytes`

Type *bytes*

Required yes, when rotation_type is `size`

the maximum log file size before it is rotated.

`backup_count`

Type integer

Required false

Default `1`

the number of old files to keep

`only_failures`

Type boolean

Required false

Default false

set to have only monitor failures written to the log file (almost, but not quite, turning it into an alerter)

5.2.7 mqtt - send to MQTT server

> **Warning:** Do not commit your credentials to a public repo!

Sends monitor status to an MQTT server. Supports Home Assistant specifics (see https://www.home-assistant.io/docs/mqtt/discovery/ for more information).

`host`

Type string

Required true

the hostname/IP to connect to

`port`

Type integer

Required false

Default `1883`

the port to connect on

`hass`

Type boolean

Required false

Default false

enable Home Assistant specific features for MQTT discovery

`topic`

Type string

Required false

Default see below

the MQTT topic to post to. By default, if `hass` is not enabled, uses `simplemonitor`, else `homeassistant/binary_sensor`

`username`

Type string

Required false

the username to use

`password`

Type string

Required false

the password to use

5.2.8 network - remote SimpleMonitor logging

> **Warning:** Do not commit your credentials to a public repo!

This logger is used to send status reports of all monitors to a remote instance. The remote instance must be configured to listen for connections. The `key` parameter is a shared secret used to generate a hash of the network traffic so the receiving instance knows to trust the data.

> **Warning:** Note that the traffic is not encrypted, just given a hash to validate it.

The remote instance will need the `remote`, `remote_port`, and `key` *configuration values* set.

If you want the remote instance to handle alerting for this instance's monitors, you need to set the *remote_alert* option on your monitors. This is a good candidate to go the `[defaults]` section of your monitors config file.

`host`

Type string

Required true

the remote hostname/IP to send to

`port`

Type string

Required true

the remote port to connect to

`key`

Type string

Required true

the shared secret to validate communications

5.2.9 seq - seq log server

Sends the status of monitors to a **seq** log server. See https://datalust.co for more information on Seq.

`endpoint`

Type string

Required true

the full URI for the endpoint on the seq server, for example `http://localhost:5341/api/events/seq`.

`timeout`

Type int

Required false

Default 5

Timeout for HTTP request to seq

CHAPTER

SIX

CREATING MONITORS

To create your own Monitor, you need to:

1. Create a Python file in `simplemonitor/Monitors` (or pick a suitable existing one to add it to)
2. If you're creating a new file, you'll need a couple of imports:

```
from .monitor import Monitor, register
```

3. Define your monitor class, which should subclass `Monitor` and be decorated by `@register`. Set a class attribute for the "type" which will be used in the monitor configuration to use it.

```
@register
class MonitorMyThing(Monitor):

    monitor_type = "my_thing"
```

4. Define your initialiser. It should call the superclass's initialiser, and then read its configuration values from the supplied dict. You can also do any other initialisation here.

   This code should be safe to re-run, as if SimpleMonitor reloads its configuration, it will call `__init__()` with the new configuration dict. Use the *`get_config_option()`* helper to read config values.

```
@register
class MonitorMyThing(Monitor):

    monitor_type = "my_thing"

    def __init__(self, name: str, config_options: dict) -> None:
        super().__init__(name, config_options)
        self.my_setting = self.get_config_option("my_setting", required=True)
```

5. Add a `run_test` function. This should perform the test for your monitor, and call `record_fail()` or `record_success()` as appropriate. It must also return `False` or `True` to match. The two `record_*()` methods return the right value, so you can just use them as the value to `return`. You can use `self.monitor_logger` to perform logging (it's a standard Python `logging` object).

