jw.mailfilter Documentation

Release 0.2

Johnny Wezel

Sep 27, 2017

Contents

1 Installation **3**

2 Configuration **5**
2.1 Account definition . . . 6

3 Filters **7**
3.1 Filter specification . . . 7
3.2 Filter lists . . . 7
3.3 Nested filter lists . . . 8
3.4 Action routines . . . 9
3.5 Action clause . . . 9
3.6 Built-in filters . . . 9
3.6.1 Filter: header . . . 9
3.6.2 Filter: body . . . 9
3.6.3 Filter: rbl . . . 10
3.6.4 Filter: pyzor . . . 10
3.6.5 Filter: url . . . 10
3.6.6 Filter: all . . . 10
3.6.7 The check argument in header and body filters . . . 11

4 Actions **13**
4.1 Action: copy-to . . . 13
4.2 Action: move-to . . . 13
4.3 Action: delete . . . 13
4.4 Action: set-flag . . . 14
4.5 Action: clear-flag . . . 14
4.6 Action: report-pyzor . . . 14
4.7 Action: report-badips . . . 14
4.8 Action: forward-to . . . 14
4.9 Action: stop . . . 15

5 Logging **17**

6 YAML **19**
6.1 Values . . . 19
6.1.1 String . . . 19
6.1.2 Integer . . . 20

6.1.3 Floating point . . . 20
6.1.4 Boolean . . . 20
6.1.5 Null value . . . 20
6.2 Lists . . . 20
6.2.1 Nested lists . . . 21
6.3 Maps . . . 21
6.3.1 Nested maps . . . 22
6.4 Nesting lists and maps . . . 22
6.5 Quoting . . . 22

7 Indices and tables **25**

Contents:

CHAPTER 1

Installation

The package is registered on pypi.python.org, so the usual

```
pip install jw.mailfilter
```

will do.

CHAPTER 2

Configuration

Configuration files are written in YAML. A short introduction to the YAML syntax is given in the chapter YAML Syntax. A minimal configuration file would look like this:

```
accounts:
    Google Mail:
        address: imap.gmail.com
        username: john.doe@gmail.com
        password: _My_PaSswOrd_
        ssl: true
        time-out: 300
        idle: on
        folders:
            - [INBOX, {filter: rbl, org: bl.spamcop.net, action: delete}]
        smtp:
            host: smtp.gmail.com

logging:
    version: 1
    formatters:
        default:
            format: '%(asctime)s:%(name)-32s:%(funcName)-16s:%(levelname)-8s:
↪%(message)s'
    handlers:
        mailfilter:
            class: logging.handlers.RotatingFileHandler
            formatter: default
            # FILENAME:
            filename: _test/log
            backupCount: 4
            maxBytes: 4194304
    loggers:
        mailfilter:
            handlers: [mailfilter]
            # LOG LEVEL:
            level: INFO
```

As you see there are two sections, the latter one only concerned about logging. More about that in the logging chapter.

The main section for defining IMAP accounts and what should happen with them is defined in the *accounts* section.

Account definition

The *address* item specifies the domain name of the IMAP server, *username* and *password* the account credentials. If SSL is to be used, an *ssl* item may be given with a value of *true*, *yes* or *on*. If it is missing, *false* is assumed.

A *time-out* item, expressed in seconds, specifies how often to poll the account. A *time-out* value can also be specified in the global section.

If the server provides *IDLE* push notifications, *idle* with a *True* value can be added. If *idle* is missing, *False* is assumed.

The folders to be examined are specified in the *folders* item as a list. Each item is a list consisting of a folder name and a filter specification.

Forwarding an email requires access to an SMTP server. The *smtp* section in the account requires a *host* item specifying the server's domain name. Optionally, a *username* and *password* item may be given if they are different from the ones in the account definition. An *smtp* section can also be put into the global section, which will provide a default specification.

CHAPTER 3

Filters

A filter specification defines what conditions have to be met and what actions are taken when a condition is met.

Filter specification

A filter specification is usually written as for example:

```
{filter: rbl, org: bl.spamcop.net, action: delete}
```

where the *filter* clause specifies the name of the filter to apply, in this case "rbl", a filter that checks whether the sending mail server's IP is registered as spam source. The remaining clauses are parameters for the filter. The *action* clause is mandatory with all filters. In the example, the *org* clause, required by *rbl* filters, specifies which RBL service is to be queried.

