
PyMailq Documentation

Release 0.9.0

Denis 'jawa' Pompilio

Feb 15, 2018

Contents

1	pymailq.store – Mails queue storage objects	3
2	pymailq.selector – Mails queue filtering	9
3	pymailq.control – Mails queue administrative operations	13
4	pymailq.shell – Mails queue management shell	15
5	pymailq.CONFIG – Configuration structure and usage	17
6	pqshell – A shell-like to interact with a Postfix mails queue	19
	Python Module Index	25

The *pymailq* package makes it easy to view and control Postfix mails queue. It provide several classes to store, view and interact with mail queue using Postfix command line tools. This module is provided for automation and monitoring.

The *pymailq* package defines the following attribute:

```
pymailq.DEBUG = False
    Boolean to control activation of the debug() decorator.

pymailq.VERSION = '0.9.0'
    Current version of the package as str.

pymailq.CONFIG = dict()
    Module configuration as dict.
```

The *pymailq* package defines the following decorators:

```
pymailq.debug(function)
    Decorator to print some call informations and timing debug on stderr.

    Function's name, passed args and kwargs are printed to stderr. Elapsed time is also print at the end of call. This decorator is based on the value of DEBUG. If True, the debug informations will be displayed.

pymailq.load_config(cfg_file)
    Load module configuration from .ini file

    Information from this file are directly used to override values stored in pymailq.CONFIG.

    Commands from configuration file are treated using shlex.split() to properly transform command string to list of arguments.

    Parameters cfg_file (str) – Configuration file
```

See also:

pymailq.CONFIG – Configuration structure and usage

The *pymailq* package provides the following submodules:

pymailq.store – Mails queue storage objects

The `store` module provide several objects to convert mails queue content into python structures.

1.1 PostqueueStore Objects

class `pymailq.store.PostqueueStore`

Postfix mails queue informations storage.

The `PostqueueStore` provides methods to load Postfix queued mails informations into Python structures. Thoses structures are based on `Mail` and `MailHeaders` classes which can be processed by a `MailSelector` instance.

The `PostqueueStore` class defines the following attributes:

mails

Loaded `MailClass` objects `list()`.

loaded_at

`datetime.datetime` instance to store load date and time informations, useful for datas deprecation tracking. Updated on `load()` call with `datetime.datetime.now()` method.

postqueue_cmd

`list` object to store Postfix command and arguments to view the mails queue content. This property use Postfix mails content parsing command defined in `pymailq.CONFIG` attribute under the key 'list_queue'. Command and arguments list is build on call with the configuration data.

spool_path

Postfix spool path string. Default is `"/var/spool/postfix"`.

postqueue_mailstatus

Postfix known queued mail status list. Default is `['active', 'deferred', 'hold']`.

mail_id_re

Python compiled regular expression object (`re.RegexObject`) provided by `re.compile()` method to match postfix IDs. Recognized IDs are either:

- hexadecimals, 8 to 12 chars length (regular queue IDs)
- encoded in a 52-character alphabet, 11 to 16 chars length (long queue IDs)

They can be followed with `*` or `!`. Default used regular expression is:

```
r"^( [A-F0-9]{8,12} | [B-Zb-z0-9]{11,16} ) [*!]? $"
```

mail_addr_re

Python compiled regular expression object (`re.RegexObject`) provided by `re.compile()` method to match email addresses. Default used regular expression is:

```
r"^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]+$"
```

MailClass

The class used to manipulate/parse mails individually. Default is `Mail`.

See also:

Python modules: `datetime` – Basic date and time types

`re` – Regular expression operations

Postfix manual: `postqueue` – Postfix queue control

RFC 3696 – Checking and Transformation of Names

The `PostqueueStore` instance provides the following methods:

`PostqueueStore.load([method])`

Load content from postfix mails queue.

Mails are loaded using `postqueue` command line tool or reading directly from spool. The optionnal argument, if present, is a method string and specifies the method used to gather mails informations. By default, method is set to `"postqueue"` and the standard Postfix queue control tool: `postqueue` is used.

Optionnal argument `parse` controls whether mails are parsed or not. This is useful to load every known mail headers for later filtering.

