
aiospamc Documentation

Release 1.0.0

Michael Caley

Oct 11, 2023

CONTENTS

1	Contents	3
1.1	Installation	3
1.2	User Guide	3
1.3	API Reference	10
1.4	SPAMC/SPAMD Protocol As Implemented by SpamAssassin	45
1.5	Release Notes	57
2	Indices and tables	59
	Python Module Index	61
	Index	63

aiospamc is an asyncio-based library to interact with SpamAssassin's SPAMD service.

CONTENTS

1.1 Installation

1.1.1 Requirements

- Python 3.8 or later
- SpamAssassin running as a service

1.1.2 Install

With pip

```
pip install aiospamc
```

With git

```
git clone https://github.com/mjcaley/aiospamc.git
cd aiospamc
poetry install
```

Note: aiospamc's build system uses Poetry which you can get from here: <https://python-poetry.org/>

1.2 User Guide

1.2.1 Command Line Interface

Description

aiospamc is the command line interface for the SpamAssassin client. It provides common actions to interact with the SpamAssassin server.

Global Options

--version

Print the version of **aiospamc** to the console.

--debug

Enable debug logging.

Commands

check [MESSAGE]

Sends message to SpamAssassin and prints the score if there is any.

If no message is given then it will read from *stdin*.

The exit code will be 0 if the message is ham and 1 if it's spam.

-h, --host HOSTNAME

Hostname or IP address of the server.

-p, --port PORT

Port number of the server.

--socket-path PATH

Path to UNIX domain socket.

--ssl

Enables or disables SSL when using a TCP connection. Will use the system's root certificates by default.

--ca-cert PATH

Path to certificate authority file or path. Overrides the default path.

--client-cert PATH

Filename for the client certificate.

--client-key PATH

Filename for the client certificate key. Specify this if this isn't included in the client certificate.

--key-password

Password for the client certificate key if encrypted.

--user USERNAME

User to send the request as.

--timeout SECONDS

Set the connection timeout. Default is 10 seconds.

--out [json|text]

Choose the output format to the console. *text* will print human friendly output. *json* will display JSON formatted output with keys for *request*, *response*, and *exit_code*. Default is *text*.

forget [MESSAGE]

Forgets the classification of a message.

-h, --host HOSTNAME

Hostname or IP address of the server.

-p, --port PORT
Port number of the server.

--socket-path PATH
Path to UNIX domain socket.

--ssl
Enables or disables SSL when using a TCP connection. Will use the system's root certificates by default.

--ca-cert PATH
Path to certificate authority file or path. Overrides the default path.

--client-cert PATH
Filename for the client certificate.

--client-key PATH
Filename for the client certificate key. Specify this if this isn't included in the client certificate.

--key-password
Password for the client certificate key if encrypted.

--user USERNAME
User to send the request as.

--timeout SECONDS
Set the connection timeout. Default is 10 seconds.

--out [json|text]
Choose the output format to the console. *text* will print human friendly output. *json* will display JSON formatted output with keys for *request*, *response*, and *exit_code*. Default is *text*.

learn [MESSAGE]

Ask SpamAssassin to learn the message as spam or ham.

-h, --host HOSTNAME
Hostname or IP address of the server.

-p, --port PORT
Port number of the server.

--socket-path PATH
Path to UNIX domain socket.

--ssl
Enables or disables SSL when using a TCP connection. Will use the system's root certificates by default.

--ca-cert PATH
Path to certificate authority file or path. Overrides the default path.

--client-cert PATH
Filename for the client certificate.

--client-key PATH
Filename for the client certificate key. Specify this if this isn't included in the client certificate.

--key-password
Password for the client certificate key if encrypted.

--user USERNAME

User to send the request as.

--timeout SECONDS

Set the connection timeout. Default is 10 seconds.

--out [json|text]

Choose the output format to the console. *text* will print human friendly output. *json* will display JSON formatted output with keys for *request*, *response*, and *exit_code*. Default is *text*.

ping

Pings SpamAssassin and prints the response.

An exit code of 0 is successful, 1 is not successful.

-h, --host HOSTNAME

Hostname or IP address of the server.

-p, --port PORT

Port number of the server.

--socket-path PATH

Path to UNIX domain socket.

--ssl

Enables or disables SSL when using a TCP connection. Will use the system's root certificates by default.

--ca-cert PATH

Path to certificate authority file or path. Overrides the default path.

--client-cert PATH

Filename for the client certificate.

--client-key PATH

Filename for the client certificate key. Specify this if this isn't included in the client certificate.

--key-password

Password for the client certificate key if encrypted.

--user USERNAME

User to send the request as.

--timeout SECONDS

Set the connection timeout. Default is 10 seconds.

--out [json|text]

Choose the output format to the console. *text* will print human friendly output. *json* will display JSON formatted output with keys for *request*, *response*, and *exit_code*. Default is *text*.

report [MESSAGE]

Report a message to collaborative filtering databases as spam.

If reporting fails will exit with a code of 1.

-h, --host HOSTNAME

Hostname or IP address of the server.

-p, --port PORT

Port number of the server.

--socket-path PATH

Path to UNIX domain socket.

--ssl

Enables or disables SSL when using a TCP connection. Will use the system's root certificates by default.

--ca-cert PATH

Path to certificate authority file or path. Overrides the default path.

--client-cert PATH

Filename for the client certificate.

--client-key PATH

Filename for the client certificate key. Specify this if this isn't included in the client certificate.

--key-password

Password for the client certificate key if encrypted.

--user USERNAME

User to send the request as.

--timeout SECONDS

Set the connection timeout. Default is 10 seconds.

--out [json|text]

Choose the output format to the console. *text* will print human friendly output. *json* will display JSON formatted output with keys for *request*, *response*, and *exit_code*. Default is *text*.

revoke [MESSAGE]

Revoke a message to collaborative filtering databases.

If revoking fails will exit with a code of 1.

-h, --host HOSTNAME

Hostname or IP address of the server.

-p, --port PORT

Port number of the server.

--socket-path PATH

Path to UNIX domain socket.

--ssl

Enables or disables SSL when using a TCP connection. Will use the system's root certificates by default.

--ca-cert PATH

Path to certificate authority file or path. Overrides the default path.

--client-cert PATH

Filename for the client certificate.

--client-key PATH

Filename for the client certificate key. Specify this if this isn't included in the client certificate.

--key-password

Password for the client certificate key if encrypted.

--user USERNAME

User to send the request as.

--timeout SECONDS

Set the connection timeout. Default is 10 seconds.

--out [json|text]

Choose the output format to the console. *text* will print human friendly output. *json* will display JSON formatted output with keys for *request*, *response*, and *exit_code*. Default is *text*.

Environment Variables

AIOSPAMC_CERT_FILE

Path to the file containing trusted certificates. These will be used in place of the default root certificates when using the `--ssl` option.

Exit Codes

3 - Error occurred when parsing response. 4 - Network timeout. 5 - Connection error. Check the host, port, or socket path. 6 - Unexpected error. 7 - Could not open the message.

1.2.2 Library

aiospamc provides top-level functions for all request types.

For example, to ask SpamAssassin to check and score a message you can use the `aiospamc.check()` function. Just give it a bytes-encoded copy of the message, specify the host and await on the request. In this case, the response will contain a header called *Spam* with a boolean if the message is considered spam as well as the score.

```
import asyncio
import aiospamc

example_message = (
    "From: John Doe <jdoe@machine.example>"
    "To: Mary Smith <mary@example.net>"
    "Subject: Saying Hello"
    "Date: Fri, 21 Nov 1997 09:55:06 -0600"
    "Message-ID: <1234@local.machine.example>"
    ""
    "This is a message just to say hello."
    "So, 'Hello'.") .encode("ascii")

response = asyncio.run(aiospamc.check(message, host="localhost"))
print(
    f"Is the message spam? {response.headers.spam.value}\n",
    f"The score and threshold is {response.headers.spam.score} ",
    f"/ {response.headers.spam.threshold}",
    sep=""
)
```

Connect using SSL

Each frontend function has a *verify* parameter which allows configuring an SSL connection.

If *True* is supplied, then root certificates from the *certifi* project will be used to verify the connection.

If a path is supplied as a string or `pathlib.Path` object then the path is used to load certificates to verify the connection.

If *False* then an SSL connection is established, but the server certificate is not verified.

Client Certificate Authentication

Client certificate authentication can be used with SSL. It's driven through the *cert* parameter on frontend functions. The parameter value takes three forms: * A path to a file expecting the certificate and key in the PEM format * A tuple of certificate and key files * A tuple of certificate file, key file, and password if the key is encrypted

```
import aiospamc

# Client certificate and key in one file
response = await aiospamc.ping("localhost", cert=cert_file)

# Client certificate and key file
response = await aiospamc.ping("localhost", cert=(cert_file, key_file))

# Client certificate and key in one file
response = await aiospamc.ping("localhost", cert=(cert_file, key_file, password))
```

Setting timeouts

aiospamc is configured by default to use a timeout of 600 seconds (or 10 minutes) from the point when a connection is attempted until a response comes in.

If you would like more fine-grained control of timeouts then an *aiospamc.connections.Timeout* object can be passed in.

You can configure any of the three optional parameters: * total - maximum time in seconds to wait for a connection and response * connection - time in seconds to wait for a connection to be established * response - time in seconds to wait for a response after sending the request

```
import aiospamc

my_timeout = aiospamc.Timeout(total=60, connection=10, response=10)

await def check():
    response = await aiospamc.check(example_message, timeout=my_timeout)

    return response
```

Logging

Logging is provided using through the `loguru` package.

The *aiospamc* package disables logging by default. It can be enabled by calling the function:

```
from loguru import logger
logger.enable("aiospamc")
```

Modules log under their own logger names (for example, frontend functions will log under *aiospamc.frontend*). Extra data like request and response objects are attached to log records which can be used to trace through flow.

Interpreting results

Responses are encapsulated in the *aiospamc.responses.Response* class. It includes the status code, headers and body.

1.3 API Reference

1.3.1 aiospamc package

Submodules

aiospamc.cli module

CLI commands.

```
class aiospamc.cli.Output(value, names=None, *, module=None, qualname=None, type=None, start=1,
                           boundary=None)
```

Bases: `str`, `Enum`

Output formats.

Json = 'json'

Text = 'text'

```
class aiospamc.cli.CliClientBuilder
```

Bases: `object`

Client builder for CLI arguments.

```
__init__()
```

Constructor for the `CliClientBuilder`.

```
build() → Client
```

Builds the *aiospamc.client.Client*.

Returns

An instance of *aiospamc.client.Client*.

```
with_connection(host: str = 'localhost', port: int = 783, socket_path: Path | None = None) →  
                  CliClientBuilder
```

Sets the type of connection manager to use.

Defaults to TCP, but if a Unix socket is provided it will be used.

Parameters

- **host** – TCP hostname to use.
- **port** – TCP port to use.
- **socket_path** – Path to the Unix socket.

Returns

This builder instance.

set_timeout(*timeout*: [Timeout](#)) → [CliClientBuilder](#)

Sets the timeout for the connection.

Parameters

timeout – Timeout object.

Returns

This builder instance.

add_verify(*verify*: [bool](#)) → [CliClientBuilder](#)

Adds an SSL context to the connection manager.

Parameters

verify – How to configure the SSL context. If *True*, add the default certificate authorities. If *False*, accept any certificate.

Returns

This builder instance.

add_ca_cert(*ca_cert*: [Path](#) | *None*) → [CliClientBuilder](#)

Adds trusted certificate authorities.

Parameters

ca_cert – Path to the certificate file or directory.

Returns

This builder instance.

add_client_cert(*cert*: [Path](#) | *None*, *key*: [Path](#) | *None* = *None*, *password*: [str](#) | *None* = *None*) → [CliClientBuilder](#)

Add a client certificate to authenticate to the server.

Parameters

- **cert** – Path to the client certificate and optionally the key.
- **key** – Path to the client key.
- **password** – Password of the client key.

Returns

This builder instance.

class `aiosпамс.cli.CommandRunner`(*client*: [Client](#), *request*: [Request](#), *output*: [Output](#) = *Output.Text*)

Bases: [object](#)

Object to execute requests and handle exceptions.

__init__(*client*: [Client](#), *request*: [Request](#), *output*: [Output](#) = *Output.Text*)

CommandRunner constructor.

Parameters

- **request** – Request to send.
- **response** – Response if returned from server.
- **output** – Output format when printing to console.

async run() → *Response*

Send the request, get the response and handle common exceptions.

Returns

The response.

to_json() → *str*

Converts the object to a JSON string.

Returns

JSON string.

exit(message: str, err=False)

Exits the program, echoing the message if outputting text. Otherwise prints the JSON object.