Filter lists

Since a filter definition can easily occupy a whole line, it is more convenient to put each of them on its own line when specifying multuple filters for a folder. For this YAML's block style is better suited to the task. So a list of filters would look like

```
folders:
    - INBOX
    -
        - {filter: header, part: subject, check: "has cana?dia?n\s+pha?rma?cy",␣
↪action: delete}
        - {filter: header, part: from, check: "is-not .*@goodguy.org", action: delete}
        - {filter: header, part: subject, check: "is \[mail-list\].*", action: move-
↪to mail-list}
```

But nothing prevents you from writing the above list in flow style, for example:

```
folders: [INBOX, [
        {filter: header, part: subject, check: "has cana?dia?n\s+pha?rma?cy", action:␣
↪delete},
        {filter: header, part: from, check: "is-not .*@goodguy.org", action: delete},
        {filter: header, part: subject, check: "is \[mail-list\].*", action: move-to␣
↪mail-list}]]
```

Nested filter lists

YAML provides a good deal of flexibility with data definitions. One particularly useful feature is anchors. To make use of these, filter lists can be nested. Together, easily switchable alternatives can be defined, for example:

```
my-alternatives-list:
    - &alternative1
        - {filter: header, part: subject, check: "has cana?dia?n\s+pha?rma?cy",␣
↪action: delete}
        - {filter: header, part: from, check: "is-not .*@goodguy.org", action: delete}
        - {filter: header, part: subject, check: "is \[mail-list\].*", action: move-
↪to mail-list}
    - &alternative2
        - {filter: header, part: subject, check: "has cana?dia?n\s+pha?rma?cy",␣
↪action: move-to INBOX/Spam}
        - {filter: header, part: from, check: "is-not .*@goodguy.org", action: move-
↪to INBOX/Spam}
        - {filter: header, part: subject, check: "is \[mail-list\].*", action: move-
↪to INBOX/Spam}

accounts:
    Google Mail:
        address: imap.gmail.com
        username: john.doe@gmail.com
        password: _My_PaSswOrd_
        ssl: true
        folders:
            - [INBOX, *alternative1]
```

Thus the filter rules in the *Google Mail* account can be switched quickly from *alternative1* to *alternative2*. The filter rule is seen by the program as a list of lists which, in itself, doesn't make much sense. It's only there to make use of YAML's anchor feature. The possibilities are enless. Consider:

```
my-sub-alternatives:
    - &all-rules
        - &special-rule
            - {filter: header, part: subject, check: "has cana?dia?n\s+pha?rma?cy",␣
↪action: delete}
        - &main-rules:
            - {filter: header, part: from, check: "is-not .*@goodguy.org", action:␣
↪move-to INBOX/Spam}
            - {filter: header, part: subject, check: "is \[mail-list\].*", action:␣
↪move-to INBOX/Spam}
```

You can switch between *all-rules*, *main-rules* and *special-rule* in an instant.

Action routines

Notice the repetition in the the previous examples' action arguments. It would get worse if a list of actions should be run. Fortunately, YAML anchors provide the solution to this problem as well:

```
my-actions:
    - report-pyzor
    - report-badips
    - delete

my-rules:
    - {filter: header, part: from, check: "is-not .*@goodguy.org", action: *my-
↪actions}
    - {filter: header, part: subject, check: "is \[mail-list\].*", action: *my-
↪actions}
```

Action clause

The *action* clause, required by all filters, may be a single item or a list of items. Since it is the same for all filters, it is not mentioned in descriptions below. See Actions for a description of actions.

Built-in filters

The following filters are included in this packages. More filters can be added through extensions.

Filter: header

Arguments: *part*, *check*

The *part* argument specifies which header is to be examined. Common values are *from* (sender), *to* (recipient), *subject*, *date* (actually the time), but generally, anything occurring in the header section of a mail is possible.

For an explanation of the *check* argument, see *The check argument in header and body filters*.

Examples:

```
{filter: header, part: subject, check: "has cana?dia?n\s+pha?rma?cy", action: delete}
{filter: header, part: from, check: "is-not .*@goodguy.org", action: delete}
{filter: header, part: subject, check: "is \[mail-list\].*", action: move-to mail-
↪list}
```

Filter: body

Arguments: *check*

Checks for contents in the body of the e-mail. For an explanation of the *check* argument, see *The check argument in header and body filters*.