Parameters

- **method** (*str*) – Method used to load mails from Postfix queue
- **filename** (*str*) – File to load mails from
- **parse** (*bool*) – Controls whether loaded mails are parsed or not.

Provided method `str()` name is directly used with `getattr()` to find a `self.load_from_<method>` method.

`PostqueueStore._load_from_postqueue()`

Load content from postfix queue using `postqueue` command output.

Output lines from `_get_postqueue_output` are parsed to build `Mail` objects. Sample Postfix queue control tool (`postqueue`) output:

```
C0004979687      4769 Tue Apr 29 06:35:05  sender@domain.com
(error message from mx.remote1.org with parenthesis)
                                first.rcpt@remote1.org
(error message from mx.remote2.org with parenthesis)
                                second.rcpt@remote2.org
                                third.rcpt@remote2.org
```

Parsing rules are pretty simple:

- Line starts with a valid `Mail.qid`: create new `Mail` object with `qid`, `size`, `date` and `sender` informations from line.

Queue ID	Size	Reception date and time				Sender
C0004979687	4769	Tue	Apr	29	06:35:05	user@domain.com

- Line starts with a parenthesis: store error messages to last created `Mail` object's `errors` attribute.
- Any other matches: add new recipient to the `recipients` attribute of the last created `Mail` object.

Optionnal argument `filename` can be set with a file containing output of the `postqueue` command. In this case, output lines of `postqueue` command are directly read from `filename` and parsed, the `postqueue` command is never used.

Optionnal argument `parse` controls whether mails are parsed or not. This is useful to load every known mail headers for later filtering.

Parameters

- **filename** (*str*) – File to load mails from
- **parse** (*bool*) – Controls whether loaded mails are parsed or not.

`PostqueueStore._load_from_spool()`

Load content from postfix queue using files from spool.

Mails are loaded using the command defined in `postqueue_cmd` attribute. Some informations may be missing using the `_load_from_spool()` method, including at least `Mail.status` field.

Optionnal argument `parse` controls whether mails are parsed or not. This is useful to load every known mail headers for later filtering.

Loaded mails are stored as `Mail` objects in `mails` attribute.

Parameters **parse** (*bool*) – Controls whether loaded mails are parsed or not.

Warning: Be aware that parsing mails on disk is slow and can lead to high load usage on system with large mails queue.

`PostqueueStore._get_postqueue_output()`

Get Postfix `postqueue` command output.

This method used the postfix command defined in `postqueue_cmd` attribute to view the mails queue content.

Command defined in `postqueue_cmd` attribute is runned using a `subprocess.Popen` instance.

Returns Command's output lines.

Return type `list()`

See also:

Python module: `subprocess` – Subprocess management

`PostqueueStore._is_mail_id(mail_id)`

Check `mail_id` for a valid postfix queued mail ID.

Validation is made using a `re.RegexObject` stored in the `mail_id_re` attribute of the `PostqueueStore` instance.

Parameters **mail_id** (*str*) – Mail Postfix queue ID string

Returns True or false

Return type `bool()`

`PostqueueStore.summary()`

Summarize the mails queue content.

Returns Mail queue summary as `dict`

Sizes are in bytes.

Example response:

```
{
    'total_mails': 500,
    'total_mails_size': 709750,
    'average_mail_size': 1419.5,
    'max_mail_size': 2414,
    'min_mail_size': 423,
    'top_errors': [
        ('mail transport unavailable', 484),
        ('Test error message', 16)
    ],
    'top_recipient_domains': [
        ('test-domain.tld', 500)
    ],
    'top_recipients': [
        ('user-3@test-domain.tld', 200),
        ('user-2@test-domain.tld', 200),
        ('user-1@test-domain.tld', 100)
    ],
    'top_sender_domains': [
        ('test-domain.tld', 500)
    ],
    'top_senders': [
        ('sender-1@test-domain.tld', 100),
        ('sender-2@test-domain.tld', 100),
        ('sender-7@test-domain.tld', 50),
        ('sender-4@test-domain.tld', 50),
        ('sender-5@test-domain.tld', 50)
    ],
    'top_status': [
        ('deferred', 500),
        ('active', 0),
        ('hold', 0)
    ],
    'unique_recipient_domains': 1,
    'unique_recipients': 3,
    'unique_sender_domains': 1,
    'unique_senders': 8
}
```

1.2 Mail Objects

class `pymailq.store.Mail` (`mail_id`[, `size`[, `date`[, `sender`]]])

Simple object to manipulate email messages.

