aiospamc Documentation

Release 0.9.0

Michael Caley

May 03, 2023

CONTENTS

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

CHAPTER

ONE

CONTENTS

1.1 User Guide

1.1.1 Requirements

- Python 3.7 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
poetry install
```

Note: aiospamc's build system uses Poetry which you can get from here: https://poetry.eustace.io/

1.1.3 How to use aiospamc

aiospamc provides top-level functions for basic functionality a lot like the *requests* library.

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>'
```

(continues on next page)

(continued from previous page)

```
                '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')

async def check_for_spam(message):
    response = await aiospamc.check(message, host='localhost')
    return response

loop = asyncio.get_event_loop()

response = loop.run_until_complete(check_for_spam(example_message))
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.

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

Example:

```
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:

```
loguru.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.

1.2 API Reference

1.2.1 aiospamc package

Submodules

aiospamc.connections module

ConnectionManager classes for TCP and Unix sockets.

class aiospamc.connections.**Timeout**(*total: float = 600*, *connection: Optional[float] = None*, *response: Optional[float] = None*)

Bases: object

Container object for defining timeouts.

__init__(*total: float = 600*, *connection: Optional[float] = None*, *response: Optional[float] = 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.**ConnectionManager**(*connection_string: str*, *timeout: Optional[Timeout] = None*)

Bases: object

Stores connection parameters and creates connections.

__init__(*connection_string: str*, *timeout: Optional[Timeout] = 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.**TcpConnectionManager**(*host: str*, *port: int*, *ssl_context: Optional[SSLContext] = None*, *timeout: Optional[*Timeout*] = None*)

Bases: *ConnectionManager*

Connection manager for TCP connections.

__init__(*host: str*, *port: int*, *ssl_context: Optional[SSLContext] = None*, *timeout: Optional[*Timeout*] = 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.**UnixConnectionManager**(*path: str*, *timeout: Optional[*Timeout*] = None*)

Bases: *ConnectionManager*

Connection manager for Unix pipes.

__init__(*path: str*, *timeout: Optional[*Timeout*] = 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.

aiospamc.connections.**new_ssl_context**(*verify: Optional[Any]*) → Optional[SSLContext]

Creates an SSL context based on the supplied parameter.

Parameters
verify – Use SSL for the connection. If True, will use root certificates. If False, will not verify the certificate. If a string to a path or a Path object, the connection will use the certificates found there.

aiospamc.connections.**new_connection_manager**(*host: Optional[str] = None*, *port: Optional[int] = None*, *socket_path: Optional[str] = None*, *timeout: Optional[*Timeout*] = None*, *context: Optional[SSLContext] = None*) → *ConnectionManager*

Create a new connection manager.

Parameters

- **host** – TCP hostname.
- **port** – TCP port number.
- **socket_path** – Unix socket path.
- **timeout** – Timeout configuration.
- **context** – SSL context configuration.

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.**ResponseException**(*code: int*, *message: str*)

Bases: Exception

Base class for exceptions raised from a response.

__init__(*code: int*, *message: str*) → None

ResponseException constructor.

Parameters

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

exception aiospamc.exceptions.**UsageException**(*message: str*)

Bases: *ResponseException*

Command line usage error.

__init__(*message: str*) → None

UsageException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**DataErrorException**(*message: str*)

Bases: *ResponseException*

Data format error.

__init__(*message: str*) → None

DataErrorException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**NoInputException**(*message: str*)

Bases: *ResponseException*

Cannot open input.

__init__(*message: str*) → None

NoInputException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**NoUserException**(*message: str*)

Bases: *ResponseException*

Addressee unknown.

__init__(*message: str*) → None

NoUserException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**NoHostException**(*message: str*)

Bases: *ResponseException*

Hostname unknown.

__init__(*message: str*) → None

NoHostException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**UnavailableException**(*message: str*)

Bases: *ResponseException*

Service unavailable.

__init__(*message: str*) → None

UnavailableException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**InternalSoftwareException**(*message: str*)

Bases: *ResponseException*

Internal software error.

__init__(*message: str*) → None

InternalSoftwareException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**OSErrorException**(*message: str*)

Bases: *ResponseException*

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

__init__(*message: str*) → None

OSErrorException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**OSFileException**(*message: str*)

Bases: *ResponseException*

Critical operating system file missing.

__init__(*message: str*) → None

OSFileException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**CantCreateException**(*message: str*)

Bases: *ResponseException*

Can't create (user) output file.

__init__(*message: str*) → None

CantCreateException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**IOErrorException**(*message: str*)

Bases: *ResponseException*

Input/output error.

__init__(*message: str*) → None

IOErrorException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**TemporaryFailureException**(*message: str*)

Bases: *ResponseException*

Temporary failure, user is invited to try again.

__init__(*message: str*) → None

TemporaryFailureException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**ProtocolException**(*message: str*)

Bases: *ResponseException*

Remote error in protocol.

__init__(*message: str*) → None

ProtocolException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**NoPermissionException**(*message: str*)

Bases: *ResponseException*

Permission denied.

__init__(*message: str*) → None

NoPermissionException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**ConfigException**(*message: str*)

Bases: *ResponseException*

Configuration error.

__init__(*message: str*) → None

ConfigException constructor.

Parameters
message – Message response.

exception aiospamc.exceptions.**TimeoutException**

Bases: Exception

General timeout exception.

exception aiospamc.exceptions.**ServerTimeoutException**(*message: str*)

Bases: *ResponseException*, *TimeoutException*

Timeout exception from the server.

__init__(*message: str*) → None

ServerTimeoutException constructor.

Parameters
message – Message response.

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*) → 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.

async aiospamc.frontend.**check**(*message: Union[bytes, SupportsBytes]*, ***, *host: str = 'localhost'*, *port: int = 783*, *socket_path: Optional[str] = None*, *timeout: Optional[Timeout] = None*, *verify: Optional[Any] = None*, *user: Optional[str] = None*, *compress: bool = False*, ***kwargs*) → *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.

- **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: Union[bytes, SupportsBytes]*, ***, *host: str = 'localhost'*, *port: int = 783*, *socket_path: Optional[str] = None*, *timeout: Optional[Timeout] = None*, *verify: Optional[Any] = None*, *user: Optional[str] = None*, *compress: bool = False*, ***kwargs*) → *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.