Examples:

```
{filter: body, check: "has cana?dia?n\s+pha?rma?cy", action: move-to Spam}
```

Note: The *is* and *is-not* operators are less useful in the *body* filter. Use *has* instead.

Filter: rbl

Arguments: *org*

Checks whether the IP of the sending server is registered with an RBL (real-time block list). There is a vast number of organizations providing such black lists. The *org* parameter specifies which one is to be used. These black lists answer domain name requests in a specific format and the *org* parameter specifies the right part of it. Here is a list with the values for some of the most prominent RBLs in use today:

RBL organization	org value
Spamcop	bl.spamcop.net
Spamhaus	zen.spamhaus.org
Barracuda	b.barracudacentral.org

Examples:

```
{filter: rbl, org: bl.spamcop.net, action: delete}
```

Filter: pyzor

Checks whether there is a signature of the whole mail message registered with Pyzor. These signatures designate spam messages previously reported to Pyzor.

Examples:

```
{filter: pyzor, action: delete}
```

Filter: url

Checks whether the mail contains URLs known as spam source.

Examples:

```
{filter: url, action: delete}
```

Filter: all

This is not exactly a filter. It applies actions to all messages encountered during the scan process.

Examples:

```
{filter: all, action: [report-badips, report-pyzor, delete]}
```

Note: Be careful with this filter. It is intended to be used on folders where messages are copied or moved under some control. Do not use it on the INBOX folder (except if the account is a honey pot anyway).

The check argument in header and body filters

The *check* argument specifies how to compare the value. The format is *operator operands*, where *operator* is one of: *is*, *is-not*, *has*, *has-no* or *has-all*. For *operands*, a single value or a list of values can be given, depending on the requirements of the operator.

The *is* operator does a regular expression match of a single value against the pattern in the operand. The *is-not* operator succeeds if the pattern does not match.

To search for a regular expression anywhere in the header or body, use the *has* operator. If you want to check against more than one pattern and you want to make sure *all* of them are found, use *has-all*. If you want to make sure none of a number of patterns are found, use *has-no*.

Note: Since regular expressions tend to use characters also used by YAML, it is a good idea to quote the value.

CHAPTER 4

Actions

Actions are specified with the *action* argument to a Filter definition. The *action* parameter takes either a single action or a list of actions, for example:

```
action: delete

action: [report-pyzor, forward-to abuse@godaddy.com, delete]
```

The follwing actions are built-in. More actions can be added through plugins.

Action: copy-to

Copies the message to a folder. The folder is created if it does not exist.

Example:

```
action: copy-to INBOX/Spam
```

Action: move-to

Moves the message to a folder. The folder is created if it does not exist.

Example:

```
action: move-to INBOX/Business
```

Action: delete

Deletes the message

Example:

```
action: delete
```

Action: set-flag

Sets an IMAP flag. The set of flags depends on the server. Probably the most minimal set of flags is: `\Deleted`, `\Seen` and `\Answered`. Flags supported by other servers are: `\Draft`, `\Flagged` and even user defined flags. Remember that the backslash `\` needs to be doubled because it is itself used to quote single characters.

Example:

```
action: set-flag \\Flagged
```

Action: clear-flag

Clears an IMAP flag. See *set-flag*.

Example:

```
action: clear-flag \\Deleted
```

Action: report-pyzor

Reports the message to Pyzor as spam. The pyzor filter will then detect the same message as spam.

Example:

```
action: report-pyzor
```

Action: report-badips

Reports the sending server's IP to badips.com as spammer.

Example:

```
action: report-badips
```

Action: forward-to

Forwards the message to another e-mail address. For this, an SMTP account has to be defined in **'global-parameters'_**.

Example:

```
action: forward-to abuse@spamdomain.com
```

Action: stop

Stop processing further filters for this message. This is only useful in combination with nested filter lists.

Example:

```
action: stop
```

CHAPTER 5

Logging

The logging section in the configuration file specifies how logging should work.

For now, only two important settings are discussed here. An example of a *logging* section:

```
accounts:
    Google Mail:
        address: imap.gmail.com
        username: john.doe@gmail.com
        password: _My_PaSswOrd_
        ssl: true
        folders:
            - [INBOX, {filter: rbl, org: bl.spamcop.net, action: delete}]

logging:
    version: 1
    formatters:
        default:
            format: '%(asctime)s:%(name)-32s:%(funcName)-16s:%(levelname)-8s:
↪%(message)s'
    handlers:
        mailfilter:
            class: logging.handlers.RotatingFileHandler
            formatter: default
            # FILENAME:
            filename: _test/log
            backupCount: 4
            maxBytes: 4194304
    loggers:
        mailfilter:
            handlers: [mailfilter]
            # LOG LEVEL:
            level: INFO
```

The *filename* entry specifies the path of the log file. The *level* entry specifies the log level. Valid log levels are: *DEBUG*, *INFO*, *WARNING*, *ERROR* and *CRITICAL*.