Parameters

- **message** – Message text to print.
- **err** – Flag if message is an error.

aiospamc.cli.ping(host: *str* = 'localhost', port: *int* = 783, socket_path: *Path* | *None* = *None*, ssl: *bool* = *False*, timeout: *float* = 10, out: *Output* = *Output.Text*, ca_cert: *Path* | *None* = *None*, client_cert: *Path* | *None* = *None*, client_key: *Path* | *None* = *None*, key_password: *str* | *None* = *None*)

Pings the SpamAssassin daemon.

A successful pong exits with code 0.

aiospamc.cli.read_message(file: *BufferedReader* | *None*) → *bytes*

Utility function to read data from stdin.

Parameters

file – File-like object.

aiospamc.cli.check(message: *FileBinaryRead* | *None* = *None*, host: *str* = 'localhost', port: *int* = 783, socket_path: *Path* | *None* = *None*, ssl: *bool* = *False*, user: *str* = 'docs', timeout: *float* = 10, out: *Output* = *Output.Text*, ca_cert: *Path* | *None* = *None*, client_cert: *Path* | *None* = *None*, client_key: *Path* | *None* = *None*, key_password: *str* | *None* = *None*)

Submits a message to SpamAssassin and returns the processed message.

aiospamc.cli.learn(message: *FileBinaryRead* | *None* = *None*, message_class: *MessageClassOption* = *MessageClassOption.spam*, host: *str* = 'localhost', port: *int* = 783, socket_path: *Path* | *None* = *None*, ssl: *bool* = *False*, user: *str* = 'docs', timeout: *float* = 10, out: *Output* = *Output.Text*, ca_cert: *Path* | *None* = *None*, client_cert: *Path* | *None* = *None*, client_key: *Path* | *None* = *None*, key_password: *str* | *None* = *None*)

Ask server to learn the message as spam or ham.

aiospamc.cli.forget(message: *FileBinaryRead* | *None* = *None*, host: *str* = 'localhost', port: *int* = 783, socket_path: *Path* | *None* = *None*, ssl: *bool* = *False*, user: *str* = 'docs', timeout: *float* = 10, out: *Output* = *Output.Text*, ca_cert: *Path* | *None* = *None*, client_cert: *Path* | *None* = *None*, client_key: *Path* | *None* = *None*, key_password: *str* | *None* = *None*)

Forgets the classification of a message.

```
aiospamc.cli.report(message: FileBinaryRead | None = None, host: str = 'localhost', port: int = 783,
                    socket_path: Path | None = None, ssl: bool = False, user: str = 'docs', timeout: float = 10,
                    out: Output = Output.Text, ca_cert: Path | None = None, client_cert: Path | None = None,
                    client_key: Path | None = None, key_password: str | None = None)
```

Report a message to collaborative filtering databases as spam.

```
aiospamc.cli.revoke(message: FileBinaryRead | None = None, host: str = 'localhost', port: int = 783,
                    socket_path: Path | None = None, ssl: bool = False, user: str = 'docs', timeout: float = 10,
                    out: Output = Output.Text, ca_cert: Path | None = None, client_cert: Path | None = None,
                    client_key: Path | None = None, key_password: str | None = None)
```

Revoke a message to collaborative filtering databases.

```
aiospamc.cli.version_callback(version: bool)
```

Callback to print the version.

Parameters

version – Switch on whether to print and exit.

```
aiospamc.cli.debug_callback(debug: bool)
```

Callback to enable debug logging.

Parameters

debug – Switch on whether to enable debug logging.

```
aiospamc.cli.main(version: bool = False, debug: bool = False)
```

aiospamc sends messages to the SpamAssassin daemon.

aiospamc.client module

Module implementing client objects that all requests go through.

```
class aiospamc.client.Client(connection_manager: ConnectionManager)
```

Bases: `object`

Client object to submit requests.

```
__init__(connection_manager: ConnectionManager)
```

Client constructor.

Parameters

connection_manager – Instance of a connection manager.

```
async request(req: Request)
```

Sends a request and returns the parsed response.

Parameters

req – The request to send.

Returns

The parsed response.

Raises

- **BadResponse** – If the response from SPAMD is ill-formed this exception will be raised.
- **AIOSpamcConnectionFailed** – Raised if an error occurred when trying to connect.
- **UsageException** – Error in command line usage.
- **DataErrorException** – Error with data format.

- **NoInputException** – Cannot open input.
- **NoUserException** – Addressee unknown.
- **NoHostException** – Hostname unknown.
- **UnavailableException** – Service unavailable.
- **InternalSoftwareException** – Internal software error.
- **OSErrorException** – System error.
- **OSFileException** – Operating system file missing.
- **CantCreateException** – Cannot create output file.
- **IOErrorException** – Input/output error.
- **TemporaryFailureException** – Temporary failure, may retry.
- **ProtocolException** – Error in the protocol.
- **NoPermissionException** – Permission denied.
- **ConfigException** – Error in configuration.
- **ServerTimeoutException** – Server returned a response that it timed out.
- **ClientTimeoutException** – Client timed out during connection.

aiospamc.connections module

ConnectionManager classes for TCP and Unix sockets.

```
class aiospamc.connections.Timeout(total: float = 600, connection: float | None = None, response: float | None = None)
```

Bases: `object`

Container object for defining timeouts.

```
__init__(total: float = 600, connection: float | None = None, response: float | None = None) → None
```

Timeout constructor.

Parameters

- **total** – The total length of time in seconds to set the timeout.
- **connection** – The length of time in seconds to allow for a connection to live before timing out.
- **response** – The length of time in seconds to allow for a response from the server before timing out.

```
class aiospamc.connections.ConnectionManagerBuilder
```

Bases: `object`

Builder for connection managers.

```
class ManagerType(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)
```

Bases: `Enum`

Define connection manager type during build.

Undefined = 1

Tcp = 2

Unix = 3

__init__()

ConnectionManagerBuilder constructor.

build() → *UnixConnectionManager* | *TcpConnectionManager*

Builds the *aiospamc.connections.ConnectionManager*.

Returns

An instance of *aiospamc.connections.TcpConnectionManager* or *aiospamc.connections.UnixConnectionManager*

with_unix_socket(*path*: *Path*) → *ConnectionManagerBuilder*

Configures the builder to use a Unix socket connection.

Parameters

path – Path to the Unix socket.

Returns

This builder instance.

with_tcp(*host*: *str*, *port*: *int* = 783) → *ConnectionManagerBuilder*

Configures the builder to use a TCP connection.

Parameters

- **host** – Hostname to use.
- **port** – Port to use.

Returns

This builder instance.

add_ssl_context(*context*: *SSLContext*) → *ConnectionManagerBuilder*

Adds an SSL context when a TCP connection is being used.

Parameters

context – *ssl.SSLContext* instance.

Returns

This builder instance.

set_timeout(*timeout*: *Timeout*) → *ConnectionManagerBuilder*

Sets the timeout for the connection.

Parameters

timeout – Timeout object.

Returns

This builder instance.

class *aiospamc.connections.ConnectionManager*(*connection_string*: *str*, *timeout*: *Timeout* | *None* = *None*)

Bases: *object*

Stores connection parameters and creates connections.

__init__(*connection_string*: *str*, *timeout*: *Timeout* | *None* = *None*) → *None*

ConnectionManager constructor.

Parameters

timeout – Timeout configuration

property logger: *loguru.Logger*

Return the logger object.

async request(*data*: *bytes*) → *bytes*

Send bytes data and receive a response.

Raises

AIOspamcConnectionFailed

Raises

ClientTimeoutException

Parameters

data – Data to send.

async open() → *Tuple*[*StreamReader*, *StreamWriter*]

Opens a connection, returning the reader and writer objects.

property connection_string: *str*

String representation of the connection.

class *aiospamc.connections.TcpConnectionManagerBuilder*

Bases: *object*

Builder for *aiospamc.connections.TcpConnectionManager*

__init__()

TcpConnectionManagerBuilder constructor.

build() → *TcpConnectionManager*

Builds the *aiospamc.connections.TcpConnectionManager*.

Returns

An instance of *aiospamc.connections.TcpConnectionManager*.

set_host(*host*: *str*) → *TcpConnectionManagerBuilder*

Sets the host to use.

Parameters

host – Hostname to use.

Returns

This builder instance.

set_port(*port*: *int*) → *TcpConnectionManagerBuilder*

Sets the port to use.

Parameters

port – Port to use.

Returns

This builder instance.

set_ssl_context(context: *SSLContext*) → *TcpConnectionManagerBuilder*

Set an SSL context.

Parameters

context – An instance of `ssl.SSLContext`.

Returns

This builder instance.

set_timeout(timeout: *Timeout*) → *TcpConnectionManagerBuilder*

Sets the timeout for the connection.

Parameters

timeout – Timeout object.

Returns

This builder instance.

class aiospamc.connections.**TcpConnectionManager**(host: *str*, port: *int*, ssl_context: *SSLContext* | *None* = *None*, timeout: *Timeout* | *None* = *None*)

Bases: *ConnectionManager*

Connection manager for TCP connections.

__init__(host: *str*, port: *int*, ssl_context: *SSLContext* | *None* = *None*, timeout: *Timeout* | *None* = *None*) → *None*

TcpConnectionManager constructor.

Parameters

- **host** – Hostname or IP address.
- **port** – TCP port.
- **ssl_context** – SSL context.
- **timeout** – Timeout configuration.

async open() → *Tuple*[*StreamReader*, *StreamWriter*]

Opens a TCP connection.

Raises

AIOspamcConnectionFailed

Returns

Reader and writer for the connection.

class aiospamc.connections.**UnixConnectionManagerBuilder**

Bases: *object*

Builder for *aiospamc.connections.UnixConnectionManager*.

__init__()

UnixConnectionManagerBuilder constructor.

build() → *UnixConnectionManager*

Builds a *aiospamc.connections.UnixConnectionManager*.

Returns

An instance of *aiospamc.connections.UnixConnectionManager*.

set_path(*path*: *Path*) → *UnixConnectionManagerBuilder*

Sets the unix socket path.

Parameters

path – Path to the Unix socket.

Returns

This builder instance.

set_timeout(*timeout*: *Timeout*) → *UnixConnectionManagerBuilder*

Sets the timeout for the connection.

Parameters

timeout – Timeout object.

Returns

This builder instance.

class aiospamc.connections.**UnixConnectionManager**(*path*: *Path*, *timeout*: *Timeout* | *None* = *None*)

Bases: *ConnectionManager*

Connection manager for Unix pipes.

__init__(*path*: *Path*, *timeout*: *Timeout* | *None* = *None*)

UnixConnectionManager constructor.

Parameters

- **path** – Unix socket path.
- **timeout** – Timeout configuration

async open() → *Tuple*[*StreamReader*, *StreamWriter*]

Opens a unix socket path connection.

Raises

AIOSpamcConnectionFailed

Returns

Reader and writer for the connection.

class aiospamc.connections.**SSLContextBuilder**

Bases: *object*

SSL context builder.

__init__()

Builder constructor. Sets up a default SSL context.

build() → *SSLContext*

Builds the SSL context.

Returns

An instance of `ssl.SSLContext`.

with_context(*context*: *SSLContext*) → *SSLContextBuilder*

Use the SSL context.

Parameters

context – Provided SSL context.

Returns

The builder instance.

add_ca_file(*file*: *Path*) → *SSLContextBuilder*

Add certificate authority from a file.

Parameters

file – File of concatenated certificates.

Returns

The builder instance.

add_ca_dir(*dir*: *Path*) → *SSLContextBuilder*

Add certificate authority from a directory.

Parameters

dir – Directory of certificates.

Returns

The builder instance.

add_ca(*path*: *Path*) → *SSLContextBuilder*

Add a certificate authority.

Parameters

path – Directory or file of certificates.

Returns

The builder instance.

add_default_ca() → *SSLContextBuilder*

Add default certificate authorities.

Returns

The builder instance.

add_client(*file*: *Path*, *key*: *Path* | *None* = *None*, *password*: *Callable*[[], *str* | *bytes* | *bytearray*] | *None* = *None*) → *SSLContextBuilder*

Add client certificate.

Parameters

- **file** – Path to the client certificate.
- **key** – Path to the key.
- **password** – Callable that returns the password, if any.

dont_verify() → *SSLContextBuilder*

Set the context to not verify certificates.

aiospamc.exceptions module

Collection of exceptions.

exception aiospamc.exceptions.**ClientException**

Bases: *Exception*

Base class for exceptions raised from the client.

exception aiospamc.exceptions.**BadRequest**

Bases: *ClientException*

Request is not in the expected format.

exception aiospamc.exceptions.BadResponse

Bases: [ClientException](#)

Response is not in the expected format.

exception aiospamc.exceptions.AIOSpamcConnectionFailed

Bases: [ClientException](#)

Connection failed.

exception aiospamc.exceptions.TimeoutException

Bases: [Exception](#)

General timeout exception.

exception aiospamc.exceptions.ClientTimeoutException

Bases: [ClientException](#), [TimeoutException](#)

Timeout exception from the client.

exception aiospamc.exceptions.ParseError(*message=None*)

Bases: [Exception](#)

Error occurred while parsing.