This class provides the necessary methods to load and inspect mails content. This object functionalities are mainly based on `email` module's provided class and methods. However, `email.message.Message` instance's stored informations are extracted to extend Mail instances attributes.

Initialization of Mail instances are made the following way:

Parameters

- **mail_id** (*str*) – Mail’s queue ID string
- **size** (*int*) – Mail size in Bytes (Default: 0)
- **date** (*datetime.datetime*) – Acceptance date and time in mails queue. (Default: *None*)
- **sender** (*str*) – Mail sender string as seen in mails queue. (Default: *empty str()*)

The Mail class defines the following attributes:

qid

Mail Postfix queue ID string, validated by `_is_mail_id()` method.

size

Mail size in bytes. Expected type is `int()`.

parsed

`bool()` value to track if mail’s content has been loaded from corresponding spool file.

parse_error

Last encountered parse error message `str()`.

date

`datetime` object of acceptance date and time in mails queue.

status

Mail’s queue status `str()`.

sender

Mail’s sender `str()` as seen in mails queue.

recipients

Recipients `list()` as seen in mails queue.

errors

Mail deliver errors `list()` as seen in mails queue.

head

Mail’s headers *MailHeaders* structure.

postcat_cmd

This property use Postfix mails content parsing command defined in *pymailq.CONFIG* attribute under the key ‘cat_message’. Command and arguments list is build on call with the configuration data.

See also:

pymailq.CONFIG – Configuration structure and usage

The Mail instance provides the following methods:

Mail.parse()

Parse message content.

This method use Postfix mails content parsing command defined in `postcat_cmd` attribute. This command is runned using `subprocess.Popen` instance.

Parsed headers become attributes and are retrieved with help of `message_from_string()` function provided by the `email` module.

See also:

Postfix manual: `postcat` – Show Postfix queue file contents

`Mail.dump()`

Dump mail's gathered informations to a `dict` object.

Mails informations are splitted in two parts in dictionary. `postqueue` key regroups every informations directly gathered from Postfix queue, while `headers` regroups `MailHeaders` attributes converted from mail content with the `parse()` method.

If mail has not been parsed with the `parse()` method, informations under the `headers` key will be empty.

Returns Mail gathered informations

Return type `dict`

`Mail.show()`

Return mail detailed representation for printing

Returns Representation as `str`

1.3 MailHeaders Objects

class `store.MailHeaders`

Simple object to store mail headers.

Object's attributes are dynamically created when parent `Mail` object's method `parse()` is called. Those attributes are retrieved with help of `message_from_string()` method provided by the `email` module.

Standard RFC 822-style mail headers becomes attributes including but not limited to:

- *Received*
- *From*
- *To*
- *Cc*
- *Bcc*
- *Sender*
- *Reply-To*
- *Subject*

Case is kept while creating attribute and access will be made with `Mail.From` or `Mail.Received` for example. All those attributes will return *list* of values.

See also:

Python modules: `email` – An email and MIME handling package

`email.message.Message` – Representing an email message

RFC 822 – Standard for ARPA Internet Text Messages

pymailq.selector – Mails queue filtering

The `selector` module mainly provide a selector class to interact with structures from the `store` module.

2.1 MailSelector Objects

class `selector.MailSelector` (*store*)

Mail selector class to request mails from store matching criterias.

The *MailSelector* instance provides the following attributes:

mails

Currently selected Mail objects list ()

store

Linked PostqueueStore at the *MailSelector* instance initialization.

filters

Applied filters list () on current selection. Filters list entries are tuples containing (function.__name__, args, kwargs) for each applied filters. This list is filled by the *filter_registration()* decorator while calling filtering methods. It is possible to replay registered filter using *replay_filters()* method.

The *MailSelector* instance provides the following methods:

`MailSelector.filter_registration` (*function*)

Decorator to register applied filter.

This decorated is used to wrap selection methods `lookup_*`. It registers a (function.__name__, args, kwargs) tuple () in the *filters* attribute.

