aiospamc Documentation

Release 0.6.1

Michael Caley

Aug 20, 2020

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 3.5 or later is required to use the new async/await syntax provided by the asyncio library.
- 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>'
                   'Subject: Saying Hello'
```

(continues on next page)

(continued from previous page)

```
                '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(
    'Is the message spam? {}'.format(response.headers['Spam'].value),
    'The score and threshold is {} / {}'.format(
        response.headers['Spam'].score,
        response.headers['Spam'].threshold)
)
```

All the frontend functions instantiate the *aiospamc.client.Client* class behind the scenes. Additional keywords arguments can be found in the class constructor documentation.

Connect using SSL

Each frontend function and *aiospamc.client.Client* 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.

Making your own requests

If a request that isn't built into aiospamc is needed a new request can be created and sent.

A new request can be made by instantiating the *aiospamc.requests.Request* class. The `aiospamc.requests.Request.verb` defines the method/verb of the request.

The *aiospamc.requests.Request* class provides a headers attribute that has a dictionary-like interface. Defined headers can be referenced in the *Headers* section in *SPAMC/SPAMD Protocol As Implemented by SpamAssassin*.

Once a request is composed, the *aiospamc.client.Client* class can be instantiated and the request can be sent through the *aiospamc.client.Client.send()* method. The method will automatically add the *User* and *Compress* headers if required.

For example:

```
import asyncio

import aiospamc
from aiospamc import Client
from aiospamc.exceptions import ResponseException
```

(continues on next page)

(continued from previous page)

```
from aiospamc.requests import Request

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')

loop = asyncio.get_event_loop()
client = aiospamc.Client(host='localhost')

async def is_spam(message):
    request = Request(verb='CHECK', body=message.encode())
    try:
        response = await client.send(request)
        return response.get_header('Spam').value
    except aiospamc.ResponseException:
        raise