__init__(*message=None*)

Construct parsing exception with optional message.

Parameters

message – User friendly message.

exception aiospamc.exceptions.NotEnoughDataError(*message=None*)

Bases: [ParseError](#)

Expected more data than what the protocol content specified.

exception aiospamc.exceptions.TooMuchDataError(*message=None*)

Bases: [ParseError](#)

Too much data was received than what the protocol content specified.

aiospamc.frontend module

Frontend functions for the package.

class aiospamc.frontend.FrontendClientBuilder

Bases: [object](#)

Builds the [aiospamc.client.Client](#) based off of frontend arguments.

__init__()

Constructor for FrontendClientBuilder.

build() → [Client](#)

Builds the [aiospamc.client.Client](#).

Returns

An instance of [aiospamc.client.Client](#).

with_connection(*host*: *str* = 'localhost', *port*: *int* = 783, *socket_path*: *Path* | *None* = *None*) → *FrontendClientBuilder*

Sets the type of connection manager to use.

Defaults to TCP, but if a Unix socket is provided it will be used.

Parameters

- **host** – TCP hostname to use.
- **port** – TCP port to use.
- **socket_path** – Path to the Unix socket.

Returns

This builder instance.

add_verify(*verify*: *bool* | *Path* | *SSLContext* | *None* = *None*) → *FrontendClientBuilder*

Adds an SSL context to the connection manager.

Parameters

verify – How to configure the SSL context. If *True*, add the default certificate authorities. If *False*, accept any certificate. If a `pathlib.Path`, add the certificates from it. If an `ssl.SSLContext`, then use it.

Returns

This builder instance.

add_client_cert(*cert*: *Path* | *Tuple*[*Path*, *Path* | *None*] | *Tuple*[*Path*, *Path* | *None*, *str* | *None*] | *None*) → *FrontendClientBuilder*

Add a client certificate to authenticate to the server.

Parameters

cert – Client certificate. Takes up to three a three tuple value. 1. Path to the certificate and key. 2. Path to the certificate and path to the key. 3. Path to the certificate, path to the key, and password of the key.

Returns

This builder instance.

set_timeout(*timeout*: *Timeout* | *None* = *None*) → *FrontendClientBuilder*

Sets the timeout for the connection.

Parameters

timeout – Timeout object.

Returns

This builder instance.

async aiospamc.frontend.check(*message*: *bytes* | *SupportsBytes*, *, *host*: *str* = 'localhost', *port*: *int* = 783, *socket_path*: *Path* | *None* = *None*, *timeout*: *Timeout* | *None* = *None*, *verify*: *bool* | *Path* | *SSLContext* | *None* = *None*, *cert*: *Path* | *Tuple*[*Path*, *Path* | *None*] | *Tuple*[*Path*, *Path* | *None*, *str* | *None*] | *None* = *None*, *user*: *str* | *None* = *None*, *compress*: *bool* = *False*) → *Response*

Checks a message if it's spam and return a response with a score header.

Parameters

- **message** – Copy of the message.
- **host** – Hostname or IP address of the SPAMD service, defaults to localhost.
- **port** – Port number for the SPAMD service, defaults to 783.

- **socket_path** – Path to Unix socket.
- **timeout** – Timeout settings.
- **verify** – Enable SSL. *True* will use the root certificates from the `certifi` package. *False* will use SSL, but not verify the root certificates. Passing a string to a filename will use the path to verify the root certificates.
- **cert** – Use client certificate. Can either be a `pathlib.Path` to a file that includes both the certificate and key. Can be a tuple containing a `pathlib.Path` to the certificate and a `pathlib.Path` to the key. Can be a tuple containing a `pathlib.Path` to the certificate, `pathlib.Path` to the key, and a password.
- **user** – Username to pass to the SPAMD service.
- **compress** – Enable compress of the request body.

Returns

A successful response with a “Spam” header showing if the message is considered spam as well as the score.

Raises

- **BadResponse** – If the response from SPAMD is ill-formed this exception will be raised.
- **AIOSpamcConnectionFailed** – Raised if an error occurred when trying to connect.
- **UsageException** – Error in command line usage.
- **DataErrorException** – Error with data format.
- **NoInputException** – Cannot open input.
- **NoUserException** – Addressee unknown.
- **NoHostException** – Hostname unknown.
- **UnavailableException** – Service unavailable.
- **InternalSoftwareException** – Internal software error.
- **OSErrorException** – System error.
- **OSFileException** – Operating system file missing.
- **CantCreateException** – Cannot create output file.
- **IOErrorException** – Input/output error.
- **TemporaryFailureException** – Temporary failure, may reattempt.
- **ProtocolException** – Error in the protocol.
- **NoPermissionException** – Permission denied.
- **ConfigException** – Error in configuration.
- **ServerTimeoutException** – Server returned a response that it timed out.
- **ClientTimeoutException** – Client timed out during connection.

```
async aiospamc.frontend.headers(message: bytes | SupportsBytes, *, host: str = 'localhost', port: int = 783,
                                socket_path: Path | None = None, timeout: Timeout | None = None, verify:
                                bool | Path | SSLContext | None = None, cert: Path | Tuple[Path, Path |
                                None] | Tuple[Path, Path | None, str | None] | None = None, user: str |
                                None = None, compress: bool = False) → Response
```

Checks a message if it's spam and return the modified message headers.

Parameters

- **message** – Copy of the message.
- **host** – Hostname or IP address of the SPAMD service, defaults to localhost.
- **port** – Port number for the SPAMD service, defaults to 783.
- **socket_path** – Path to Unix socket.
- **timeout** – Timeout settings.
- **verify** – Enable SSL. *True* will use the root certificates from the `certifi` package. *False* will use SSL, but not verify the root certificates. Passing a string to a filename will use the path to verify the root certificates.
- **cert** – Use client certificate. Can either be a `pathlib.Path` to a file that includes both the certificate and key. Can be a tuple containing a `pathlib.Path` to the certificate and a `pathlib.Path` to the key. Can be a tuple containing a `pathlib.Path` to the certificate, `pathlib.Path` to the key, and a password.
- **user** – Username to pass to the SPAMD service.
- **compress** – Enable compress of the request body.

Returns

A successful response with a “Spam” header showing if the message is considered spam as well as the score. The body contains the modified message headers, but not the content of the message.

Raises

- ***BadResponse*** – If the response from SPAMD is ill-formed this exception will be raised.
- ***AIOSpamcConnectionFailed*** – Raised if an error occurred when trying to connect.
- ***UsageException*** – Error in command line usage.
- ***DataErrorException*** – Error with data format.
- ***NoInputException*** – Cannot open input.
- ***NoUserException*** – Addressee unknown.
- ***NoHostException*** – Hostname unknown.
- ***UnavailableException*** – Service unavailable.
- ***InternalSoftwareException*** – Internal software error.
- ***OSErrorException*** – System error.
- ***OSFileException*** – Operating system file missing.
- ***CantCreateException*** – Cannot create output file.
- ***IOErrorException*** – Input/output error.
- ***TemporaryFailureException*** – Temporary failure, may reattempt.
- ***ProtocolException*** – Error in the protocol.
- ***NoPermissionException*** – Permission denied.
- ***ConfigException*** – Error in configuration.
- ***ServerTimeoutException*** – Server returned a response that it timed out.
- ***ClientTimeoutException*** – Client timed out during connection.

```
async aiospamc.frontend.ping(*, host: str = 'localhost', port: int = 783, socket_path: Path | None = None,
                             timeout: Timeout | None = None, verify: bool | Path | SSLContext | None =
                             None, cert: Path | Tuple[Path, Path | None] | Tuple[Path, Path | None, str |
                             None] | None = None) → Response
```

Sends a ping to the SPAMD service.

Parameters

- **host** – Hostname or IP address of the SPAMD service, defaults to localhost.
- **port** – Port number for the SPAMD service, defaults to 783.
- **socket_path** – Path to Unix socket.
- **timeout** – Timeout settings.
- **verify** – Enable SSL. *True* will use the root certificates from the *certifi* package. *False* will use SSL, but not verify the root certificates. Passing a string to a filename will use the path to verify the root certificates.
- **cert** – Use client certificate. Can either be a `pathlib.Path` to a file that includes both the certificate and key. Can be a tuple containing a `pathlib.Path` to the certificate and a `pathlib.Path` to the key. Can be a tuple containing a `pathlib.Path` to the certificate, `pathlib.Path` to the key, and a password.

Returns

A response with “PONG”.

Raises

- **BadResponse** – If the response from SPAMD is ill-formed this exception will be raised.
- **AIOSpamcConnectionFailed** – Raised if an error occurred when trying to connect.
- **UsageException** – Error in command line usage.
- **DataErrorException** – Error with data format.
- **NoInputException** – Cannot open input.
- **NoUserException** – Addressee unknown.
- **NoHostException** – Hostname unknown.
- **UnavailableException** – Service unavailable.
- **InternalSoftwareException** – Internal software error.
- **OSErrorException** – System error.
- **OSFileException** – Operating system file missing.
- **CantCreateException** – Cannot create output file.
- **IOErrorException** – Input/output error.
- **TemporaryFailureException** – Temporary failure, may reattempt.
- **ProtocolException** – Error in the protocol.
- **NoPermissionException** – Permission denied.
- **ConfigException** – Error in configuration.
- **ServerTimeoutException** – Server returned a response that it timed out.
- **ClientTimeoutException** – Client timed out during connection.

```

async aiospamc.frontend.process(message: bytes | SupportsBytes, *, host: str = 'localhost', port: int = 783,
                                socket_path: Path | None = None, timeout: Timeout | None = None, verify:
                                bool | Path | SSLContext | None = None, cert: Path | Tuple[Path, Path |
                                None] | Tuple[Path, Path | None, str | None] | None = None, user: str |
                                None = None, compress: bool = False) → Response

```

Checks a message if it's spam and return a response with a score header.

Parameters

- **message** – Copy of the message.
- **host** – Hostname or IP address of the SPAMD service, defaults to localhost.
- **port** – Port number for the SPAMD service, defaults to 783.
- **socket_path** – Path to Unix socket.
- **timeout** – Timeout settings.
- **verify** – Enable SSL. *True* will use the root certificates from the *certifi* package. *False* will use SSL, but not verify the root certificates. Passing a string to a filename will use the path to verify the root certificates.
- **cert** – Use client certificate. Can either be a *pathlib.Path* to a file that includes both the certificate and key. Can be a tuple containing a *pathlib.Path* to the certificate and a *pathlib.Path* to the key. Can be a tuple containing a *pathlib.Path* to the certificate, *pathlib.Path* to the key, and a password.
- **user** – Username to pass to the SPAMD service.
- **compress** – Enable compress of the request body.

Returns

A successful response with a “Spam” header showing if the message is considered spam as well as the score. The body contains a modified copy of the message.

Raises

- **BadResponse** – If the response from SPAMD is ill-formed this exception will be raised.
- **AIOSpamcConnectionFailed** – Raised if an error occurred when trying to connect.
- **UsageException** – Error in command line usage.
- **DataErrorException** – Error with data format.
- **NoInputException** – Cannot open input.
- **NoUserException** – Addressee unknown.
- **NoHostException** – Hostname unknown.
- **UnavailableException** – Service unavailable.
- **InternalSoftwareException** – Internal software error.
- **OSErrorException** – System error.
- **OSFileException** – Operating system file missing.
- **CantCreateException** – Cannot create output file.
- **IOErrorException** – Input/output error.
- **TemporaryFailureException** – Temporary failure, may reattempt.
- **ProtocolException** – Error in the protocol.

- **NoPermissionException** – Permission denied.
- **ConfigException** – Error in configuration.
- **ServerTimeoutException** – Server returned a response that it timed out.
- **ClientTimeoutException** – Client timed out during connection.

```
async aiospamc.frontend.report(message: bytes | SupportsBytes, *, host: str = 'localhost', port: int = 783,
                                socket_path: Path | None = None, timeout: Timeout | None = None, verify:
                                bool | Path | SSLContext | None = None, cert: Path | Tuple[Path, Path |
                                None] | Tuple[Path, Path | None, str | None] | None = None, user: str | None
                                = None, compress: bool = False) → Response
```

Checks a message if it's spam and return a response with a score header.