`MailSelector.reset` ()

Reset mail selector with initial store mails list.

Selected Mail objects are deleted and the *mails* attribute is removed for memory releasing purpose (with help of `gc.collect()`). Attribute *mails* is then reinitialized a copy of *store*'s mails attribute.

Registered *filters* are also emptied.

`MailSelector.replay_filters()`

Reset selection with store content and replay registered filters.

Like with the *reset()* method, selected Mail objects are deleted and reinitialized with a copy of *store*'s mails attribute.

However, registered *filters* are kept and replayed on resetted selection. Use this method to refresh your store content while keeping your filters.

`MailSelector.get_mails_by_qids(qids)`

Get mails with specified IDs.

This function is not registered as filter.

Parameters *qids* (*list*) – List of mail IDs.

Returns List of newly selected Mail objects

Return type *list()*

`MailSelector.lookup_qids(*args, **kwargs)`

Lookup mails with specified IDs.

Parameters *qids* (*list*) – List of mail IDs.

Returns List of newly selected Mail objects

Return type *list()*

`MailSelector.lookup_status(*args, **kwargs)`

Lookup mails with specified postqueue status.

Parameters *status* (*list*) – List of matching status to filter on.

Returns List of newly selected Mail objects

Return type *list()*

`MailSelector.lookup_sender(*args, **kwargs)`

Lookup mails send from a specific sender.

Optionnal parameter *partial* allow lookup of partial sender like @domain.com or sender@. By default, *partial* is False and selection is made on exact sender.

Note: Matches are made against Mail.sender attribute instead of real mail header *Sender*.

Parameters

- **sender** (*str*) – Sender address to lookup in Mail objects selection.

- **exact** (*bool*) – Allow lookup with partial or exact match

Returns List of newly selected Mail objects

Return type *list()*

`MailSelector.lookup_recipient(*args, **kwargs)`

Lookup mails send to a specific recipient.

Optionnal parameter *partial* allow lookup of partial sender like @domain.com or sender@. By default, *partial* is False and selection is made on exact sender.

Note: Matches are made against Mail.recipients attribute instead of real mail header *To*.

Parameters

- **recipient** (*str*) – Recipient address to lookup in Mail objects selection.
- **exact** (*bool*) – Allow lookup with partial or exact match

Returns List of newly selected Mail objects

Return type `list()`

`MailSelector.lookup_error(*args, **kwargs)`

Lookup mails with specific error message (message may be partial).

Parameters **error_msg** (*str*) – Error message to filter on

Returns List of newly selected Mail objects

Return type `list()`

`MailSelector.lookup_date(*args, **kwargs)`

Lookup mails send on specific date range(s).

Parameters

- **start** (*datetime.date*) – Start date (Default: None)
- **stop** (*datetime.date*) – Stop date (Default: None)

Returns List of newly selected Mail objects

Return type `list()`

`MailSelector.lookup_size(*args, **kwargs)`

Lookup mails send with specific size.

Both arguments `smin` and `smax` are optionnal and default is set to 0. Maximum size is ignored if setted to 0. If both `smin` and `smax` are setted to 0, no filtering is done and the entire Mail objects selection is returned.

Parameters

- **smin** (*int*) – Minimum size (Default: 0)
- **smax** (*int*) – Maximum size (Default: 0)

Returns List of newly selected Mail objects

Return type `list()`

pymailq.control – Mails queue administrative operations

The `control` module define a basic python class to simplify administrative operations against the mails queue. This module is mainly based on the `postsuper` administrative tool fonctionnalités.

3.1 QueueControl Objects

class `control.QueueControl`

Postfix queue control using `postsuper` command.

The `QueueControl` instance defines the following attributes:

use_sudo

Boolean to control the use of `sudo` to invoke Postfix command. Default is `False`

postsuper_cmd

Postfix command and arguments `list()` for mails queue administrative operations.
Default is `["postsuper"]`

known_operations

Known Postfix administrative operations `dict` to associate operations to command arguments. Known associations are:

```
delete: -d
hold: -h
release: -H
requeue: -r
```

Warning: Default known associations are provided for the default mails queue administrative command `postsuper`.