spam_result = loop.run_until_complete(is_spam(example_message))
print('Example message is spam:', spam_result)
```

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

Subpackages

aiospamc.connections package

Submodules

aiospamc.connections.tcp_connection module

TCP socket connection and manager.

class aiospamc.connections.tcp_connection.**TcpConnectionManager**(*host: str*, *port: int*, *ssl: ssl.SSLContext = None*, *loop: asyncio.events.AbstractEventLoop = None*)

Bases: *aiospamc.connections.ConnectionManager*

Creates new connections based on host and port provided.

__init__(*host: str*, *port: int*, *ssl: ssl.SSLContext = None*, *loop: asyncio.events.AbstractEventLoop = None*) → None
Constructor for TcpConnectionManager.

Parameters

- **host** – Hostname or IP address of server.
- **port** – Port number
- **ssl** – SSL/TLS context.
- **loop** – The asyncio event loop.

new_connection() → *aiospamc.connections.tcp_connection.TcpConnection*
Creates a new TCP connection.

Raises ***AIOSpamcConnectionFailed*** –

class aiospamc.connections.tcp_connection.**TcpConnection**(*host: str*, *port: int*, *ssl: ssl.SSLContext = None*, *loop: asyncio.events.AbstractEventLoop = None*)

Bases: *aiospamc.connections.Connection*

Manages a TCP connection.

__init__(*host: str*, *port: int*, *ssl: ssl.SSLContext = None*, *loop: asyncio.events.AbstractEventLoop = None*)
Constructor for TcpConnection.

Parameters

- **host** – Hostname or IP address of server.
- **port** – Port number
- **ssl** – SSL/TLS context.

async open() → Tuple[asyncio.streams.StreamReader, asyncio.streams.StreamWriter]
Opens a connection.

Raises ***AIOSpamcConnectionFailed*** –

property connection_string
String representation of the connection.

aiospamc.connections.unix_connection module

Unix domain socket connection and manager.

class aiospamc.connections.unix_connection.**UnixConnectionManager**(*path: str*, *loop: asyncio.events.AbstractEventLoop = None*)

Bases: *aiospamc.connections.ConnectionManager*

Creates new connections based on Unix domain socket path provided.

__init__(*path: str*, *loop: asyncio.events.AbstractEventLoop = None*) → None
Constructor for UnixConnectionManager.

Parameters

- **path** – Path of the socket.
- **loop** – The asyncio event loop.

new_connection() → *aiospamc.connections.unix_connection.UnixConnection*
Creates a new Unix domain socket connection.

Raises ***AIOSpamcConnectionFailed*** –

class aiospamc.connections.unix_connection.**UnixConnection**(*path: str*, *loop: asyncio.events.AbstractEventLoop = None*)
Bases: *aiospamc.connections.Connection*

Manages a Unix domain socket connection.

__init__(*path: str*, *loop: asyncio.events.AbstractEventLoop = None*) → None
Constructor for UnixConnection.

Parameters

- **path** – Path of the socket.
- **loop** – The asyncio event loop.

async open() → Tuple[asyncio.streams.StreamReader, asyncio.streams.StreamWriter]
Opens a connection.

Raises ***AIOSpamcConnectionFailed*** –

property connection_string
String representation of the connection.

Module contents

Connection and ConnectionManager base classes.

class aiospamc.connections.**Connection**(*loop: asyncio.events.AbstractEventLoop = None*)
Bases: object

Base class for connection objects.

__init__(*loop: asyncio.events.AbstractEventLoop = None*)
Connection constructor.

Parameters **loop** – The asyncio event loop.

async open() → Tuple[asyncio.streams.StreamReader, asyncio.streams.StreamWriter]
Connect to a service.

Raises ***AIOSpamcConnectionFailed*** –

property connection_string
String representation of the connection.

close() → None
Closes the connection.

async send(*data: Union[bytes, SupportsBytes]*) → None
Sends data through the connection.

async receive() → bytes
Receives data from the connection.

class aiospamc.connections.**ConnectionManager**(*loop: asyncio.events.AbstractEventLoop = None*)
Bases: object

Stores connection parameters and creates connections.

__init__(*loop: asyncio.events.AbstractEventLoop = None*)
Initialize self. See help(type(self)) for accurate signature.

new_connection() → *aiospamc.connections.Connection*
Creates a connection object.

Submodules

aiospamc.client module

Contains the Client class that is used to interact with SPAMD.

class aiospamc.client.**Client**(*socket_path: str = None, host: str = 'localhost', port: int = 783, user: str = None, compress: bool = False, verify: Union[bool, str, pathlib.Path] = None, loop: asyncio.events.AbstractEventLoop = None*)
Bases: object

Client object for interacting with SPAMD.

__init__(*socket_path: str = None, host: str = 'localhost', port: int = 783, user: str = None, compress: bool = False, verify: Union[bool, str, pathlib.Path] = None, loop: asyncio.events.AbstractEventLoop = None*) → None
Client constructor.

Parameters

- **socket_path** – The path to the Unix socket for the SPAMD service.
- **host** – Hostname or IP address of the SPAMD service, defaults to localhost.
- **port** – Port number for the SPAMD service, defaults to 783.