Note: The *logging* section is the YAML equivalent of the dict to be supplied to the logging.config.dictConfig() function. For a complete explanation of the *logging* section, see the Logging facility for Python.

CHAPTER 6

YAML

YAML is a data description language similar to JSON, but with human readability in mind.

There are two fundamental data container types making up a data structure: *lists* and *maps*. All other data is scalar, meaning a single data items.

A *list* is a sequence of data items, one after another. *Maps* are an unordered set of data items with an associated key. Since lists and maps can be nested, they themselves become data items. In YAML, both can be written in two notations: either the *block* style or the *flow* style. The two styles can be chosen for every data item indivdually and can be mixed freely.

Values

Simple values (scalar values) can have the following types:

- string
- integer
- floating point
- boolean
- null value

String

A sequence of characters is normally taken as a string. However, if a different data type is detected, that data type is returned to the program. To force a sequence of characters to be interpreted as a string, it can be quoted with either single quote or double quotes:

```
- '123' # would be taken as integer otherwise
- "2.4" # would be taken as floating point value otherwise
- 'yes' # would be taken as a boolean otherwise
```

Strings containing the following characters have to be quoted: *[ ] { } ~ & * | > ! # @ % ‘*.

Integer

A number without decimal point:

```
- 200
- -123
```

Floating point

A number with decimal point:

```
- 1.2
- -3.4
```

Boolean

Either a value for true:

```
- true
- yes
- on
```

or a value for false:

```
- false
- no
- off
```

Null value

One of these:

```
- ~
- null
```

Lists

A *list* in *block* style:

```
- item 1
- item 2
- item 3
```

A *list* in *flow* style:

```
[item 1, item 2, item3]
```

or:

```
[
    item 1,
    item 2,
    item 3
]
```

In *flow* style, tokens can be formatted freely with white space, where as in *block* style, the line format must be kept.

Nested lists

Lists can be nested:

```
-
    - Item 1a
    - Item 1b
-
    - Item 2a
    - Item 2b
```

or:

```
[
    [Item 1b, Item 1b],
    [Item 2a, Item 2b]
]
```

or a mix of *block* and *flow* style:

```
- [Item 1a, Item 1b]
- [Item 2a, Item 2b]
```

Maps

A *map* in *block* style:

```
First name: Jack
Last name: Miller
Email: jack.miller@gmail.com
```

A *map* in *flow* style:

```
{First name: Jack, Last name: Miller, Email: jack.miller@gmail.com}
```

Again, as with lists in *flow* style, white space can be used to format the data:

```
{
    First name: Jack,
    Last name: Miller,
    Email: jack.miller@gmail.com
}
```

Nested maps

Nested maps in *block* style:

```
Key1: Value1
Key2:
    Key2a: Value2a
    Key2b: Value2b
```

In *flow* style:

```
{
    Key1: Value1,
    Key2: {
        Key2a: Value2a,
        Key2b, Value2b
    }
}
```

Nesting lists and maps

A list of maps in *block* style:

```
-
    Key1a: Value1a
    Key1b: Value1b
-
    Key2a: Value2a
    Key2b: Value2b
```

In *flow* style:

```
[{Key1a: Value1a, Key1b: Value1b}, {Key2a: Value2a, Key2b: Value2b}]
```

A map of lists in *block* style:

```
Key1:
    - Item 1a
    - Item 1b
Key2:
    - Item 2a
    - Item 2b
```

In *flow style*:

```
{Key1: [Item 1a, Item 1b], Key2: [Item 2a, Item 2b]}
```

Quoting

Certain characters are used in YAML to make up the syntax for data definitions. Also, certains words are used to denote certain data values. These are: *[ ] { } , : -* and *~*. Reserved words are: *null*, *true*, *false*, *yes*, *no*, *on* and *off*.

When writing data items containing the reserved characters or as the character sequence of the reserved words, a data value has to be written in either single or double quotes:

```
- "Item [1]"
- 'Item {2}'
- 'null'
- 'yes'
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search