Parameters

- **message** – Copy of the message.
- **host** – Hostname or IP address of the SPAMD service, defaults to localhost.
- **port** – Port number for the SPAMD service, defaults to 783.
- **socket_path** – Path to Unix socket.
- **timeout** – Timeout settings.
- **verify** – Enable SSL. *True* will use the root certificates from the `certifi` package. *False* will use SSL, but not verify the root certificates. Passing a string to a filename will use the path to verify the root certificates.
- **cert** – Use client certificate. Can either be a `pathlib.Path` to a file that includes both the certificate and key. Can be a tuple containing a `pathlib.Path` to the certificate and a `pathlib.Path` to the key. Can be a tuple containing a `pathlib.Path` to the certificate, `pathlib.Path` to the key, and a password.
- **user** – Username to pass to the SPAMD service.
- **compress** – Enable compress of the request body.

Returns

A successful response with a “Spam” header showing if the message is considered spam as well as the score. The body contains a report.

Raises

- **BadResponse** – If the response from SPAMD is ill-formed this exception will be raised.
- **AIOSpamcConnectionFailed** – Raised if an error occurred when trying to connect.
- **UsageException** – Error in command line usage.
- **DataErrorException** – Error with data format.
- **NoInputException** – Cannot open input.
- **NoUserException** – Addressee unknown.
- **NoHostException** – Hostname unknown.
- **UnavailableException** – Service unavailable.
- **InternalSoftwareException** – Internal software error.
- **OSErrorException** – System error.
- **OSFileException** – Operating system file missing.

- **`CantCreateException`** – Cannot create output file.
- **`IOErrorException`** – Input/output error.
- **`TemporaryFailureException`** – Temporary failure, may reattempt.
- **`ProtocolException`** – Error in the protocol.
- **`NoPermissionException`** – Permission denied.
- **`ConfigException`** – Error in configuration.
- **`ServerTimeoutException`** – Server returned a response that it timed out.
- **`ClientTimeoutException`** – Client timed out during connection.

```
async aiospamc.frontend.report_if_spam(message: bytes | SupportsBytes, *, host: str = 'localhost', port: int
                                     = 783, socket_path: Path | None = None, timeout: Timeout | None
                                     = None, verify: bool | Path | SSLContext | None = None, cert:
                                     Path | Tuple[Path, Path | None] | Tuple[Path, Path | None, str |
                                     None] | None = None, user: str | None = None, compress: bool =
                                     False) → Response
```

Checks a message if it's spam and return a response with a score header.

Parameters

- **`message`** – Copy of the message.
- **`host`** – Hostname or IP address of the SPAMD service, defaults to localhost.
- **`port`** – Port number for the SPAMD service, defaults to 783.
- **`socket_path`** – Path to Unix socket.
- **`timeout`** – Timeout settings.
- **`verify`** – Enable SSL. *True* will use the root certificates from the `certifi` package. *False* will use SSL, but not verify the root certificates. Passing a string to a filename will use the path to verify the root certificates.
- **`cert`** – Use client certificate. Can either be a `pathlib.Path` to a file that includes both the certificate and key. Can be a tuple containing a `pathlib.Path` to the certificate and a `pathlib.Path` to the key. Can be a tuple containing a `pathlib.Path` to the certificate, `pathlib.Path` to the key, and a password.
- **`user`** – Username to pass to the SPAMD service.
- **`compress`** – Enable compress of the request body.

Returns

A successful response with a “Spam” header showing if the message is considered spam as well as the score. The body contains a report if the message is considered spam.

Raises

- **`BadResponse`** – If the response from SPAMD is ill-formed this exception will be raised.
- **`AIOSpamcConnectionFailed`** – Raised if an error occurred when trying to connect.
- **`UsageException`** – Error in command line usage.
- **`DataErrorException`** – Error with data format.
- **`NoInputException`** – Cannot open input.
- **`NoUserException`** – Addressee unknown.

- **NoHostException** – Hostname unknown.
- **UnavailableException** – Service unavailable.
- **InternalSoftwareException** – Internal software error.
- **OSErrorException** – System error.
- **OSFileException** – Operating system file missing.
- **CantCreateException** – Cannot create output file.
- **IOErrorException** – Input/output error.
- **TemporaryFailureException** – Temporary failure, may reattempt.
- **ProtocolException** – Error in the protocol.
- **NoPermissionException** – Permission denied.
- **ConfigException** – Error in configuration.
- **ServerTimeoutException** – Server returned a response that it timed out.
- **ClientTimeoutException** – Client timed out during connection.

```
async aiospamc.frontend.symbols(message: bytes | SupportsBytes, *, host: str = 'localhost', port: int = 783,
                                socket_path: Path | None = None, timeout: Timeout | None = None, verify:
                                bool | Path | SSLContext | None = None, cert: Path | Tuple[Path, Path |
                                None] | Tuple[Path, Path | None, str | None] | None = None, user: str |
                                None = None, compress: bool = False) → Response
```

Checks a message if it's spam and return a response with rules that matched.

Parameters

- **message** – Copy of the message.
- **host** – Hostname or IP address of the SPAMD service, defaults to localhost.
- **port** – Port number for the SPAMD service, defaults to 783.
- **socket_path** – Path to Unix socket.
- **timeout** – Timeout settings.
- **verify** – Enable SSL. *True* will use the root certificates from the *certifi* package. *False* will use SSL, but not verify the root certificates. Passing a string to a filename will use the path to verify the root certificates.
- **cert** – Use client certificate. Can either be a `pathlib.Path` to a file that includes both the certificate and key. Can be a tuple containing a `pathlib.Path` to the certificate and a `pathlib.Path` to the key. Can be a tuple containing a `pathlib.Path` to the certificate, `pathlib.Path` to the key, and a password.
- **user** – Username to pass to the SPAMD service.
- **compress** – Enable compress of the request body.

Returns

A successful response with a “Spam” header showing if the message is considered spam as well as the score. The body contains a comma-separated list of the symbols that were hit.

Raises

- **BadResponse** – If the response from SPAMD is ill-formed this exception will be raised.
- **AIOSpamcConnectionFailed** – Raised if an error occurred when trying to connect.

- **UsageException** – Error in command line usage.
- **DataErrorException** – Error with data format.
- **NoInputException** – Cannot open input.
- **NoUserException** – Addressee unknown.
- **NoHostException** – Hostname unknown.
- **UnavailableException** – Service unavailable.
- **InternalSoftwareException** – Internal software error.
- **OSErrorException** – System error.
- **OSFileException** – Operating system file missing.
- **CantCreateException** – Cannot create output file.
- **IOErrorException** – Input/output error.
- **TemporaryFailureException** – Temporary failure, may reattempt.
- **ProtocolException** – Error in the protocol.
- **NoPermissionException** – Permission denied.
- **ConfigException** – Error in configuration.
- **ServerTimeoutException** – Server returned a response that it timed out.
- **ClientTimeoutException** – Client timed out during connection.

```

async aiospamc.frontend.tell(message: bytes | SupportsBytes, message_class: str | MessageClassOption,
                               remove_action: str | ActionOption | None = None, set_action: str |
                               ActionOption | None = None, *, host: str = 'localhost', port: int = 783,
                               socket_path: Path | None = None, timeout: Timeout | None = None, verify:
                               bool | Path | SSLContext | None = None, cert: Path | Tuple[Path, Path | None] |
                               Tuple[Path, Path | None, str | None] | None = None, user: str | None = None,
                               compress: bool = False) → Response

```

Checks a message if it's spam and return a response with a score header.

Parameters

- **message** – Copy of the message.
- **message_class** – Classify the message as 'spam' or 'ham'.
- **remove_action** – Remove message class for message in database.
- **set_action** – Set message class for message in database.
- **host** – Hostname or IP address of the SPAMD service, defaults to localhost.
- **port** – Port number for the SPAMD service, defaults to 783.
- **socket_path** – Path to Unix socket.
- **timeout** – Timeout settings.
- **verify** – Enable SSL. *True* will use the root certificates from the *certifi* package. *False* will use SSL, but not verify the root certificates. Passing a string to a filename will use the path to verify the root certificates.

- **cert** – Use client certificate. Can either be a `pathlib.Path` to a file that includes both the certificate and key. Can be a tuple containing a `pathlib.Path` to the certificate and a `pathlib.Path` to the key. Can be a tuple containing a `pathlib.Path` to the certificate, `pathlib.Path` to the key, and a password.
- **user** – Username to pass to the SPAMD service.
- **compress** – Enable compress of the request body.

Returns

A successful response with “DidSet” and/or “DidRemove” headers along with the actions that were taken.

Raises

- **BadResponse** – If the response from SPAMD is ill-formed this exception will be raised.
- **AIOSpamcConnectionFailed** – Raised if an error occurred when trying to connect.
- **UsageException** – Error in command line usage.
- **DataErrorException** – Error with data format.
- **NoInputException** – Cannot open input.
- **NoUserException** – Addressee unknown.
- **NoHostException** – Hostname unknown.
- **UnavailableException** – Service unavailable.
- **InternalSoftwareException** – Internal software error.
- **OSErrorException** – System error.
- **OSFileException** – Operating system file missing.
- **CantCreateException** – Cannot create output file.
- **IOErrorException** – Input/output error.
- **TemporaryFailureException** – Temporary failure, may reattempt.
- **ProtocolException** – Error in the protocol.
- **NoPermissionException** – Permission denied.
- **ConfigException** – Error in configuration.
- **ServerTimeoutException** – Server returned a response that it timed out.
- **ClientTimeoutException** – Client timed out during connection.

aiospamc.header_values module

Collection of request and response header value objects.

protocol aiospamc.header_values.**HeaderValue**

Bases: `Protocol`

Protocol for headers.

Classes that implement this protocol must have the following methods / attributes:

`__bytes__()` → `bytes`

`to_json()` → *Any*

Convert to a JSON object.

class aiospamc.header_values.**BytesHeaderValue**(*value: bytes*)

Bases: *object*

Header with bytes value.

Parameters

value – Value of the header.

value: *bytes*

`to_json()` → *Any*

Converts object to a JSON serializable object.

`__init__`(*value: bytes*) → *None*

class aiospamc.header_values.**GenericHeaderValue**(*value: str, encoding: str = 'utf8'*)

Bases: *object*

Generic header value.

value: *str*

encoding: *str = 'utf8'*

`to_json()` → *Any*

Converts object to a JSON serializable object.

`__init__`(*value: str, encoding: str = 'utf8'*) → *None*

class aiospamc.header_values.**CompressValue**(*algorithm: str = 'zlib'*)

Bases: *object*

Compress header. Specifies what encryption scheme to use. So far only ‘zlib’ is supported.

algorithm: *str = 'zlib'*

`to_json()` → *Any*

Converts object to a JSON serializable object.

`__init__`(*algorithm: str = 'zlib'*) → *None*

class aiospamc.header_values.**ContentLengthValue**(*length: int = 0*)

Bases: *object*

ContentLength header. Indicates the length of the body in bytes.

length: *int = 0*

`to_json()` → *Any*

Converts object to a JSON serializable object.

`__init__`(*length: int = 0*) → *None*

class aiospamc.header_values.**MessageClassOption**(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: *str, Enum*

Option to be used for the MessageClass header.

```
spam = 'spam'
```

```
ham = 'ham'
```

```
class aiospamc.header_values.MessageClassValue(value: MessageClassOption =
                                              MessageClassOption.ham)
```

Bases: `object`

MessageClass header. Used to specify whether a message is ‘spam’ or ‘ham.’

```
value: MessageClassOption = 'ham'
```

```
to_json() → Any
```

Converts object to a JSON serializable object.

```
__init__(value: MessageClassOption = MessageClassOption.ham) → None
```

```
class aiospamc.header_values.ActionOption(local: bool = False, remote: bool = False)
```

Bases: `object`

Option to be used in the DidRemove, DidSet, Set, and Remove headers.

Parameters

- **local** – An action will be performed on the SPAMD service’s local database.
- **remote** – An action will be performed on the SPAMD service’s remote database.

```
local: bool = False
```

```
remote: bool = False
```

```
__init__(local: bool = False, remote: bool = False) → None
```

```
class aiospamc.header_values.SetOrRemoveValue(action: ActionOption)
```

Bases: `object`

Base class for headers that implement “local” and “remote” rules.

```
action: ActionOption
```

```
to_json() → Any
```

Converts object to a JSON serializable object.

```
__init__(action: ActionOption) → None
```

```
class aiospamc.header_values.SpamValue(value: bool = False, score: float = 0.0, threshold: float = 0.0)
```

Bases: `object`

Spam header. Used by the SPAMD service to report on if the submitted message was spam and the score/threshold that it used.

```
value: bool = False
```

```
score: float = 0.0
```

```
threshold: float = 0.0
```

```
to_json() → Any
```

Converts object to a JSON serializable object.