- **user** – Name of the user that SPAMD will run the checks under.
- **compress** – If true, the request body will be compressed.
- **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.
- **loop** – The asyncio event loop.

Raises **ValueError** – Raised if the constructor can't tell if it's using a TCP or a Unix domain socket connection.

static new_ssl_context(*value: Union[bool, str, pathlib.Path]*) → ssl.SSLContext
Creates an SSL context based on the supplied parameter.

Parameters **value** – 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.

async send(*request:* aiospamc.requests.Request) → *aiospamc.responses.Response*
Sends a request to the SPAMD service.

If the SPAMD service gives a temporary failure response, then its retried.

Parameters **request** – Request object to send.

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.
- ***TimeoutException*** – Timeout during connection.

async check(*message: Union[bytes, SupportsBytes]*) → *aiospamc.responses.Response*
Request the SPAMD service to check a message.

Parameters **message** – A byte string containing the contents of the message to be scanned.

SPAMD will perform a scan on the included message. SPAMD expects an RFC 822 or RFC 2822 formatted email.

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.
- **TimeoutException** – Timeout during connection.

async headers(*message: Union[bytes, SupportsBytes]*) → *aiospamc.responses.Response*

Request the SPAMD service to check a message with a HEADERS request.

Parameters **message** – A byte string containing the contents of the message to be scanned.

SPAMD will perform a scan on the included message. SPAMD expects an RFC 822 or RFC 2822 formatted email.

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.
- **TimeoutException** – Timeout during connection.

async ping() → *aiospamc.responses.Response*

Sends a ping request to the SPAMD service and will receive a response if the service is alive.

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.
- **TimeoutException** – Timeout during connection.

async process(*message: Union[bytes, SupportsBytes]*) → *aiospamc.responses.Response*

Process the message and return a modified copy of the message.

Parameters **message** – A byte string containing the contents of the message to be scanned.

SPAMD will perform a scan on the included message. SPAMD expects an RFC 822 or RFC 2822 formatted email.

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.
- **TimeoutException** – Timeout during connection.

async report (*message: Union[bytes, SupportsBytes]*) → *aiospamc.responses.Response*

Check if message is spam and return report.

Parameters **message** – A byte string containing the contents of the message to be scanned.

SPAMD will perform a scan on the included message. SPAMD expects an RFC 822 or RFC 2822 formatted email.

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.
- **TimeoutException** – Timeout during connection.

async report_if_spam(*message: Union[bytes, SupportsBytes]*) → *aiospamc.responses.Response*

Check if a message is spam and return a report if the message is spam.

Parameters **message** – A byte string containing the contents of the message to be scanned.

SPAMD will perform a scan on the included message. SPAMD expects an RFC 822 or RFC 2822 formatted email.

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.
- **TimeoutException** – Timeout during connection.

async symbols(*message: Union[bytes, SupportsBytes]*) → *aiospamc.responses.Response*

Check if the message is spam and return a list of symbols that were hit.

Parameters **message** – A byte string containing the contents of the message to be scanned.

SPAMD will perform a scan on the included message. SPAMD expects an RFC 822 or RFC 2822 formatted email.

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.
- ***TimeoutException*** – Timeout during connection.

async tell(*message: Union[bytes, SupportsBytes]*, *message_class: Union[str, aiospamc.options.MessageClassOption]*, *remove_action: Union[str, aiospamc.options.ActionOption]* = *None*, *set_action: Union[str, aiospamc.options.ActionOption]* = *None*)

Instruct the SPAMD service to to mark the message.

Parameters

- **message** – A byte string containing the contents of the message to be scanned.

  SPAMD will perform a scan on the included message. SPAMD expects an RFC 822 or RFC 2822 formatted email.

- **message_class** – How to classify the message, either "ham" or "spam".
- **remove_action** – Remove message class for message in database.
- **set_action** – Set message class for message in database. Either *ham* or *spam*.

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.
- ***TimeoutException*** – Timeout during connection.

aiospamc.common module

Common classes for the project.

class aiospamc.common.**SpamcBody**
Bases: object

Provides a descriptor for a bytes-like object.

class aiospamc.common.**SpamcHeaders**(*, *headers: Mapping[str, Union[*aiospamc.header_values.HeaderValue*, str, Any]] = None*)
Bases: object

Provides a dictionary-like interface for headers.