See also:

Postfix manual: [postsuper](#) – Postfix superintendent

The `QueueControl` instance provides the following methods:

static `QueueControl.get_operation_cmd(operation)`

Get operation related command from configuration

This method use Postfix administrative commands defined in `pymailq.CONFIG` attribute under the key 'list_queue'. Command and arguments list is build on call with the configuration data.

Command keys are built with the `operation` argument suffixed with `_message`. Example: `hold_message` for the hold command.

Parameters `operation (str)` – Operation name

Returns Command and arguments as list

Raises `KeyError` – Operation is unknown

See also:

[pymailq.CONFIG](#) – Configuration structure and usage

`QueueControl._operate(*args, **kwargs)`

Generic method to lead operations messages from postfix mail queue.

Operations can be one of Postfix known operations stored in PyMailq module configuration.

Parameters

- **operation (str)** – Known operation from `pymailq.CONFIG`.
- **messages (list)** – List of Mail objects targetted for operation.

Returns Command's `stderr` output lines

Return type `list()`

`QueueControl.delete_messages(messages)`

Delete several messages from postfix mail queue.

This method is a `partial()` wrapper on `_operate()`. Passed operation is delete

`QueueControl.hold_messages(messages)`

Hold several messages from postfix mail queue.

This method is a `partial()` wrapper on `_operate()`. Passed operation is hold

`QueueControl.release_messages(messages)`

Release several messages from postfix mail queue.

This method is a `partial()` wrapper on `_operate()`. Passed operation is release

`QueueControl.requeue_messages(messages)`

Requeue several messages from postfix mail queue.

This method is a `partial()` wrapper on `_operate()`. Passed operation is requeue

CHAPTER 4

pymailq.shell – Mails queue management shell

The `pymailq.shell` module provide a shell to view and control Postfix mails queue. More documentation will soon be available.

pymailq.CONFIG – Configuration structure and usage

PyMailq module takes an optional *.ini* configuration file.

5.1 Section: core

- **postfix_spool** Path to postfix spool (defaults to */var/spool/postfix*)

5.2 Section: commands

- **use_sudo (yes|no)** Control the use of sudo to invoke commands (default: *yes*)
- **list_queue** Command to list messages queue (default: *mailq*)
- **cat_message** Command to cat message (default: *postcat -qv*)
- **hold_message** Command to hold message (default: *postsuper -h*)
- **release_message** Command to release message (default: *postsuper -H*)
- **requeue_message** Command to requeue message (default: *postsuper -r*)
- **delete_message** Command to delete message (default: *postsuper -d*)

5.3 Example

pymailq.ini:

```
[core]
postfix_spool = /var/spool/postfix

[commands]
```

```
use_sudo = yes
list_queue = mailq
cat_message = postcat -qv
hold_message = postsuper -h
release_message = postsuper -H
requeue_message = postsuper -r
delete_message = postsuper -d
```

pqshell – A shell-like to interact with a Postfix mails queue

6.1 DESCRIPTION

pqshell is a shell-like to interact with Postfix mails queue. It provide simple means to view the queue content, filter mails on criterias like *Sender* or *delivery errors* and lead administrative operations.

6.2 SYNOPSIS

```
pqshell [OPTION]...
```

6.3 FEATURES

- Asynchronous interactions with Postfix mails queue.
- Mails filtering on various criterias.
- Administrative operations on mails queue
- History and autocomplete via readline, if installed.

6.4 OPTIONS

-h, --help	show help message and exit
--version	show shell version and exit
--debug	activate shell debug and timing info
--config CFG_FILE	specify a configuration file for PyMailq

--summary	show mails queue summary and exit
--no-auto-load	deactivate store auto load at shell startup

6.5 SHELL COMMANDS

An inside help is available with the help command. Each provided command takes subcommands and command's help can be obtained while running it without argument.

6.5.1 store

Control of Postfix queue content storage

Subcommands:

status Show store status.

load Load Postfix queue content.

Example:

```
PyMailq (sel:0)> store status
store is not loaded
PyMailq (sel:0)> store load
590 mails loaded from queue
PyMailq (sel:590)> store status
store loaded with 590 mails at 2014-05-05 13:43:22.592767
```

6.5.2 select

Select mails from Postfix queue content. Filters are cumulatives and designed to simply implement advanced filtering with simple syntax. The default prompt will show how many mails are currently selected by all applied filters. Order of filters application is also important.