`__init__(value: bool = False, score: float = 0.0, threshold: float = 0.0) → None`

class aiospamc.header_values.UserValue(name: str = 'docs')

Bases: `object`

User header. Used to specify which user the SPAMD service should use when loading configuration files.

name: `str` = 'docs'

to_json() → `Any`

Converts object to a JSON serializable object.

`__init__(name: str = 'docs') → None`

class aiospamc.header_values.Headers(dict=None, /, **kwargs)

Bases: `UserDict`

Class to store headers with shortcut properties.

get_header(name: str) → `str` | `None`

Get a string header if it exists.

Parameters

name – Name of the header.

Returns

The header value.

set_header(name: str, value: str)

Sets a string header.

Parameters

- **name** – Name of the header.
- **value** – Value of the header.

get_bytes_header(name: str) → `bytes` | `None`

Get a bytes header if it exists.

Parameters

name – Name of the header.

Returns

The header value.

set_bytes_header(name: str, value: bytes)

Sets a string header.

Parameters

- **name** – Name of the header.
- **value** – Value of the header.

property compress: `str` | `None`

Gets the Compress header if it exists.

Returns

Compress header value.

property content_length: `int` | `None`

Gets the Content-length header if it exists.

Returns

Content-length header value.

property message_class: `MessageClassOption` | `None`

Gets the Message-class header if it exists.

Returns

Message-class header value.

property set_: `ActionOption` | `None`

Gets the Set header if it exists.

Returns

Set header value.

property remove: `ActionOption` | `None`

Gets the Remove header if it exists.

Returns

Remove header value.

property did_set: `ActionOption` | `None`

Gets the DidSet header if it exists.

Returns

DidSet header value.

property did_remove: `ActionOption` | `None`

Gets the DidRemove header if it exists.

Returns

DidRemove header value.

property spam: `SpamValue` | `None`

Gets the Spam header if it exists.

Returns

Spam header value.

property user: `str` | `None`

Gets the User header if it exists.

Returns

User header value.

aiospamc.incremental_parser module

Module for the parsing functions and objects.

class aiospamc.incremental_parser.States(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `Enum`

States for the parser state machine.

Status = 1

Header = 2

Body = 3

Done = 4

```
class aiospamc.incremental_parser.Parser(delimiter: bytes, status_parser: Callable[[bytes], Mapping[str, str]], header_parser: Callable[[bytes], Tuple[str, Any]], body_parser: Callable[[bytes, int], bytes], start: States = States.Status)
```

Bases: `object`

The parser state machine.

Variables

result – Storage location for parsing results.

```
__init__(delimiter: bytes, status_parser: Callable[[bytes], Mapping[str, str]], header_parser: Callable[[bytes], Tuple[str, Any]], body_parser: Callable[[bytes, int], bytes], start: States = States.Status) → None
```

Parser constructor.

Parameters

- **delimiter** – Byte string to split the different sections of the message.
- **status_parser** – Callable to parse the status line of the message.
- **header_parser** – Callable to parse each header line of the message.
- **body_parser** – Callable to parse the body of the message.
- **start** – The state to start the parser on. Allowed for easier testing.

property state: `States`

The current state of the parser.

Returns

The `States` instance.

```
parse(stream: bytes) → Mapping[str, Any]
```

Entry method to parse a message.

Parameters

stream – Byte string to parse.

Returns

Returns the parser results dictionary stored in the class attribute **result**.

Raises

- **NotEnoughDataError** – Raised when not enough data is sent to be parsed.
- **TooMuchDataError** – Raised when too much data is sent to be parsed.
- **ParseError** – Raised when a general parse error is found.

```
status() → None
```

Splits the message at the delimiter and sends the first part of the message to the **status_parser** callable to be parsed. If successful then the results are stored in the **result** class attribute and the state transitions to `States.Header`.

Raises

- ***NotEnoughDataError*** – When there is no delimiter the message is incomplete.
- ***ParseError*** – When the `status_parser` callable experiences an error.

header() → *None*

Splits the message at the delimiter and sends the line to the `header_parser`.

When splitting the action will be determined depending what is matched:

Header line	Delimiter	Left-over	Action
No	Yes	Delimiter	Headers done, empty body. Clear buffer and transition to <i>States.Body</i> .
No	Yes	N/A	Headers done. Transition to <i>States.Body</i> .
Yes	Yes	N/A	Parse header. Record in <code>result</code> class attribute.
No	No	No	Message was a status line only. Transition to <i>States.Body</i> .

Raises

ParseError – None of the previous conditions are matched.

body() → *None*

Uses the length defined in the *Content-length* header (defaulted to 0) to determine how many bytes the body contains.

Raises

TooMuchDataError – When there are too many bytes in the buffer compared to the *Content-length* header value. Transitions the state to *States.Done*.

`aiosпамc.incremental_parser.parse_request_status(stream: bytes) → Dict[str, str]`

Parses the status line from a request.

Parameters

stream – The byte stream to parse.

Returns

A dictionary with the keys *verb*, *protocol* and *version*.

Raises

ParseError – When the status line is in an invalid format, not a valid verb, or doesn't have the correct protocol.

`aiosпамc.incremental_parser.parse_response_status(stream: bytes) → Dict[str, str | int]`

Parse the status line for a response.

Parameters

stream – The byte stream to parse.

Returns

A dictionary with the keys *protocol*, *version*, *status_code*, and *message*.

Raises

ParseError – When the status line is in an invalid format, status code is not an integer, or doesn't have the correct protocol.

`aiosпамc.incremental_parser.parse_message_class_value(stream: str | MessageClassOption) → MessageClassValue`

Parses the *Message-class* header value.

Parameters

stream – String or *aiospamc.header_values.MessageClassOption* instance.

Returns

A *aiospamc.header_values.MessageClassValue* instance representing the value.

Raises

ParseError – When the value doesn't match either *ham* or *spam*.

`aiospamc.incremental_parser.parse_content_length_value(stream: str | int) → ContentLengthValue`

Parses the *Content-length* header value.

Parameters

stream – String or integer value of the header.

Returns

A *aiospamc.header_values.ContentLengthValue* instance.

Raises

ParseError – When the value cannot be cast to an integer.

`aiospamc.incremental_parser.parse_compress_value(stream: str) → CompressValue`

Parses a value for the *Compress* header.

Parameters

stream – String to parse.

Returns

A *aiospamc.header_values.CompressValue* instance.

`aiospamc.incremental_parser.parse_set_remove_value(stream: ActionOption | str) → SetOrRemoveValue`

Parse a value for the *aiospamc.header_values.DidRemove*, *aiospamc.header_values.DidSet*, *aiospamc.header_values.Remove*, and *aiospamc.header_values.Set* headers.

Parameters

stream – String to parse or an instance of *aiospamc.header_values.ActionOption*.

Returns

A *aiospamc.header_values.SetOrRemoveValue* instance.

`aiospamc.incremental_parser.parse_spam_value(stream: str) → SpamValue`

Parses the values for the *Spam* header.

Parameters

stream – String to parse.

Returns

A *aiospamc.header_values.SpamValue* instance.

Raises

ParseError – Raised if there is no true/false value, or valid numbers for the score or threshold.

`aiospamc.incremental_parser.parse_user_value(stream: str) → UserValue`

Parse the username.

Parameters

stream – String of username to parse. Whitespace is trimmed.

Returns

The *aiospamc.header_values.UserValue* instance.

`aiospamc.incremental_parser.parse_header_value(header: str, value: str | bytes) → Any`

Sends the header value stream to the header value parsing function.

Parameters

- **header** – Name of the header.
- **value** – String or byte stream of the header value.

Returns

The `aiospamc.header_values.HeaderValue` instance from the parsing function.

`aiospamc.incremental_parser.parse_header(stream: bytes) → Tuple[str, Any]`

Splits the header line and sends to the header parsing function.

Parameters

stream – Byte stream of the header line.

Returns

A tuple of the header name and value.

`aiospamc.incremental_parser.parse_body(stream: bytes, content_length: int) → bytes`

Parses the body of a message.

Parameters

- **stream** – Byte stream for the body.
- **content_length** – Expected length of the body in bytes.

Returns

Byte stream of the body.

Raises

- `NotEnoughDataError` – If the length is less than the stream.
- `TooMuchDataError` – If the length is more than the stream.

```
aiospamc.incremental_parser.header_value_parsers = {'Compress': <function
parse_compress_value>, 'Content-length': <function parse_content_length_value>,
'DidRemove': <function parse_set_remove_value>, 'DidSet': <function
parse_set_remove_value>, 'Message-class': <function parse_message_class_value>,
'Remove': <function parse_set_remove_value>, 'Set': <function parse_set_remove_value>,
'Spam': <function parse_spam_value>, 'User': <function parse_user_value>}
```

Mapping for header names to their parsing functions.

`class aiospamc.incremental_parser.RequestParser`

Bases: `Parser`

Sub-class of the parser for requests.

`__init__()`

RequestParser constructor.

`class aiospamc.incremental_parser.ResponseParser`

Bases: `Parser`

Sub-class of the parser for responses.

`__init__()`

ResponseParser constructor.

aiospamc.requests module

Contains all requests that can be made to the SPAMD service.

```
class aiospamc.requests.Request(verb: str, version: str = '1.5', headers: Dict[str, Any] | Headers | None = None, body: bytes | SupportsBytes = b'', **_)
```

Bases: `object`

SPAMC request object.

```
__init__(verb: str, version: str = '1.5', headers: Dict[str, Any] | Headers | None = None, body: bytes | SupportsBytes = b'', **_) → None
```

Request constructor.

Parameters

- **verb** – Method name of the request.
- **version** – Version of the protocol.
- **headers** – Collection of headers to be added.
- **body** – Byte string representation of the body.

property body: `bytes`

Body property getter.

Returns

Value of body.

to_json()

Converts to JSON serializable object.

aiospamc.responses module

Contains classes used for responses.

```
class aiospamc.responses.Status(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)
```

Bases: `IntEnum`

Enumeration for the status values defined by SPAMD.

EX_OK = 0

EX_USAGE = 64

EX_DATAERR = 65

EX_NOINPUT = 66

EX_NOUSER = 67

EX_NOHOST = 68

EX_UNAVAILABLE = 69

EX_SOFTWARE = 70

EX_OSERR = 71

`EX_OSFILE = 72`

`EX_CANTCREAT = 73`

`EX_IOERR = 74`

`EX_TEMPFAIL = 75`

`EX_PROTOCOL = 76`

`EX_NOPERM = 77`

`EX_CONFIG = 78`

`EX_TIMEOUT = 79`

```
class aiospamc.responses.Response(version: str = '1.5', status_code: Status | int = 0, message: str = "",
                                   headers: Dict[str, Any] | Headers | None = None, body: bytes = b'', **_)
```

Bases: `object`

Class to encapsulate response.

```
__init__(version: str = '1.5', status_code: Status | int = 0, message: str = "", headers: Dict[str, Any] |
          Headers | None = None, body: bytes = b'', **_)
```

Response constructor.

Parameters

- **version** – Version reported by the SPAMD service response.
- **status_code** – Success or error code.
- **message** – Message associated with status code.
- **body** – Byte string representation of the body.
- **headers** – Collection of headers to be added.

property status_code: `Status | int`

Status code property getter.

Returns

Value of status code.

property body: `bytes`

Body property getter.

Returns

Value of body.

raise_for_status() → `None`

Raises an exception if the status code isn't zero.

Raises

- `ResponseException` –
- `UsageException` –
- `DataErrorException` –
- `NoInputException` –
- `NoUserException` –

- *NoHostException* –
- *UnavailableException* –
- *InternalSoftwareException* –
- *OSErrorException* –
- *OSFileException* –
- *CantCreateException* –
- *IOErrorException* –
- *TemporaryFailureException* –
- *ProtocolException* –
- *NoPermissionException* –
- *ConfigException* –
- *ServerTimeoutException* –

`to_json()` → Dict[str, Any]

Converts to JSON serializable object.

exception aiospamc.responses.**ResponseException**(code: int, message: str, response: Response)

Bases: *Exception*

Base class for exceptions raised from a response.

`__init__`(code: int, message: str, response: Response)

ResponseException constructor.

Parameters

- **code** – Response code number.
- **message** – Message response.

exception aiospamc.responses.**UsageException**(message: str, response: Response)

Bases: *ResponseException*

Command line usage error.

`__init__`(message: str, response: Response)

UsageException constructor.

Parameters

- **message** – Message response.

exception aiospamc.responses.**DataErrorException**(message: str, response: Response)

Bases: *ResponseException*

Data format error.

`__init__`(message: str, response: Response)

DataErrorException constructor.

Parameters

- **message** – Message response.

exception aiospamc.responses.NoInputException(message: str, response: Response)

Bases: ResponseException

Cannot open input.

