21.18. `smtpd` — SMTP Server

Source code: Lib/smtpd.py

This module offers several classes to implement SMTP (email) servers.

> **See also:** The aiosmtpd package is a recommended replacement for this module. It is based on `asyncio` and provides a more straightforward API. `smtpd` should be considered deprecated.

Several server implementations are present; one is a generic do-nothing implementation, which can be overridden, while the other two offer specific mail-sending strategies.

Additionally the SMTPChannel may be extended to implement very specific interaction behaviour with SMTP clients.

The code supports **RFC 5321**, plus the **RFC 1870** SIZE and **RFC 6531** SMTPUTF8 extensions.

21.18.1. SMTPServer Objects

class `smtpd.`**`SMTPServer`**(*localaddr*, *remoteaddr*, *data_size_limit=33554432*, *map=None*, *enable_SMTPUTF8=False*, *decode_data=False*)

Create a new `SMTPServer` object, which binds to local address *localaddr*. It will treat *remoteaddr* as an upstream SMTP relayer. Both *localaddr* and *remoteaddr* should be a (host, port) tuple. The object inherits from `asyncore.dispatcher`, and so will insert itself into `asyncore`'s event loop on instantiation.

data_size_limit specifies the maximum number of bytes that will be accepted in a `DATA` command. A value of `None` or `0` means no limit.

map is the socket map to use for connections (an initially empty dictionary is a suitable value). If not specified the `asyncore` global socket map is used.

enable_SMTPUTF8 determines whether the `SMTPUTF8` extension (as defined in **RFC 6531**) should be enabled. The default is `False`. When `True`, `SMTPUTF8` is accepted as a parameter to the `MAIL` command and when present is passed to `process_message()` in the `kwargs['mail_options']` list. *decode_data* and *enable_SMTPUTF8* cannot be set to `True` at the same time.

decode_data specifies whether the data portion of the SMTP transaction should be decoded using UTF-8. When *decode_data* is `False` (the default), the server advertises the `8BITMIME` extension (**RFC 6152**), accepts the `BODY=8BITMIME` parameter to the `MAIL` command, and when present passes it to `process_message()` in the `kwargs['mail_options']` list. *decode_data* and *enable_SMTPUTF8* cannot be set to `True` at the same time.

`process_message`(*peer*, *mailfrom*, *rcpttos*, *data*, ***kwargs*)

Raise a `NotImplementedError` exception. Override this in subclasses to do something useful with this message. Whatever was passed in the constructor as *remoteaddr* will be available as the `_remoteaddr` attribute. *peer* is the remote host's address, *mailfrom* is the envelope originator, *rcpttos* are the envelope recipients and *data* is a string containing the contents of the e-mail (which should be in **RFC 5321** format).

If the *decode_data* constructor keyword is set to `True`, the *data* argument will be a unicode string. If it is set to `False`, it will be a bytes object.

kwargs is a dictionary containing additional information. It is empty if `decode_data=True` was given as an init argument, otherwise it contains the following keys:

mail_options:
: a list of all received parameters to the `MAIL` command (the elements are uppercase strings; example: `['BODY=8BITMIME', 'SMTPUTF8']`).

rcpt_options:
: same as *mail_options* but for the `RCPT` command. Currently no `RCPT TO` options are supported, so for now this will always be an empty list.

Implementations of `process_message` should use the `**kwargs` signature to accept arbitrary keyword arguments, since future feature enhancements may add keys to the kwargs dictionary.

Return `None` to request a normal `250 Ok` response; otherwise return the desired response string in **RFC 5321** format.

`channel_class`

Override this in subclasses to use a custom `SMTPChannel` for managing SMTP clients.

New in version 3.4: The *map* constructor argument.

Changed in version 3.5: *localaddr* and *remoteaddr* may now contain IPv6 addresses.

New in version 3.5: The *decode_data* and *enable_SMTPUTF8* constructor parameters, and the *kwargs* parameter to `process_message()` when *decode_data* is `False`.

Changed in version 3.6: *decode_data* is now `False` by default.

21.18.2. DebuggingServer Objects

class `smtpd.`**`DebuggingServer`**(*localaddr*, *remoteaddr*)

Create a new debugging server. Arguments are as per `SMTPServer`. Messages will be discarded, and printed on stdout.