   You should catch any suitable exceptions and handle them as a failure of the monitor. The main loop will handle any uncaught exceptions and fail the monitor with a generic message.

```
@register
class MonitorMyThing(Monitor):

    # ...
```

(continues on next page)

(continued from previous page)

```
    def run_test(self) -> bool:
        # my test logic here
        if test_succeeded:
            return self.record_success("it worked")
        return self.record_fail(f"failed with message {test_result}")
```

6. You should also give a `describe` function, which explains what this monitor is checking for:

```
@register
class MonitorMyThing(Monitor):

    # ...

    def describe(self) -> str:
        return f"checking that thing f{my_setting} does foo"
```

7. In `simplemonitor/Monitors/__init__.py`, add your Monitor to the list of imports.

That's it! You should now be able to use `type=my_thing` in your Monitors configuration to use your monitor.

If you'd like to share your monitor back via a PR, please also:

1. Use type decorators, and verify with mypy. You may need to use `cast(TYPE, self.get_config_option(...))` in your `__init__()` to get things to settle down. See existing monitors for examples.
2. Use Black to format the code.
3. Add documentation for your monitor. Create a file in *docs/monitors/* called *my_thing.rst* and follow the pattern in the other files to document it.

There's a pre-commit configuration in the repo which you can use to check things over.

CHAPTER

SEVEN

CREATING ALERTERS

To create your own Alerter, you need to:

1. Create a Python file in `simplemonitor/Alerters` (or pick a suitable existing one to add it to)
2. If you're creating a new file, you'll need a couple of imports:

```
from ..Monitors.monitor import Monitor
from .alerter import Alerter, AlertLength, AlertType, register
```

3. Define your alerter class, which should subclass `Alerter` and be decorated by `@register`. Set a class attribute for the "type" which will be used in the alerter configuration to use it.

```
@register
class MyAlerter(Alerter):

    alerter_type = "my_alerter"
```

4. Define your initialiser. It should call the superclass's initialiser, and then read its configuration values from the supplied dict. You can also do any other initialisation here.

   This code should be safe to re-run, as if SimpleMonitor reloads its configuration, it will call `__init__()` with the new configuration dict. Use the *get_config_option()* helper to read config values.

```
@register
class MyAlerter(Alerter):

    alerter_type = "my_alerter"

    def __init__(self, config_options: dict) -> None:
        super().__init__(config_options)
        self.my_setting = self.get_config_option("setting", required=True)
```

5. Add a `send_alerter` function. This receives the information for a single monitor. You should first call `self.should_alert(monitor)`, which will return the type of alert to be sent (e.g. failure). You should return if it is `AlertType.NONE`.

   You should then prepare your message. Call `self.build_message()` to generate the message content. Check the value of `self._dry_run` and if it is True, you should log (using `self.alerter_logger.info(...)`) what you would do, else you should do it.

   `Alerter.`**`build_message`**`(`*length: AlertLength*, *alert_type: AlertType*, *monitor: Monitor*`)` → str

   Generate a suitable length alert message for the given type of alert, for the given Monitor.

   Parameters

- **AlertLength** – one of the AlertLength enum values: NOTIFICATION (shortest), SMS (will be <= 140 chars), ONELINE, TERSE (not currently supported), FULL, or ESSAY
- **AlertType** – one of the AlertType enum values: NONE, FAILURE, CATCHUP, or SUCCESS
- **monitor** – the Monitor to generate the message for

Returns the built message

Return type str

Raises

- **ValueError** – if the AlertType is unknown
- **NotImplementedError** – if the AlertLength is unknown or unsupported

7. You should also give a `_describe_action` function, which explains what this alerter does. Note that the time configuration for the alerter will be automatically added:

```
@register
class MyAlerter(Alerter):

    # ...

    def _describe_action(self) -> str:
        return f"sending FooAlerters to {self.recipient}"
```

7. In `simplemonitor/Alerters/__init__.py`, add your Alerter to the list of imports.

That's it! You should now be able to use `type=my_alerter` in your Alerters configuration to use your alerter.

CHAPTER

EIGHT

CREATING LOGGERS

Before writing your logger, you need to consider if you should support **batching** or not. If a logger supports batching, then it collects all the monitor results and then performs its logging action. For example, the HTML logger uses batching so that when it generates the HTML output, it knows all the monitors to include (and can sort them etc). Non-batching loggers will simply perform their logging action multiple times, once per monitor.