Subcommands:

qids Select mails by queue IDs.

Usage: select qids <qid> [qid] ...

date Select mails by date.

Usage: select date <DATESPEC>

Where <DATESPEC> can be:

```
YYYY-MM-DD (exact date)
YYYY-MM-DD..YYYY-MM-DD (within a date range (included))
+YYYY-MM-DD (after a date (included))
-YYYY-MM-DD (before a date (included))
```

error Select mails by error message. Specified error message can be partial and will be check against the whole error message.

Usage: select error <error_msg>

replay Reset content of selector with store content and replay filters.

reset Reset content of selector with store content, remove filters.

rmfilter Remove filter previously applied. Filters ids are used to specify filter to remove.

Usage: select rmfilter <filterid>

sender Select mails from sender.

Usage: select sender <sender> [exact]

recipient Select mails to recipient.

Usage: select recipient <recipient> [exact]

size Select mails by size in Bytes. Signs - and + are supported, if not specified, search for exact size. Size range is allowed by using - (lesser than) and + (greater than).

Usage: select size <-n|n|+n> [-n]

status Select mails with specific postfix status.

Usage: select status <status>

Filtering Example:

```
PyMailq (sel:608)> select size -5000
PyMailq (sel:437)> select sender MAILER-DAEMON
PyMailq (sel:316)> select status active
PyMailq (sel:0)>
```

Filters management:

```
PyMailq (sel:608)> select size -5000
PyMailq (sel:437)> select sender MAILER-DAEMON
PyMailq (sel:316)> show filters
0: select size:
    smax: 5000
    smin: 0
1: select sender:
    partial: True
    sender: MAILER-DAEMON
PyMailq (sel:316)> select rmfilter 1
PyMailq (sel:437)> select sender greedy-sender@domain.com
PyMailq (sel:25)> select reset
Selector resetted with store content (608 mails)
PyMailq (sel:608)>
```

6.5.3 inspect

Display mails content.

Subcommands:

mails: Show mails most common fields content including by not limited to *From*, *To*, *Subject*, *Received*, ... This command parses mails content and requires specific privileges or the use of *sudo* in configuration.

Usage: inspect mails <qid> [qid] ...

6.5.4 show

Display the content of current mails selection or specific mail IDs. Modifiers have been implemented to allow quick output manipulation. These allow you to sort, limit or even output a ranking by specific field. By default, output is sorted by **date of acceptance** in queue.

Optionnal modifiers can be provided to alter output:

limit <n> Display the first n entries.

sortby <field> [**asc**|**desc**] Sort output by field asc or desc. Default sorting is made descending.

rankby <field> Produce mails ranking by field.

Known fields:

- **qid** – Postqueue mail ID.
- **date** – Mail date.
- **sender** – Mail sender.
- **recipients** – Mail recipients (list, no sort).
- **size** – Mail size.
- **errors** – Postqueue deferred error messages (list, no sort).

Output formatting:

- **brief** – Default single line output to display selection
- **long** – Long format to also display errors and recipients

Subcommands:

filters Show filters applied on current mails selection.

Usage: show filters

selected Show selected mails.

Usage: show selected [modifiers]

Example:

```
PyMailq (sel:608)> show selected limit 5
2014-05-05 20:54:24 699C11831669 [active] jjj@dom1.com (14375B)
2014-05-05 20:43:39 8D60C13C14C6 [deferred] bbb@dom9.com (39549B)
2014-05-05 20:35:08 B0077198BC31 [deferred] rrr@dom2.com (4809B)
2014-05-05 20:30:09 014E21AB4B78 [deferred] aaa@dom7.com (2450B)
2014-05-05 20:25:04 CF1BE127A8D3 [deferred] xxx@dom2.com (4778B)
...Preview of first 5 (603 more)...
PyMailq (sel:608)> show selected sortby sender limit 5 asc
2014-05-02 11:36:16 40AA9149A9D7 [deferred] aaa@dom1.com (8262B)
2014-05-01 05:30:23 5E0B2162BE63 [deferred] bbb@dom4.com (3052B)
2014-05-02 05:30:20 653471AC5F76 [deferred] ccc@dom5.com (3052B)
2014-05-02 09:49:01 A00D3159AEE [deferred] ddd@dom1.com (3837B)
2014-05-05 18:18:59 98E9A790749 [deferred] ddd@dom2.com (1551B)
...Preview of first 5 (603 more)...
PyMailq (sel:608)> show selected rankby sender limit 5
sender                                     count
=====
jjj@dom8.com                               334
```