__init__(*, *headers: Mapping[str, Union[*aiospamc.header_values.HeaderValue*, str, Any]] = None*) → None
Initialize self. See help(type(self)) for accurate signature.

keys() → KeysView[str]

items() → ItemsView[str, *aiospamc.header_values.HeaderValue*]

values() → ValuesView[*aiospamc.header_values.HeaderValue*]

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: *aiospamc.exceptions.ClientException*

Request is not in the expected format.

exception aiospamc.exceptions.**BadResponse**
Bases: *aiospamc.exceptions.ClientException*

Response is not in the expected format.

exception aiospamc.exceptions.**AIOSpamcConnectionException**
Bases: Exception

Base class for exceptions from the connection.

exception aiospamc.exceptions.**AIOSpamcConnectionFailed**
Bases: *aiospamc.exceptions.AIOSpamcConnectionException*

Connection failed.

exception aiospamc.exceptions.**ResponseException**
Bases: Exception

Base class for exceptions raised from a response.

exception aiospamc.exceptions.**UsageException**
Bases: *aiospamc.exceptions.ResponseException*

Command line usage error.

code = 64

exception aiospamc.exceptions.**DataErrorException**
Bases: *aiospamc.exceptions.ResponseException*

Data format error.

code = 65

exception aiospamc.exceptions.**NoInputException**
Bases: *aiospamc.exceptions.ResponseException*

Cannot open input.

code = 66

exception aiospamc.exceptions.**NoUserException**
Bases: *aiospamc.exceptions.ResponseException*

Addressee unknown.

code = 67

exception aiospamc.exceptions.**NoHostException**
Bases: *aiospamc.exceptions.ResponseException*

Hostname unknown.

code = 68

exception aiospamc.exceptions.**UnavailableException**
Bases: *aiospamc.exceptions.ResponseException*

Service unavailable.

code = 69

exception aiospamc.exceptions.**InternalSoftwareException**
Bases: *aiospamc.exceptions.ResponseException*

Internal software error.

code = 70

exception aiospamc.exceptions.**OSErrorException**
Bases: *aiospamc.exceptions.ResponseException*

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

code = 71

exception aiospamc.exceptions.**OSFileException**
Bases: *aiospamc.exceptions.ResponseException*

Critical operating system file missing.

code = 72

exception aiospamc.exceptions.**CantCreateException**
Bases: *aiospamc.exceptions.ResponseException*

Can't create (user) output file.

code = 73

exception aiospamc.exceptions.**IOErrorException**
Bases: *aiospamc.exceptions.ResponseException*

Input/output error.

code = 74

exception aiospamc.exceptions.**TemporaryFailureException**
Bases: *aiospamc.exceptions.ResponseException*

Temporary failure, user is invited to try again.

code = 75

exception aiospamc.exceptions.**ProtocolException**
Bases: *aiospamc.exceptions.ResponseException*

Remote error in protocol.

code = 76

exception aiospamc.exceptions.**NoPermissionException**
Bases: *aiospamc.exceptions.ResponseException*

Permission denied.

code = 77

exception aiospamc.exceptions.**ConfigException**
Bases: *aiospamc.exceptions.ResponseException*

Configuration error.

code = 78

exception aiospamc.exceptions.**TimeoutException**
Bases: *aiospamc.exceptions.ResponseException*

Read timeout.

code = 79

aiospamc.header_values module

Collection of request and response header value objects.

class aiospamc.header_values.**HeaderValue**
Bases: object

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

Generic header value.

__init__(*value: str*, *encoding='utf8'*) → None
Initialize self. See help(type(self)) for accurate signature.

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

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

__init__(*algorithm='zlib'*) → None
Constructor

Parameters **algorithm** – Compression algorithm to use. Currently on zlib is supported.

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

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

__init__(*length: int = 0*) → None
ContentLength constructor.

Parameters **length** – Length of the body.

class aiospamc.header_values.**MessageClassValue**(*value:* aiospamc.options.MessageClassOption *= None*)
Bases: *aiospamc.header_values.HeaderValue*

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

__init__(*value:* aiospamc.options.MessageClassOption *= None*) → None
MessageClass constructor.

Parameters **value** – Specifies the classification of the message.

class aiospamc.header_values.**SetOrRemoveValue**(*action:* aiospamc.options.ActionOption *= None*)
Bases: *aiospamc.header_values.HeaderValue*

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

__init__(*action:* aiospamc.options.ActionOption *= None*) → None
_SetRemoveBase constructor.

Parameters **action** – Actions to be done on local or remote.

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

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

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

Parameters

- **value** – True if the message is spam, False if not.
- **score** – Score of the message after being scanned.