__init__(message: str, response: Response)

NoInputException constructor.

Parameters

message – Message response.

exception aiospamc.responses.NoUserException(message: str, response: Response)

Bases: ResponseException

Addressee unknown.

__init__(message: str, response: Response)

NoUserException constructor.

Parameters

message – Message response.

exception aiospamc.responses.NoHostException(message: str, response: Response)

Bases: ResponseException

Hostname unknown.

__init__(message: str, response: Response)

NoHostException constructor.

Parameters

message – Message response.

exception aiospamc.responses.UnavailableException(message: str, response: Response)

Bases: ResponseException

Service unavailable.

__init__(message: str, response: Response)

UnavailableException constructor.

Parameters

message – Message response.

exception aiospamc.responses.InternalSoftwareException(message: str, response: Response)

Bases: ResponseException

Internal software error.

__init__(message: str, response: Response)

InternalSoftwareException constructor.

Parameters

message – Message response.

exception aiospamc.responses.OSErrorException(message: str, response: Response)

Bases: ResponseException

System error (e.g. can't fork the process).

__init__(message: *str*, response: *Response*)

OSErrorException constructor.

Parameters

message – Message response.

exception aiospamc.responses.OSFileException(message: *str*, response: *Response*)

Bases: *ResponseException*

Critical operating system file missing.

__init__(message: *str*, response: *Response*)

OSFileException constructor.

Parameters

message – Message response.

exception aiospamc.responses.CantCreateException(message: *str*, response: *Response*)

Bases: *ResponseException*

Can't create (user) output file.

__init__(message: *str*, response: *Response*)

CantCreateException constructor.

Parameters

message – Message response.

exception aiospamc.responses.IOErrorException(message: *str*, response: *Response*)

Bases: *ResponseException*

Input/output error.

__init__(message: *str*, response: *Response*)

IOErrorException constructor.

Parameters

message – Message response.

exception aiospamc.responses.TemporaryFailureException(message: *str*, response: *Response*)

Bases: *ResponseException*

Temporary failure, user is invited to try again.

__init__(message: *str*, response: *Response*)

TemporaryFailureException constructor.

Parameters

message – Message response.

exception aiospamc.responses.ProtocolException(message: *str*, response: *Response*)

Bases: *ResponseException*

Remote error in protocol.

__init__(message: *str*, response: *Response*)

ProtocolException constructor.

Parameters

message – Message response.

exception aiospamc.responses.NoPermissionException(message: str, response: Response)

Bases: ResponseException

Permission denied.

__init__(message: str, response: Response)

NoPermissionException constructor.

Parameters

message – Message response.

exception aiospamc.responses.ConfigException(message: str, response: Response)

Bases: ResponseException

Configuration error.

__init__(message: str, response: Response)

ConfigException constructor.

Parameters

message – Message response.

exception aiospamc.responses.ServerTimeoutException(message: str, response: Response)

Bases: ResponseException, TimeoutException

Timeout exception from the server.

__init__(message: str, response: Response)

ServerTimeoutException constructor.

Parameters

message – Message response.

aiospamc.user_warnings module

Functions to raise warnings based on user inputs.

aiospamc.user_warnings.raise_warnings(request: Request, connection: ConnectionManager)

Calls all warning functions.

Parameters

- **request** – Instance of a request.
- **connection** – Connection manager instance.

aiospamc.user_warnings.warn_spamd_bug_7183(request: Request, connection: ConnectionManager)

Warn on spamd bug if using compression with an SSL connection.

Parameters

- **request** – Instance of a request.
- **connection** – Connection manager instance.

Bug: https://bz.apache.org/SpamAssassin/show_bug.cgi?id=7183

Module contents

aiospamc package.

An asyncio-based library to communicate with SpamAssassin's SPAMD service.

1.4 SPAMC/SPAMD Protocol As Implemented by SpamAssassin

1.4.1 Requests and Responses

The structure of a request is similar to an HTTP request.¹ The method/verb, protocol name and version are listed followed by headers separated by newline characters (carriage return and linefeed or `\r\n`). Following the headers is a blank line with a newline (`\r\n`). If there is a message body it will be added after all headers.

The current requests are *CHECK*, *HEADERS*, *PING*, *PROCESS*, *REPORT*, *REPORT_IFSPAM*, *SKIP*, *SYMBOLS*, and *TELL*:

```
METHOD SPAMC/1.5\r\n
HEADER_NAME1: HEADER_VALUE1\r\n
HEADER_NAME2: HEADER_VALUE2\r\n
...
\r\n
REQUEST_BODY
```

The structure of responses are also similar to HTTP responses. The protocol name, version, status code, and message are listed on the first line. Any headers are also listed and all are separated by newline characters. Following the headers is a newline. If there is a message body it's included after all headers:

```
SPAMD/1.5 STATUS_CODE MESSAGE\r\n
HEADER_NAME1: HEADER_VALUE1\r\n
HEADER_NAME2: HEADER_VALUE2\r\n
...
\r\n
RESPONSE_BODY
```

Note: The header name and value are separated by a `:` character. For built-in headers the name must not have any whitespace surrounding it. It will be parsed exactly as it's represented.

The following are descriptions of the requests that can be sent and examples of the responses that you can expect to receive.

¹ <https://svn.apache.org/viewvc/spamassassin/branches/3.4/spamd/PROTOCOL?revision=1676616&view=co>

CHECK

Instruct SpamAssassin to process the included message.

Request

Required Headers

- *Content-length*

Optional Headers

- *Compress*
- *User*

Required body

An email based on the [RFC 5322](#) standard.

Response

Will include a Spam header with a “True” or “False” value, followed by the score and threshold. Example:

```
SPAMD/1.1 0 EX_OK
Spam: True ; 1000.0 / 5.0
```

HEADERS

Process the included message and return only the modified headers.

Request

Required Headers

- *Content-length*

Optional Headers

- *Compress*
- *User*

Required Body

An email based on the [RFC 5322](#) standard.

Response

Will return the modified headers of the message in the body. The *Spam* header is also included.

```

SPAMD/1.1 0 EX_OK
Spam: True ; 1000.0 / 5.0
Content-length: 654

Received: from localhost by debian
        with SpamAssassin (version 3.4.0);
        Tue, 10 Jan 2017 11:09:26 -0500
From: Sender <sender@example.net>
To: Recipient <recipient@example.net>
Subject: Test spam mail (GTUBE)
Date: Wed, 23 Jul 2003 23:30:00 +0200
Message-Id: <GTUBE1.1010101@example.net>
X-Spam-Checker-Version: SpamAssassin 3.4.0 (2014-02-07) on debian
X-Spam-Flag: YES
X-Spam-Level: *****
X-Spam-Status: Yes, score=1000.0 required=5.0 tests=GTUBE,NO_RECEIVED,
        NO_RELAYS autolearn=no autolearn_force=no version=3.4.0
MIME-Version: 1.0Content-Type: multipart/mixed; boundary="-----=_58750736.8D9F70BC"

```

PING

Send a request to test if the server is alive.

Request

Required Headers

None.

Optional Headers

None.

Response

Example:

```
SPAMD/1.5 0 PONG
```

PROCESS

Instruct SpamAssassin to process the message and return the modified message.

Request

Required Headers

- *Content-length*

Optional Headers

- *Compress*
- *User*

Required Body

An email based on the [RFC 5322](#) standard.

Response

Will return a modified message in the body. The *Spam* header is also included. Example:

```
SPAMD/1.1 0 EX_OK
Spam: True ; 1000.0 / 5.0
Content-length: 2948

Received: from localhost by debian
        with SpamAssassin (version 3.4.0);
        Tue, 10 Jan 2017 10:57:02 -0500
From: Sender <sender@example.net>
To: Recipient <recipient@example.net>
Subject: Test spam mail (GTUBE)
Date: Wed, 23 Jul 2003 23:30:00 +0200
Message-Id: <GTUBE1.1010101@example.net>
X-Spam-Checker-Version: SpamAssassin 3.4.0 (2014-02-07) on debian
X-Spam-Flag: YES
X-Spam-Level: *****
X-Spam-Status: Yes, score=1000.0 required=5.0 tests=GTUBE,NO_RECEIVED,
        NO_RELAYS autolearn=no autolearn_force=no version=3.4.0
MIME-Version: 1.0
```

(continues on next page)

(continued from previous page)

Content-Type: multipart/mixed; boundary="-----=_5875044E.D4EFFF7D"

This is a multi-part message in MIME format.

-----=_5875044E.D4EFFF7D

Content-Type: text/plain; charset=iso-8859-1

Content-Disposition: inline

Content-Transfer-Encoding: 8bit

Spam detection software, running on the system "debian", has identified this incoming email as possible spam. The original message has been attached to this so you can view it or label similar future email. If you have any questions, see @@CONTACT_ADDRESS@@ for details.

Content preview: This is the GTUBE, the Generic Test for Unsolicited Bulk Email. If your spam filter supports it, the GTUBE provides a test by which you can verify that the filter is installed correctly and is detecting incoming spam. You can send yourself a test mail containing the following string of characters (in upper case and with no white spaces and line breaks): [...]

Content analysis details: (1000.0 points, 5.0 required)

pts	rule name	description
1000	GTUBE	BODY: Generic Test for Unsolicited Bulk Email
-0.0	NO_RELAYS	Informational: message was not relayed via SMTP
-0.0	NO_RECEIVED	Informational: message has no Received headers

-----=_5875044E.D4EFFF7D

Content-Type: message/rfc822; x-spam-type=original

Content-Description: original message before SpamAssassin

Content-Disposition: inline

Content-Transfer-Encoding: 8bit

Subject: Test spam mail (GTUBE)

Message-ID: <GTUBE1.1010101@example.net>

Date: Wed, 23 Jul 2003 23:30:00 +0200

From: Sender <sender@example.net>

To: Recipient <recipient@example.net>

Precedence: junk

MIME-Version: 1.0

Content-Type: text/plain; charset=us-ascii

Content-Transfer-Encoding: 7bit

This is the GTUBE, the
Generic
Test for
Unsolicited
Bulk

(continues on next page)

(continued from previous page)

Email

If your spam filter supports it, the GTUBE provides a test by which you can verify that the filter is installed correctly and is detecting incoming spam. You can send yourself a test mail containing the following string of characters (in upper case and with no white spaces and line breaks):

```
XJS*C4JDBQADN1.NSBN3*2IDNEN*GTUBE-STANDARD-ANTI-UBE-TEST-EMAIL*C.34X
```

You should send this test mail from an account outside of your network.

```
-----=_5875044E.D4EFFF7D7--
```

REPORT

Send a request to process a message and return a report.

Request

Required Headers

- *Content-length*

Optional Headers

- *Compress*
- *User*

Required body

An email based on the [RFC 5322](#) standard.

Response

Response returns the *Spam* header and the body containing a report of the message scanned.

Example:

```
SPAMD/1.1 0 EX_OK
Content-length: 1071
Spam: True ; 1000.0 / 5.0
```

Spam detection software, running on the system "debian", has identified this incoming email as possible spam. The original message has been attached to this so you can view it or label similar future email. If you have any questions, see

(continues on next page)

(continued from previous page)

@@CONTACT_ADDRESS@@ for details.

Content preview: This is the GTUBE, the Generic Test for Unsolicited Bulk Email
 If your spam filter supports it, the GTUBE provides a test by which you can
 verify that the filter is installed correctly and is detecting incoming spam.
 You can send yourself a test mail containing the following string of characters
 (in upper case and with no white spaces and line breaks): [...]

Content analysis details: (1000.0 points, 5.0 required)

pts	rule name	description
-----	-----	-----
1000	GTUBE	BODY: Generic Test for Unsolicited Bulk Email
-0.0	NO_RELAYS	Informational: message was not relayed via SMTP
-0.0	NO_RECEIVED	Informational: message has no Received headers

REPORT_IFSPAM

Matches the *REPORT* request, with the exception a report will not be generated if the message is not spam.

SKIP

Sent when a connection is made in error. The SPAMD service will immediately close the connection.

Request

Required Headers

None.

Optional Headers

None.

SYMBOLS

Instruct SpamAssassin to process the message and return the rules that were matched.

Request

Required Headers

- *Content-length*

Optional Headers

- *Compress*
- *User*

Required body

An email based on the [RFC 5322](#) standard.

Response

Response includes the *Spam* header. The body contains the SpamAssassin rules that were matched. Example:

```
SPAMD/1.1 0 EX_OK
Content-length: 27
Spam: True ; 1000.0 / 5.0

GTUBE,NO_RECEIVED,NO_RELAYS
```

TELL

Send a request to classify a message and add or remove it from a database. The message type is defined by the *Message-class*. The *Remove* and *Set* headers are used to choose the location (“local” or “remote”) to add or remove it. SpamAssassin will return an error if a request tries to apply a conflicting change (e.g. both setting and removing to the same location).