xxx@dom4.com	43
nnn@dom1.com	32
ccc@dom3.com	14
sss@dom5.com	13
...Preview of first 5 (64 more)...	

p

`pymailq, ??`

`pymailq.shell, 15`

Symbols

`_get_postqueue_output()` (store.PostqueueStore method), 5
`_is_mail_id()` (store.PostqueueStore method), 5
`_load_from_postqueue()` (store.PostqueueStore method), 4
`_load_from_spool()` (store.PostqueueStore method), 5
`_operate()` (control.QueueControl method), 14

C

CONFIG (in module pymailq), 1

D

date (pymailq.store.Mail attribute), 7
 DEBUG (in module pymailq), 1
 debug() (in module pymailq), 1
 delete_messages() (control.QueueControl method), 14
 dump() (store.Mail method), 8

E

errors (pymailq.store.Mail attribute), 7

F

filter_registration() (selector.MailSelector method), 9
 filters (selector.MailSelector attribute), 9

G

get_mails_by_qids() (selector.MailSelector method), 10
 get_operation_cmd() (control.QueueControl static method), 14

H

head (pymailq.store.Mail attribute), 7
 hold_messages() (control.QueueControl method), 14

K

known_operations (control.QueueControl attribute), 13

L

load() (store.PostqueueStore method), 4
 load_config() (in module pymailq), 1
 loaded_at (pymailq.store.PostqueueStore attribute), 3
 lookup_date() (selector.MailSelector method), 11
 lookup_error() (selector.MailSelector method), 11
 lookup_qids() (selector.MailSelector method), 10
 lookup_recipient() (selector.MailSelector method), 10
 lookup_sender() (selector.MailSelector method), 10
 lookup_size() (selector.MailSelector method), 11
 lookup_status() (selector.MailSelector method), 10

M

Mail (class in pymailq.store), 6
 mail_addr_re (pymailq.store.PostqueueStore attribute), 4
 mail_id_re (pymailq.store.PostqueueStore attribute), 3
 MailClass (pymailq.store.PostqueueStore attribute), 4
 MailHeaders (class in store), 8
 mails (pymailq.store.PostqueueStore attribute), 3
 mails (selector.MailSelector attribute), 9
 MailSelector (class in selector), 9

P

parse() (store.Mail method), 7
 parse_error (pymailq.store.Mail attribute), 7
 parsed (pymailq.store.Mail attribute), 7
 postcat_cmd (pymailq.store.Mail attribute), 7
 postqueue_cmd (pymailq.store.PostqueueStore attribute), 3
 postqueue_mailstatus (pymailq.store.PostqueueStore attribute), 3
 PostqueueStore (class in pymailq.store), 3
 postsuper_cmd (control.QueueControl attribute), 13
 pymailq (module), 1
 pymailq.shell (module), 15

Q

qid (pymailq.store.Mail attribute), 7
 QueueControl (class in control), 13

R

recipients (pymailq.store.Mail attribute), 7
release_messages() (control.QueueControl method), 14
replay_filters() (selector.MailSelector method), 10
requeue_messages() (control.QueueControl method), 14
reset() (selector.MailSelector method), 9
RFC
 RFC 3696, 4
 RFC 822, 8

S

sender (pymailq.store.Mail attribute), 7
show() (store.Mail method), 8
size (pymailq.store.Mail attribute), 7
spool_path (pymailq.store.PostqueueStore attribute), 3
status (pymailq.store.Mail attribute), 7
store (selector.MailSelector attribute), 9
summary() (store.PostqueueStore method), 5

U

use_sudo (control.QueueControl attribute), 13

V

VERSION (in module pymailq), 1