To create your own Logger, you need to:

1. Create a Python file in `simplemonitor/Loggers` (or pick a suitable existing one to add it to)
2. If you're creating a new file, you'll need a couple of imports:

```
from ..Monitors.monitor import Monitor
from .logger import Logger, register
```

3. Define your logger class, which should subclass `Logger` and be decorated by `@register`. Set a class attribute for the "type" which will be used in the logger configuration to use it. Additionally, set the `supports_batch` value to indicate if your logger should be used in batching mode.

```
@register
class MyLogger(Logger):

    logger_type = "my_logger"
    supports_batch = True  # or False
```

4. Define your initialiser. It should call the superclass's initialiser, and then read its configuration values from the supplied dict. You can also do any other initialisation here.

   This code should be safe to re-run, as if SimpleMonitor reloads its configuration, it will call `__init__()` with the new configuration dict. Use the *get_config_option()* helper to read config values.

```
@register
class MyLogger(Logger):

    logger_type = "my_logger"

    def __init__(self, config_options: dict) -> None:
        super().__init__(config_options)
        self.my_setting = self.get_config_option("setting", required=True)
```

5. Add a `save_result2` function (yes, I know). This receives the information for a single monitor.

   Batching loggers should save the information they need to into *self.batch_data*, which should (but does not have to be) a dict of *str: Any* using the monitor name as the key. This is automatically initialised to an empty dict at the start of the batch. You should extend the *start_batch* method from *Logger* to customise it.

```
@register
class MyLogger(Logger):

    # ...

    def save_result2(self, name: str, monitor: Monitor) -> None:
        self.batch_data[name] = monitor.state
```

Non-batching loggers can perform whatever logging action they are designed for at this point.

```
@register
class MyLogger(Logger):

    # ...

    def save_result2(self, name: str, monitor: Monitor) -> None:
        self._my_logger_action(f"Monitor {name} is in state {monitor.state}")
```

6. **Batching loggers** only should provide a `process_batch` method, which is called after all the monitors have been processed. This is where you should perform your batched logging operation.

```
@register
class MyLogger(Logger):

    # ...

    def process_batch(self) -> None:
        with open(self.filename, "w") as file_handle:
            for monitor, state in self.batch_data.iteritems():
            file_handle.write(f"Monitor {monitor} is in state {state}\n")
```

7. You should also give a `describe` function, which explains what this logger does:

```
@register
class MyLogger(Logger):

    # ...

    def describe(self) -> str:
        return f"writing monitor info to {self.filename}"
```

7. In `simplemonitor/Loggers/__init__.py`, add your Logger to the list of imports.

That's it! You should now be able to use `type=my_thing` in your Loggers configuration to use your logger.

CHAPTER

NINE

GETTING CONFIGURATION VALUES

When loading configuration values for Monitors, Alerters and Loggers, you can use the *get_config_option()* function to perform sanity checks on the loaded config.

get_config_option(*config_options: dict*, *key: str*[, *default=None*[, *required=False*[, *required_type="str"*[, *allowed_values=None*[, *allow_empty=True*[, *mininum=None*[, *maximum=None*]]]]]]])

Get a config value out of a dict, and perform basic validation on it.

Parameters

- **config_options** (*dict*) – The dict to get the value from
- **key** (*str*) – The key to the value in the dict
- **default** – The default value to return if the key is not found
- **required** (*bool*) – Throw an exception if the value is not present (and default is None)
- **required_type** (*str*) – One of str, int, float, bool, [int] (list of int), [str] (list of str)
- **allowed_values** – A list of allowed values
- **allow_empty** (*bool*) – Allow the empty string when required_type is "str"
- **minimum** (*integer, float or None*) – The minimum allowed value for int and float
- **maximum** (*integer, float or None*) – The maximum allowed value for int and float

Returns the fetched configuration value (or the default)

Note that the return type of the function signature covers all supported types, so you should use `typing.cast()` to help mypy understand. Do not use assert.