21.18.3. PureProxy Objects

class `smtpd.`**`PureProxy`**(*localaddr*, *remoteaddr*)

Create a new pure proxy server. Arguments are as per `SMTPServer`. Everything will be relayed to *remoteaddr*. Note that running this has a good chance to make you into an open relay, so please be careful.

21.18.4. MailmanProxy Objects

class `smtpd.`**`MailmanProxy`**(*localaddr*, *remoteaddr*)

Create a new pure proxy server. Arguments are as per `SMTPServer`. Everything will be relayed to *remoteaddr*, unless local mailman configurations knows about an address, in which case it will be handled via mailman. Note that running this has a good chance to make you into an open relay, so please be careful.

21.18.5. SMTPChannel Objects

class `smtpd.`**`SMTPChannel`**(*server*, *conn*, *addr*, *data_size_limit=33554432*, *map=None*, *enable_SMTPUTF8=False*, *decode_data=False*)

Create a new `SMTPChannel` object which manages the communication between the server and a single SMTP client.

conn and *addr* are as per the instance variables described below.

data_size_limit specifies the maximum number of bytes that will be accepted in a `DATA` command. A value of `None` or `0` means no limit.

enable_SMTPUTF8 determines whether the `SMTPUTF8` extension (as defined in **RFC 6531**) should be enabled. The default is `False`. *decode_data* and *enable_SMTPUTF8* cannot be set to `True` at the same time.

A dictionary can be specified in *map* to avoid using a global socket map.

decode_data specifies whether the data portion of the SMTP transaction should be decoded using UTF-8. The default is `False`. *decode_data* and *enable_SMTPUTF8* cannot be set to `True` at the same time.

To use a custom SMTPChannel implementation you need to override the `SMTPServer.channel_class` of your `SMTPServer`.

Changed in version 3.5: The *decode_data* and *enable_SMTPUTF8* parameters were added.

Changed in version 3.6: decode_data is now `False` by default.

The `SMTPChannel` has the following instance variables:

`smtp_server`

Holds the `SMTPServer` that spawned this channel.

`conn`

Holds the socket object connecting to the client.

`addr`

Holds the address of the client, the second value returned by `socket.accept`

`received_lines`

Holds a list of the line strings (decoded using UTF-8) received from the client. The lines have their `"\r\n"` line ending translated to `"\n"`.

`smtp_state`

Holds the current state of the channel. This will be either `COMMAND` initially and then `DATA` after the client sends a "DATA" line.

`seen_greeting`

Holds a string containing the greeting sent by the client in its "HELO".

`mailfrom`

Holds a string containing the address identified in the "MAIL FROM:" line from the client.

`rcpttos`

Holds a list of strings containing the addresses identified in the “RCPT TO:” lines from the client.

`received_data`

Holds a string containing all of the data sent by the client during the DATA state, up to but not including the terminating `"\r\n.\r\n"`.

`fqdn`

Holds the fully-qualified domain name of the server as returned by `socket.getfqdn()`.

`peer`

Holds the name of the client peer as returned by `conn.getpeername()` where `conn` is `conn`.

The `SMTPChannel` operates by invoking methods named `smtp_<command>` upon reception of a command line from the client. Built into the base `SMTPChannel` class are methods for handling the following commands (and responding to them appropriately):

Command	**Action taken**
HELO	Accepts the greeting from the client and stores it in `seen_greeting`. Sets server to base command mode.
EHLO	Accepts the greeting from the client and stores it in `seen_greeting`. Sets server to extended command mode.
NOOP	Takes no action.
QUIT	Closes the connection cleanly.
MAIL	Accepts the “MAIL FROM:” syntax and stores the supplied address as `mailfrom`. In extended command mode, accepts the **RFC 1870** SIZE attribute and responds appropriately based on the value of *data_size_limit*.
RCPT	Accepts the “RCPT TO:” syntax and stores the supplied addresses in the `rcpttos` list.
RSET	Resets the `mailfrom`, `rcpttos`, and `received_data`, but not the greeting.
DATA	Sets the internal state to `DATA` and stores remaining lines from the client in `received_data` until the terminator `"\r\n.\r\n"` is received.
HELP	Returns minimal information on command syntax

Command	**Action taken**
VRFY	Returns code 252 (the server doesn’t know if the address is valid)
EXPN	Reports that the command is not implemented.