- **threshold** – Threshold of which the message would have been marked as spam.

class aiospamc.header_values.**UserValue**(*name: str = None*)
Bases: *aiospamc.header_values.HeaderValue*

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

__init__(*name: str = None*) → None
User constructor.

Parameters **name** – Name of the user account.

aiospamc.incremental_parser module

Module for the parsing functions and objects.

exception aiospamc.incremental_parser.**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.incremental_parser.**NotEnoughDataError**(*message=None*)
Bases: *aiospamc.incremental_parser.ParseError*

Expected more data than what the protocol content specified.

exception aiospamc.incremental_parser.**TooMuchDataError**(*message=None*)
Bases: *aiospamc.incremental_parser.ParseError*

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

class aiospamc.incremental_parser.**States**(*value*)
Bases: enum.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: aiospamc.incremental_parser.States = <States.Status: 1>*)
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: aiospamc.incremental_parser.States = <States.Status: 1>*) → 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

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	Delimiter	Left-over	Action
No	Yes	Delimiter	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, str]
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, aiospamc.options.MessageClassOption]*) → *aiospamc.header_values.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]*) → *aiospamc.header_values.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*) → *aiospamc.header_values.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[aiospamc.options.ActionOption, str]*) → *aiospamc.header_values.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*) → *aiospamc.header_values.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*) → *aiospamc.header_values.UserValue*

aiospamc.incremental_parser.**parse_generic_header_value**(*stream: str*) → *aiospamc.header_values.GenericHeaderValue*

Parses any user-defined or currently unknown header values.

Parameters **stream** – String to parse.

Returns A *GenericHeaderValue* instance.

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

Mapping for header names to their parsing functions.

class aiospamc.incremental_parser.**RequestParser**

Bases: *aiospamc.incremental_parser.Parser*

Sub-class of the parser for requests.

__init__()
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.

class aiospamc.incremental_parser.**ResponseParser**
Bases: *aiospamc.incremental_parser.Parser*

Sub-class of the parser for responses.

__init__()
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.

aiospamc.options module

Data structures used for function parameters.

class aiospamc.options.**MessageClassOption**(*value*)
Bases: enum.IntEnum

Option to be used for the MessageClass header.

spam = 1

ham = 2

class aiospamc.options.**ActionOption**(*local*, *remote*)
Bases: tuple

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

local [bool] An action will be performed on the SPAMD service's local database.

remote [bool] An action will be performed on the SPAMD service's remote database.

property local
Alias for field number 0

property remote
Alias for field number 1

aiospamc.requests module

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

class aiospamc.requests.**Request**(*verb: str = None, version: str = '1.5', headers: Mapping[str, Union[str, aiospamc.header_values.HeaderValue]] = None, body: Union[bytes, SupportsBytes] = b''*, **_)

Bases: object

SPAMC request object.

__init__(*verb: str = None, version: str = '1.5', headers: Mapping[str, Union[str, aiospamc.header_values.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.

body

Provides a descriptor for a bytes-like object.

aiospamc.responses module

Contains classes used for responses.

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

Bases: enum.IntEnum

Enumeration of status codes that the SPAMD will accompany with a response.

Reference: https://svn.apache.org/repos/asf/spamassassin/trunk/spamd/spamd.raw Look for the %resphash variable.

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[aiospamc.responses.Status, int] = 0, message: str = '', headers: Mapping[str, aiospamc.header_values.HeaderValue] = None, body: bytes = b''*, **_)

Bases: object

Class to encapsulate response.

__init__(*version: str = '1.5', status_code: Union[aiospamc.responses.Status, int] = 0, message: str = '', headers: Mapping[str, aiospamc.header_values.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.

body

Provides a descriptor for a bytes-like object.

raise_for_status() → None

Module contents

aiospamc package.

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

async aiospamc.**check**(*message: Union[bytes, SupportsBytes], *, host: str = 'Localhost', port: int = 783, loop: asyncio.events.AbstractEventLoop = None, **kwargs*) → *aiospamc.responses.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.
- **loop** – The asyncio event loop.