- **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: Optional[str] = None*, *timeout: Optional[Timeout] = None*, *verify: Optional[Any] = None*, ***kwargs*) → *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.

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: Union[bytes, SupportsBytes]*, ***, *host: str = 'localhost'*, *port: int = 783*, *socket_path: Optional[str] = None*, *timeout: Optional[Timeout] = None*, *verify: Optional[Any] = None*, *user: Optional[str] = None*, *compress: bool = False*, ***kwargs*) → *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.
- **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: Union[bytes, SupportsBytes]*, ***, *host: str = 'localhost'*, *port: int = 783*, *socket_path: Optional[str] = None*, *timeout: Optional[Timeout] = None*, *verify: Optional[Any] = None*, *user: Optional[str] = None*, *compress: bool = False*, ***kwargs*) → *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.
- **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: Union[bytes, SupportsBytes]*, ***, *host: str = 'localhost'*, *port: int = 783*, *socket_path: Optional[str] = None*, *timeout: Optional[Timeout] = None*, *verify: Optional[Any] = None*, *user: Optional[str] = None*, *compress: bool = False*, ***kwargs*) → *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.
- **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: Union[bytes, SupportsBytes]*, ***, *host: str = 'localhost'*, *port: int = 783*, *socket_path: Optional[str] = None*, *timeout: Optional[Timeout] = None*, *verify: Optional[Any] = None*, *user: Optional[str] = None*, *compress: bool = False*, ***kwargs*) → *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.
- **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: Union[bytes, SupportsBytes]*, *message_class: Union[str,* MessageClassOption*]*, *remove_action: Optional[Union[str,* ActionOption*]] = None*, *set_action: Optional[Union[str,* ActionOption*]] = None*, ***, *host: str = 'localhost'*, *port: int = 783*, *socket_path: Optional[str] = None*, *timeout: Optional[*Timeout*] = None*, *verify: Optional[Any] = None*, *user: Optional[str] = None*, *compress: bool = False*, ***kwargs*) → *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.
- **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.

class aiospamc.header_values.**HeaderValue**

Bases: `object`

Base class for header values.

to_dict() → Dict[str, Any]

Converts the value to a dictionary.

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

Bases: *HeaderValue*

Header with bytes value.

Parameters
value – Value of the header.

value: bytes

__init__(*value: bytes*) → None

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

Bases: *HeaderValue*

Generic header value.

value: str

encoding: str = 'utf8'

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

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

Bases: *HeaderValue*

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

algorithm: str = 'zlib'

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

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

Bases: *HeaderValue*

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

length: int = 0

__init__(*length: int = 0*) → None

class aiospamc.header_values.**MessageClassOption**(*value*)

Bases: Enum

Option to be used for the MessageClass header.

spam = 'spam'

ham = 'ham'

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

Bases: *HeaderValue*

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

value: *MessageClassOption* = 'ham'

to_dict() → Dict[str, Any]

Converts the value to a dictionary.

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

class aiospamc.header_values.**ActionOption**(*local: Optional[bool]*, *remote: Optional[bool]*)

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: Optional[bool]

remote: Optional[bool]

__init__(*local: Optional[bool]*, *remote: Optional[bool]*) → None

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

Bases: *HeaderValue*

Base class for headers that implement "local" and "remote" rules.

action: *ActionOption*

__init__(*action:* ActionOption) → None

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

Bases: *HeaderValue*

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

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

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

Bases: *HeaderValue*

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

name: str = 'docs'

__init__(*name: str = 'docs'*) → None

aiospamc.incremental_parser module

Module for the parsing functions and objects.

class aiospamc.incremental_parser.**States**(*value*)

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_line* 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_line* 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	Delim-iter	Left-over	Action
No	Yes	Delim-iter	Headers done, empty body. Clear buffer and transition to *States.Body*.
No	Yes	N/A	Headers done. Transition to *States.Body*.
Yes	Yes	N/A	Parse header. Record in `result` class attribute.
No	No	No	Message was a status line only. Transition to *States.Body*.

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*.

aiospamc.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.

aiospamc.incremental_parser.**parse_response_status**(*stream: bytes*) → Dict[str, Union[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.

aiospamc.incremental_parser.**parse_message_class_value**(*stream: Union[str,* MessageClassOption*]*) → *MessageClassValue*

Parses the *Message-class* header value.

Parameters
stream – String or *MessageClassOption* instance.

Returns
A *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: Union[str, int]*) → *ContentLengthValue*

Parses the *Content-length* header value.

Parameters
stream – String or integer value of the header.

Returns
A *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 *CompressValue* instance.

aiospamc.incremental_parser.**parse_set_remove_value**(*stream: Union[*ActionOption, *str]*) → *SetOrRemoveValue*

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

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

Returns
A *SetOrRemoveValue* instance.

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

Parses the values for the *Spam* header.

Parameters
stream – String to parse.

Returns
An *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 *UserValue* instance.

aiospamc.incremental_parser.**parse_header_value**(*header: str*, *value: Union[str, bytes]*) → *HeaderValue*

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 *HeaderValue* instance from the parsing function.

aiospamc.incremental_parser.**parse_header**(*stream: bytes*) → Tuple[str, *HeaderValue*]

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__()

RequestParse 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: Optional[Dict[str, HeaderValue]] = None*, *body: Union[bytes, SupportsBytes] = b''*, ***_*)

Bases: object

SPAMC request object.

__init__(*verb: str*, *version: str = '1.5'*, *headers: Optional[Dict[str, HeaderValue]] = None*, *body: Union[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_dict() → Dict[str, Any]

Converts the request to a dictionary.

aiospamc.responses module

Contains classes used for responses.

class aiospamc.responses.**Status**(*value*)

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: Union[Status, int] = 0*, *message: str = ''*, *headers: Optional[Dict[str, HeaderValue]] = None*, *body: bytes = b''*, ***_*)

Bases: object

Class to encapsulate response.

__init__(*version: str = '1.5'*, *status_code: Union[Status, int] = 0*, *message: str = ''*, *headers: Optional[Dict[str, HeaderValue]] = 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: Union[*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_dict() → Dict[str, Any]

Converts the response to a dictionary.

Module contents

aiospamc package.

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

1.3 SPAMC/SPAMD Protocol As Implemented by SpamAssassin

1.3.1 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:

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