Note: The SpamAssassin daemon must have the `--allow-tell` option enabled to support this feature.

Request

Required Headers

- *Content-length*
- *Message-class*
- *Remove* and/or *Set*
- *User*

Optional Headers

- *Compress*

Required Body

An email based on the [RFC 5322](#) standard.

Response

If successful, the response will include the *DidRemove* and/or *DidSet* headers depending on the request.

Response from a request that sent a *Remove*:

```
SPAMD/1.1 0 EX_OK
DidRemove: local
Content-length: 2
```

Response from a request that sent a *Set*:

```
SPAMD/1.1 0 EX_OK
DidSet: local
Content-length: 2
```

1.4.2 Headers

Headers are structured very simply. They have a name and value which are separated by a colon (:). All headers are followed by a newline. The current headers include *Compress*, *Content-length*, *DidRemove*, *DidSet*, *Message-class*, *Remove*, *Set*, *Spam*, and *User*.

For example:

```
Content-length: 42\r\n
```

The following is a list of headers defined by SpamAssassin, although anything is allowable as a header. If an unrecognized header is included in the request or response it should be ignored.

Compress

Specifies that the body is compressed and what compression algorithm is used. Contains a string of the compression algorithm. Currently only *zlib* is supported.

Content-length

The length of the body in bytes. Contains an integer representing the body length.

DidRemove

Included in a response to a *TELL* request. Identifies which databases a message was removed from. Contains a string containing either *local*, *remote* or both separated by a comma.

DidSet

Included in a response to a *TELL* request. Identifies which databases a message was set in. Contains a string containing either *local*, *remote* or both separated by a comma.

Message-class

Classifies the message contained in the body. Contains a string containing either *local*, *remote* or both separated by a comma.

Remove

Included in a *TELL* request to remove the message from the specified database. Contains a string containing either *local*, *remote* or both separated by a comma.

Set

Included in a *TELL* request to remove the message from the specified database. Contains a string containing either *local*, *remote* or both separated by a comma.

Spam

Identify whether the message submitted was spam or not including the score and threshold. Contains a string containing a boolean if the message is spam (either *True*, *False*, *Yes*, or *No*), followed by a *;*, a floating point number representing the score, followed by a */*, and finally a floating point number representing the threshold of which to consider it spam.

For example:

```
Spam: True ; 1000.0 / 5.0
```

User

Specify which user the request will run under. SpamAssassin will use the configuration files for the user included in the header. Contains a string containing the name of the user.

1.4.3 Status Codes

A status code is an integer detailing whether the request was successful or if an error occurred.

The following status codes are defined in the SpamAssassin source repository².

EX_OK

Code: 0

Definition: No problems were found.

EX_USAGE

Code: 64

Definition: Command line usage error.

EX_DATAERR

Code: 65

Definition: Data format error.

EX_NOINPUT

Code: 66

Definition: Cannot open input.

EX_NOUSER

Code: 67

Definition: Addressee unknown.

EX_NOHOST

Code: 68

Definition: Hostname unknown.

² <https://svn.apache.org/viewvc/spamassassin/branches/3.4/spamd/spamd.raw?revision=1749346&view=co>

EX_UNAVAILABLE

Code: 69

Definition: Service unavailable.

EX_SOFTWARE

Code: 70

Definition: Internal software error.

EX_OSERR

Code: 71

Definition: System error (e.g. can't fork the process).

EX_OSFILE

Code: 72

Definition: Critical operating system file missing.

EX_CANTCREAT

Code: 73

Definition: Can't create (user) output file.

EX_IOERR

Code: 74

Definition: Input/output error.

EX_TEMPFAIL

Code: 75

Definition: Temporary failure, user is invited to retry.

EX_PROTOCOL

Code: 76

Definition: Remote error in protocol.

EX_NOPERM

Code: 77

Definition: Permission denied.

EX_CONFIG

Code: 78

Definition: Configuration error.

EX_TIMEOUT

Code: 79

Definition: Read timeout.

1.4.4 Body

SpamAssassin will generally want the body of a request to be in a supported RFC email format. The response body will differ depending on the type of request that was sent.

1.4.5 References

1.5 Release Notes

1.5.1 v1.0.0

New Features

- Marked package as stable [#320](#)
- Adding support for using client certificates. [#404](#)
- Added support for Python 3.12 [#422](#)

Bug Fixes

- Changed `-ssl` CLI option so it accepts a boolean instead of an optional boolean. [#394](#)
- CLI `-ssl` option type changed from optional boolean to boolean. [#394](#)
- Fixed `report` CLI sub-command that did not set the `User` header. [#385](#)

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

a

- `aiospamc`, 45
- `aiospamc.cli`, 10
- `aiospamc.client`, 13
- `aiospamc.connections`, 14
- `aiospamc.exceptions`, 19
- `aiospamc.frontend`, 20
- `aiospamc.header_values`, 30
- `aiospamc.incremental_parser`, 34
- `aiospamc.requests`, 39
- `aiospamc.responses`, 39
- `aiospamc.user_warnings`, 44

INDEX

Symbols

<code>__bytes__()</code> (<i>aiospamc.header_values.HeaderValue</i> method), 30	<code>__init__()</code> (<i>aiospamc.header_values.UserValue</i> method), 33
<code>__init__()</code> (<i>aiospamc.cli.CliClientBuilder</i> method), 10	<code>__init__()</code> (<i>aiospamc.incremental_parser.Parser</i> method), 35
<code>__init__()</code> (<i>aiospamc.cli.CommandRunner</i> method), 11	<code>__init__()</code> (<i>aiospamc.incremental_parser.RequestParser</i> method), 38
<code>__init__()</code> (<i>aiospamc.client.Client</i> method), 13	<code>__init__()</code> (<i>aiospamc.incremental_parser.ResponseParser</i> method), 38
<code>__init__()</code> (<i>aiospamc.connections.ConnectionManager</i> method), 15	<code>__init__()</code> (<i>aiospamc.requests.Request</i> method), 39
<code>__init__()</code> (<i>aiospamc.connections.ConnectionManagerBuilder</i> method), 15	<code>__init__()</code> (<i>aiospamc.responses.CantCreateException</i> method), 43
<code>__init__()</code> (<i>aiospamc.connections.SSLContextBuilder</i> method), 18	<code>__init__()</code> (<i>aiospamc.responses.ConfigException</i> method), 44
<code>__init__()</code> (<i>aiospamc.connections.TcpConnectionManager</i> method), 17	<code>__init__()</code> (<i>aiospamc.responses.DataErrorException</i> method), 41
<code>__init__()</code> (<i>aiospamc.connections.TcpConnectionManagerBuilder</i> method), 16	<code>__init__()</code> (<i>aiospamc.responses.IOErrorException</i> method), 43
<code>__init__()</code> (<i>aiospamc.connections.Timeout</i> method), 14	<code>__init__()</code> (<i>aiospamc.responses.InternalSoftwareException</i> method), 42
<code>__init__()</code> (<i>aiospamc.connections.UnixConnectionManager</i> method), 18	<code>__init__()</code> (<i>aiospamc.responses.NoHostException</i> method), 42
<code>__init__()</code> (<i>aiospamc.connections.UnixConnectionManagerBuilder</i> method), 17	<code>__init__()</code> (<i>aiospamc.responses.NoInputException</i> method), 42
<code>__init__()</code> (<i>aiospamc.exceptions.ParseError</i> method), 20	<code>__init__()</code> (<i>aiospamc.responses.NoPermissionException</i> method), 44
<code>__init__()</code> (<i>aiospamc.frontend.FrontendClientBuilder</i> method), 20	<code>__init__()</code> (<i>aiospamc.responses.NoUserException</i> method), 42
<code>__init__()</code> (<i>aiospamc.header_values.ActionOption</i> method), 32	<code>__init__()</code> (<i>aiospamc.responses.OSErrorException</i> method), 42
<code>__init__()</code> (<i>aiospamc.header_values.BytesHeaderValue</i> method), 31	<code>__init__()</code> (<i>aiospamc.responses.OSFileException</i> method), 43
<code>__init__()</code> (<i>aiospamc.header_values.CompressValue</i> method), 31	<code>__init__()</code> (<i>aiospamc.responses.ProtocolException</i> method), 43
<code>__init__()</code> (<i>aiospamc.header_values.ContentLengthValue</i> method), 31	<code>__init__()</code> (<i>aiospamc.responses.Response</i> method), 40
<code>__init__()</code> (<i>aiospamc.header_values.GenericHeaderValue</i> method), 31	<code>__init__()</code> (<i>aiospamc.responses.ResponseException</i> method), 41
<code>__init__()</code> (<i>aiospamc.header_values.MessageClassValue</i> method), 32	<code>__init__()</code> (<i>aiospamc.responses.ServerTimeoutException</i> method), 44
<code>__init__()</code> (<i>aiospamc.header_values.SetOrRemoveValue</i> method), 32	<code>__init__()</code> (<i>aiospamc.responses.TemporaryFailureException</i> method), 43
<code>__init__()</code> (<i>aiospamc.header_values.SpamValue</i> method), 32	<code>__init__()</code> (<i>aiospamc.responses.UnavailableException</i> method), 43

method), 42
__init__() (*aiospamc.responses.UsageException*
method), 41
--ca-cert
 aiospamc command line option, 4–7
--client-cert
 aiospamc command line option, 4–7
--client-key
 aiospamc command line option, 4–7
--debug
 command line option, 4
--host
 aiospamc command line option, 4–7
--key-password
 aiospamc command line option, 4–7
--out
 aiospamc command line option, 4–8
--port
 aiospamc command line option, 4–7
--socket-path
 aiospamc command line option, 4–7
--ssl
 aiospamc command line option, 4–7
--timeout
 aiospamc command line option, 4–7
--user
 aiospamc command line option, 4–7
--version
 command line option, 4
-h
 aiospamc command line option, 4–7
-p
 aiospamc command line option, 4–7

A

action (*aiospamc.header_values.SetOrRemoveValue* at-
tribute), 32
ActionOption (class in *aiospamc.header_values*), 32
add_ca() (*aiospamc.connections.SSLContextBuilder*
method), 19
add_ca_cert() (*aiospamc.cli.CliClientBuilder*
method), 11
add_ca_dir() (*aiospamc.connections.SSLContextBuilder*
method), 19
add_ca_file() (*aiospamc.connections.SSLContextBuilder*
method), 18
add_client() (*aiospamc.connections.SSLContextBuilder*
method), 19
add_client_cert() (*aiospamc.cli.CliClientBuilder*
method), 11
add_client_cert() (*aiospamc.frontend.FrontendClientBuilder*
method), 21
add_default_ca() (*aiospamc.connections.SSLContextBuilder*
method), 19
add_ssl_context() (*aiospamc.connections.ConnectionManagerBuilder*
method), 15
add_verify() (*aiospamc.cli.CliClientBuilder* *method*),
11
add_verify() (*aiospamc.frontend.FrontendClientBuilder*
method), 21
aiospamc
 module, 45
aiospamc command line option
 --ca-cert, 4–7
 --client-cert, 4–7
 --client-key, 4–7
 --host, 4–7
 --key-password, 4–7
 --out, 4–8
 --port, 4–7
 --socket-path, 4–7
 --ssl, 4–7
 --timeout, 4–7
 --user, 4–7
 -h, 4–7
 -p, 4–7
 check, 4
 forget, 4
 learn, 5
 ping, 6
 report, 6
 revoke, 7
aiospamc.cli
 module, 10
aiospamc.client
 module, 13
aiospamc.connections
 module, 14
aiospamc.exceptions
 module, 19
aiospamc.frontend
 module, 20
aiospamc.header_values
 module, 30
aiospamc.incremental_parser
 module, 34
aiospamc.requests
 module, 39
aiospamc.responses
 module, 39
aiospamc.user_warnings
 module, 44
AIOSpamcConnectionFailed, 20
algorithm (*aiospamc.header_values.CompressValue* at-
tribute), 31