- **kwargs** – Additional options to pass to the *aiospamc.client.Client* instance.

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.
- ***TimeoutException*** – Timeout during connection.

async aiospamc.**headers**(*message: Union[bytes, SupportsBytes]*, ***, *host: str = 'Localhost'*, *port: int = 783*, *loop: asyncio.events.AbstractEventLoop = None*, ***kwargs*) → *aiospamc.responses.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.
- **loop** – The asyncio event loop.
- **kwargs** – Additional options to pass to the *aiospamc.client.Client* instance.

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.
- ***TimeoutException*** – Timeout during connection.

async aiospamc.**ping**(*, *host: str = 'Localhost'*, *port: int = 783*, *loop: asyncio.events.AbstractEventLoop = None*, ***kwargs*) → *aiospamc.responses.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.
- **loop** – The asyncio event loop.
- **kwargs** – Additional options to pass to the *aiospamc.client.Client* instance.

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.
- **TimeoutException** – Timeout during connection.

async aiospamc.**process**(*message: Union[bytes, SupportsBytes]*, ***, *host: str = 'Localhost'*, *port: int = 783*, *loop: asyncio.events.AbstractEventLoop = None*, ***kwargs*) → *aiospamc.responses.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.
- **loop** – The asyncio event loop.
- **kwargs** – Additional options to pass to the *aiospamc.client.Client* instance.

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.
- **TimeoutException** – Timeout during connection.

async aiospamc.**report**(*message: Union[bytes, SupportsBytes]*, ***, *host: str = 'Localhost'*, *port: int = 783*, *loop: asyncio.events.AbstractEventLoop = None*, ***kwargs*) → *aiospamc.responses.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.
- **loop** – The asyncio event loop.
- **kwargs** – Additional options to pass to the *aiospamc.client.Client* instance.

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.
- ***TimeoutException*** – Timeout during connection.

async aiospamc.**report_if_spam**(*message: Union[bytes, SupportsBytes]*, ***, *host: str = 'Localhost'*, *port: int = 783*, *loop: asyncio.events.AbstractEventLoop = None*, ***kwargs*) → *aiospamc.responses.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.
- **loop** – The asyncio event loop.
- **kwargs** – Additional options to pass to the *aiospamc.client.Client* instance.

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.
- ***TimeoutException*** – Timeout during connection.

async aiospamc.**symbols**(*message: Union[bytes, SupportsBytes]*, ***, *host: str = 'Localhost'*, *port: int = 783*, *loop: asyncio.events.AbstractEventLoop = None*, ***kwargs*) → *aiospamc.responses.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.
- **loop** – The asyncio event loop.
- **kwargs** – Additional options to pass to the *aiospamc.client.Client* instance.

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.
- **TimeoutException** – Timeout during connection.

async aiospamc.**tell**(*message: Union[bytes, SupportsBytes]*, *message_class: Union[str, aiospamc.options.MessageClassOption]*, ***, *host: str = 'Localhost'*, *port: int = 783*, *remove_action: Union[str, aiospamc.options.ActionOption] = None*, *set_action: Union[str, aiospamc.options.ActionOption] = None*, *loop: asyncio.events.AbstractEventLoop = None*, ***kwargs*) → *aiospamc.responses.Response*

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

Parameters

- **message** – Copy of the message.
- **message_class** – How to classify the message, either "ham" or "spam".
- **host** – Hostname or IP address of the SPAMD service, defaults to localhost.
- **port** – Port number for the SPAMD service, defaults to 783.
- **remove_action** – Remove message class for message in database.
- **set_action** – Set message class for message in database. Either *ham* or *spam*.
- **loop** – The asyncio event loop.
- **kwargs** – Additional options to pass to the *aiospamc.client.Client* instance.

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.
- ***TimeoutException*** – Timeout during connection.

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:

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

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

```

(continues on next page)

(continued from previous page)

```
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



------------=_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):
```

(continues on next page)

(continued from previous page)

```
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): [...]
```

(continues on next page)

(continued from previous page)

```
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].

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

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.

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

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

G

H

I

K

S

T

U

V