```
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.

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
Content-Type: multipart/mixed; boundary="----------=_5875044E.D4EFFFD7"

This is a multi-part message in MIME format.

------------=_5875044E.D4EFFFD7
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
```

(continues on next page)

(continued from previous page)

```
------------=_5875044E.D4EFFFD7
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
    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.D4EFFFD7--
```

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
@@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
```

1.3.2 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 seprated 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 seprated by a comma.

Message-class

Classifies the message contained in the body. Contains a string containing either `local`, `remote` or both seprated 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 seprated 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 seprated 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.

1.3.3 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 repository[2].

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.

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

EX_NOINPUT

Code: 66

Definition: Cannot open input.

EX_NOUSER

Code: 67

Definition: Addressee unknown.

EX_NOHOST

Code: 68

Definition: Hostname unknown.

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.

1.3.4 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.

1.3.5 References

1.4 Release Notes

1.4.1 v0.9.0

New Features

- Add support for Python 3.11. #355

Bug Fixes

- Resolved *ValueError* exception when importing ActionOption. #357

1.4.2 v0.8.1

Security Issues

- Updated loguru dependency to 0.6.0 resolve vulnerability CVE-2022-0329.

1.4.3 v0.8.0

New Features

- Added support and tests for Python 3.10.
- Added warning for using SSL and compression hangs the connection.
- Made repr methods of Request and Response objects more descriptive.
- Using the loguru library <https://github.com/Delgan/loguru> instead of logging from the standard library.

Deprecation Notes

- Support for Python 3.6.

Bug Fixes

- Now correctly parses response messages that had spaces.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

U

V