SimpleMonitor is a Python script which monitors hosts and network connectivity and status. It is designed to be quick and easy to set up and lacks complex features that can make things like Nagios, OpenNMS and Zenoss overkill for a small business or home network. Remote monitor instances can send their results back to a central location.

SimpleMonitor supports Python 3.6.2 and higher on Windows, Linux and FreeBSD.

To get started, see *Installation*.

CHAPTER

TEN

FEATURES

10.1 Things SimpleMonitor can monitor

For the complete list, see *Monitors*.

- Host ping
- Host open ports (TCP)
- HTTP (is a URL fetchable without error? Does the page content it match a regular expression?)
- DNS record return value
- Services: Windows, Linux, FreeBSD services are supported
- Disk space
- File existence, age and time
- FreeBSD portaudit (and pkg audit) for security notifications
- Load average
- Process existence
- Exim queue size monitoring
- APC UPS monitoring (requires apcupsd to be installed and configured)
- Running an arbitrary command and checking the output
- Compound monitors to combine any other types

Adding your own Monitor type is straightforward with a bit of Python knowledge.

10.2 Logging and Alerting

To SimpleMonitor, a Logger is something which reports the status of every monitor, each time it's checked. An Alerter sends a message about a monitor changing state.

Some of the options include (for the complete list, see *Loggers* and *Alerters*):

- Writing the state of each monitor at each iteration to a SQLite database
- Sending an email alert when a monitor fails, and when it recovers, directly over SMTP or via Amazon SES
- Writing a log file of all successes and failures, or just failures

- Sending a message via BulkSMS, Amazon Simple Notification Service (SNS), Telegram, Slack, MQTT (with HomeBridge support) and more
- Writing an HTML status page
- Writing an entry to the syslog (non-Windows only)
- Executing arbitary commands on monitor failure and recovery

10.3 Other features

- Simple configuration file format: it's a standard INI file for the overall configuration and another for the monitor definitions
- Remote monitors: An instance running on a remote machine can send its results back to a central instance for central logging and alerting
- Dependencies: Monitors can be declared as depending on the success of others. If a monitor fails, its dependencies will be skipped until it succeeds
- Tolerance: Monitors checking things the other side of unreliable links or which have many transient failures can be configured to require their test to fail a number of times in a row before they report a problem
- Escalation of alerts: Alerters can be configured to require a monitor to fail a number of times in a row (after its tolerance limit) before they fire, so alerts can be sent to additional addresses or people
- Urgency: Monitors can be defined as non-urgent so that urgent alerting methods (like SMS) are not wasted on them
- Per-host monitors: Define a monitor which should only run on a particular host and all other hosts will ignore it – so you can share one configuration file between all your hosts
- Groups: Configure some Alerters to only react to some monitors
- Monitor gaps: By default every monitor polls every interval (e.g. 60 seconds). Monitors can be given a gap between polls so that they only poll once a day (for example)
- Alert periods: Alerters can be configured to only alert during certain times and/or on certain days
- Alert catchup: . . . and also to alert you to a monitor which failed when they were unable to tell you. (For example, I don't want to be woken up overnight by an SMS, but if something's still broken I'd like an SMS at 7am as I'm getting up.)

CHAPTER

ELEVEN

CONTRIBUTING

- Clone the GitHub repo
- `poetry install`

You can use pre-commit to ensure your code is up to my exacting standards ;)

You can run tests with `make unit-test`. See the Makefile for other useful targets.

CHAPTER

TWELVE

LICENCE

SimpleMonitor is released under the BSD Licence.

CHAPTER

THIRTEEN

CONTACT

- Open an issue or start a discussion on GitHub
- Twitter: @jamesoff
- Email: james at jamesoff dot net

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- search

INDEX

D

P

R

S

T

U

V

W