B

BadRequest, 19

- BadResponse, 19
 Body (aiospamc.incremental_parser.States attribute), 35
 body (aiospamc.requests.Request property), 39
 body (aiospamc.responses.Response property), 40
 body() (aiospamc.incremental_parser.Parser method), 36
 build() (aiospamc.cli.CliClientBuilder method), 10
 build() (aiospamc.connections.ConnectionManagerBuilder method), 15
 build() (aiospamc.connections.SSLContextBuilder method), 18
 build() (aiospamc.connections.TcpConnectionManagerBuilder method), 16
 build() (aiospamc.connections.UnixConnectionManagerBuilder method), 17
 build() (aiospamc.frontend.FrontendClientBuilder method), 20
 BytesHeaderValue (class in aiospamc.header_values), 31
- ## C
- CantCreateException, 43
 check
 aiospamc command line option, 4
 check() (in module aiospamc.cli), 12
 check() (in module aiospamc.frontend), 21
 CliClientBuilder (class in aiospamc.cli), 10
 Client (class in aiospamc.client), 13
 ClientException, 19
 ClientTimeoutException, 20
 command line option
 --debug, 4
 --version, 4
 CommandRunner (class in aiospamc.cli), 11
 compress (aiospamc.header_values.Headers property), 33
 CompressValue (class in aiospamc.header_values), 31
 ConfigException, 44
 connection_string (aiospamc.connections.ConnectionManager property), 16
 ConnectionManager (class in aiospamc.connections), 15
 ConnectionManagerBuilder (class in aiospamc.connections), 14
 ConnectionManagerBuilder.ManagerType (class in aiospamc.connections), 14
 content_length (aiospamc.header_values.Headers property), 33
 ContentLengthValue (class in aiospamc.header_values), 31
- ## D
- DataErrorException, 41
 debug_callback() (in module aiospamc.cli), 13
 did_remove (aiospamc.header_values.Headers property), 34
 did_set (aiospamc.header_values.Headers property), 34
 Done (aiospamc.incremental_parser.States attribute), 35
 dont_verify() (aiospamc.connections.SSLContextBuilder method), 19
- ## E
- encoding (aiospamc.header_values.GenericHeaderValue attribute), 31
 environment variable
 AIOSPAMC_CERT_FILE, 8
- ## EX
- EX_CANTCREAT (aiospamc.responses.Status attribute), 40
 EX_CONFIG (aiospamc.responses.Status attribute), 40
 EX_DATAERR (aiospamc.responses.Status attribute), 39
 EX_IOERR (aiospamc.responses.Status attribute), 40
 EX_NOHOST (aiospamc.responses.Status attribute), 39
 EX_NOINPUT (aiospamc.responses.Status attribute), 39
 EX_NOPERM (aiospamc.responses.Status attribute), 40
 EX_NOUSER (aiospamc.responses.Status attribute), 39
 EX_OK (aiospamc.responses.Status attribute), 39
 EX_OSERR (aiospamc.responses.Status attribute), 39
 EX_OSFILE (aiospamc.responses.Status attribute), 39
 EX_PROTOCOL (aiospamc.responses.Status attribute), 40
 EX_SOFTWARE (aiospamc.responses.Status attribute), 39
 EX_TEMPFAIL (aiospamc.responses.Status attribute), 40
 EX_TIMEOUT (aiospamc.responses.Status attribute), 40
 EX_UNAVAILABLE (aiospamc.responses.Status attribute), 39
 EX_USAGE (aiospamc.responses.Status attribute), 39
 exit() (aiospamc.cli.CommandRunner method), 12
- ## F
- forget
 aiospamc command line option, 4
 forget() (in module aiospamc.cli), 12
 FrontendClientBuilder (class in aiospamc.frontend), 20
- ## G
- GenericHeaderValue (class in aiospamc.header_values), 31
 get_bytes_header() (aiospamc.header_values.Headers method), 33
 get_header() (aiospamc.header_values.Headers method), 33
- ## H
- ham (aiospamc.header_values.MessageClassOption attribute), 32
 Header (aiospamc.incremental_parser.States attribute), 34

header() (*aiospamc.incremental_parser.Parser* method), 36
header_value_parsers (in module *aiospamc.incremental_parser*), 38
Headers (class in *aiospamc.header_values*), 33
headers() (in module *aiospamc.frontend*), 22
HeaderValue (protocol in *aiospamc.header_values*), 30

I

InternalSoftwareException, 42
IOErrorException, 43

J

Json (*aiospamc.cli.Output* attribute), 10

L

learn
 aiospamc command line option, 5
learn() (in module *aiospamc.cli*), 12
length (*aiospamc.header_values.ContentLengthValue* attribute), 31
local (*aiospamc.header_values.ActionOption* attribute), 32
logger (*aiospamc.connections.ConnectionManager* property), 16

M

main() (in module *aiospamc.cli*), 13
message_class (*aiospamc.header_values.Headers* property), 34
MessageClassOption (class in *aiospamc.header_values*), 31
MessageClassValue (class in *aiospamc.header_values*), 32
module
 aiospamc, 45
 aiospamc.cli, 10
 aiospamc.client, 13
 aiospamc.connections, 14
 aiospamc.exceptions, 19
 aiospamc.frontend, 20
 aiospamc.header_values, 30
 aiospamc.incremental_parser, 34
 aiospamc.requests, 39
 aiospamc.responses, 39
 aiospamc.user_warnings, 44

N

name (*aiospamc.header_values.UserValue* attribute), 33
NoHostException, 42
NoInputException, 41
NoPermissionException, 43
NotEnoughDataError, 20

NoUserException, 42

O

open() (*aiospamc.connections.ConnectionManager* method), 16
open() (*aiospamc.connections.TcpConnectionManager* method), 17
open() (*aiospamc.connections.UnixConnectionManager* method), 18
OSErrorException, 42
OSFileException, 43
Output (class in *aiospamc.cli*), 10

P

parse() (*aiospamc.incremental_parser.Parser* method), 35
parse_body() (in module *aiospamc.incremental_parser*), 38
parse_compress_value() (in module *aiospamc.incremental_parser*), 37
parse_content_length_value() (in module *aiospamc.incremental_parser*), 37
parse_header() (in module *aiospamc.incremental_parser*), 38
parse_header_value() (in module *aiospamc.incremental_parser*), 37
parse_message_class_value() (in module *aiospamc.incremental_parser*), 36
parse_request_status() (in module *aiospamc.incremental_parser*), 36
parse_response_status() (in module *aiospamc.incremental_parser*), 36
parse_set_remove_value() (in module *aiospamc.incremental_parser*), 37
parse_spam_value() (in module *aiospamc.incremental_parser*), 37
parse_user_value() (in module *aiospamc.incremental_parser*), 37
ParseError, 20
Parser (class in *aiospamc.incremental_parser*), 35
ping
 aiospamc command line option, 6
ping() (in module *aiospamc.cli*), 12
ping() (in module *aiospamc.frontend*), 23
process() (in module *aiospamc.frontend*), 24
ProtocolException, 43

R

raise_for_status() (*aiospamc.responses.Response* method), 40
raise_warnings() (in module *aiospamc.user_warnings*), 44
read_message() (in module *aiospamc.cli*), 12

- `remote` (*aiospamc.header_values.ActionOption* attribute), 32
 - `remove` (*aiospamc.header_values.Headers* property), 34
 - `report`
 - aiospamc command line option, 6
 - `report()` (in module *aiospamc.cli*), 12
 - `report()` (in module *aiospamc.frontend*), 26
 - `report_if_spam()` (in module *aiospamc.frontend*), 27
 - `Request` (class in *aiospamc.requests*), 39
 - `request()` (*aiospamc.client.Client* method), 13
 - `request()` (*aiospamc.connections.ConnectionManager* method), 16
 - `RequestParser` (class in *aiospamc.incremental_parser*), 38
 - `Response` (class in *aiospamc.responses*), 40
 - `ResponseException`, 41
 - `ResponseParser` (class in *aiospamc.incremental_parser*), 38
 - `revoke`
 - aiospamc command line option, 7
 - `revoke()` (in module *aiospamc.cli*), 13
 - RFC
 - RFC 5322, 46–48, 50, 52, 53
 - `run()` (*aiospamc.cli.CommandRunner* method), 12
- ## S
- `score` (*aiospamc.header_values.SpamValue* attribute), 32
 - `ServerTimeoutException`, 44
 - `set_` (*aiospamc.header_values.Headers* property), 34
 - `set_bytes_header()` (*aiospamc.header_values.Headers* method), 33
 - `set_header()` (*aiospamc.header_values.Headers* method), 33
 - `set_host()` (*aiospamc.connections.TcpConnectionManagerBuilder* method), 31
 - `set_path()` (*aiospamc.connections.UnixConnectionManagerBuilder* method), 31
 - `set_port()` (*aiospamc.connections.TcpConnectionManagerBuilder* method), 31
 - `set_ssl_context()` (*aiospamc.connections.TcpConnectionManagerBuilder* method), 31
 - `set_timeout()` (*aiospamc.cli.CliClientBuilder* method), 11
 - `set_timeout()` (*aiospamc.connections.ConnectionManagerBuilder* method), 32
 - `set_timeout()` (*aiospamc.connections.TcpConnectionManagerBuilder* method), 32
 - `set_timeout()` (*aiospamc.connections.UnixConnectionManagerBuilder* method), 32
 - `set_timeout()` (*aiospamc.frontend.FrontendClientBuilder* method), 21
 - `SetOrRemoveValue` (class in *aiospamc.header_values*), 32
 - `spam` (*aiospamc.header_values.Headers* property), 34
 - `spam` (*aiospamc.header_values.MessageClassOption* attribute), 31
 - `SpamValue` (class in *aiospamc.header_values*), 32
 - `SSLContextBuilder` (class in *aiospamc.connections*), 18
 - `state` (*aiospamc.incremental_parser.Parser* property), 35
 - `States` (class in *aiospamc.incremental_parser*), 34
 - `Status` (*aiospamc.incremental_parser.States* attribute), 34
 - `Status` (class in *aiospamc.responses*), 39
 - `status()` (*aiospamc.incremental_parser.Parser* method), 35
 - `status_code` (*aiospamc.responses.Response* property), 40
 - `symbols()` (in module *aiospamc.frontend*), 28
- ## T
- `Tcp` (*aiospamc.connections.ConnectionManagerBuilder.ManagerType* attribute), 15
 - `TcpConnectionManager` (class in *aiospamc.connections*), 17
 - `TcpConnectionManagerBuilder` (class in *aiospamc.connections*), 16
 - `tell()` (in module *aiospamc.frontend*), 29
 - `TemporaryFailureException`, 43
 - `Text` (*aiospamc.cli.Output* attribute), 10
 - `threshold` (*aiospamc.header_values.SpamValue* attribute), 32
 - `Timeout` (class in *aiospamc.connections*), 14
 - `TimeoutException`, 20
 - `to_json()` (*aiospamc.cli.CommandRunner* method), 12
 - `to_json()` (*aiospamc.header_values.BytesHeaderValue* method), 31
 - `to_json()` (*aiospamc.header_values.CompressValue* method), 31
 - `to_json()` (*aiospamc.header_values.ContentLengthValue* method), 31
 - `to_json()` (*aiospamc.header_values.GenericHeaderValue* method), 31
 - `to_json()` (*aiospamc.header_values.HeaderValue* method), 30
 - `to_json()` (*aiospamc.header_values.MessageClassValue* method), 30
 - `to_json()` (*aiospamc.header_values.SetOrRemoveValue* method), 33
 - `to_json()` (*aiospamc.header_values.SpamValue* method), 33
 - `to_json()` (*aiospamc.header_values.UserValue* method), 33
 - `to_json()` (*aiospamc.requests.Request* method), 39
 - `to_json()` (*aiospamc.responses.Response* method), 41
 - `TooMuchDataError`, 20

U

`UnavailableException`, [42](#)

`Undefined` (*aiospamc.connections.ConnectionManagerBuilder.ManagerType* attribute), [14](#)

`Unix` (*aiospamc.connections.ConnectionManagerBuilder.ManagerType* attribute), [15](#)

`UnixConnectionManager` (class in *aiospamc.connections*), [18](#)

`UnixConnectionManagerBuilder` (class in *aiospamc.connections*), [17](#)

`UsageException`, [41](#)

`user` (*aiospamc.header_values.Headers* property), [34](#)

`UserValue` (class in *aiospamc.header_values*), [33](#)

V

`value` (*aiospamc.header_values.BytesHeaderValue* attribute), [31](#)

`value` (*aiospamc.header_values.GenericHeaderValue* attribute), [31](#)

`value` (*aiospamc.header_values.MessageClassValue* attribute), [32](#)

`value` (*aiospamc.header_values.SpamValue* attribute), [32](#)

`version_callback()` (in module *aiospamc.cli*), [13](#)

W

`warn_spamd_bug_7183()` (in module *aiospamc.user_warnings*), [44](#)

`with_connection()` (*aiospamc.cli.CliClientBuilder* method), [10](#)

`with_connection()` (*aiospamc.frontend.FrontendClientBuilder* method), [20](#)

`with_context()` (*aiospamc.connections.SSLContextBuilder* method), [18](#)

`with_tcp()` (*aiospamc.connections.ConnectionManagerBuilder* method), [15](#)

`with_unix_socket()` (*aiospamc.connections.ConnectionManagerBuilder* method), [15](#)