# 20.12. `smtplib` — SMTP protocol client

The `smtplib` module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon. For details of SMTP and ESMTP operation, consult **RFC 821** (Simple Mail Transfer Protocol) and **RFC 1869** (SMTP Service Extensions).

*class* `smtplib.`**`SMTP`**([*host*[, *port*[, *local_hostname*[, *timeout*]]]])

A `SMTP` instance encapsulates an SMTP connection. It has methods that support a full repertoire of SMTP and ESMTP operations. If the optional host and port parameters are given, the SMTP `connect()` method is called with those parameters during initialization. An `SMTPConnectError` is raised if the specified host doesn't respond correctly. The optional *timeout* parameter specifies a timeout in seconds for blocking operations like the connection attempt (if not specified, the global default timeout setting will be used).

For normal use, you should only require the initialization/connect, `sendmail()`, and `quit()` methods. An example is included below.

*Changed in version 2.6: timeout* was added.

*class* `smtplib.`**`SMTP_SSL`**([*host*[, *port*[, *local_hostname*[, *keyfile*[, *certfile*[, *timeout*]]]]]])

A `SMTP_SSL` instance behaves exactly the same as instances of `SMTP`. `SMTP_SSL` should be used for situations where SSL is required from the beginning of the connection and using `starttls()` is not appropriate. If *host* is not specified, the local host is used. If *port* is omitted, the standard SMTP-over-SSL port (465) is used. *keyfile* and *certfile* are also optional, and can contain a PEM formatted private key and certificate chain file for the SSL connection. The optional *timeout* parameter specifies a timeout in seconds for blocking operations like the connection attempt (if not specified, the global default timeout setting will be used).

*New in version 2.6.*

*class* `smtplib.`**`LMTP`**([*host*[, *port*[, *local_hostname*]]])

The LMTP protocol, which is very similar to ESMTP, is heavily based on the standard SMTP client. It's common to use Unix sockets for LMTP, so our `connect()` method must support that as well as a regular host:port server.

To specify a Unix socket, you must use an absolute path for *host*, starting with a '/'.

Authentication is supported, using the regular SMTP mechanism. When using a Unix socket, LMTP generally don't support or require any authentication, but your mileage might vary.

*New in version 2.6.*

A nice selection of exceptions is defined as well:

*exception* `smtplib.`**`SMTPException`**

Base exception class for all exceptions raised by this module.

*exception* `smtplib.`**`SMTPServerDisconnected`**

This exception is raised when the server unexpectedly disconnects, or when an attempt is made to use the `SMTP` instance before connecting it to a server.

*exception* `smtplib.`**`SMTPResponseException`**

Base class for all exceptions that include an SMTP error code. These exceptions are generated in some instances when the SMTP server returns an error code. The error code is stored in the `smtp_code` attribute of the error, and the `smtp_error` attribute is set to the error message.

*exception* `smtplib.`**`SMTPSenderRefused`**

Sender address refused. In addition to the attributes set by on all `SMTPResponseException` exceptions, this sets 'sender' to the string that the SMTP server refused.

*exception* `smtplib.`**`SMTPRecipientsRefused`**

All recipient addresses refused. The errors for each recipient are accessible through the attribute `recipients`, which is a dictionary of exactly the same sort as `SMTP.sendmail()` returns.

*exception* `smtplib.`**`SMTPDataError`**

The SMTP server refused to accept the message data.

*exception* `smtplib.`**`SMTPConnectError`**

Error occurred during establishment of a connection with the server.

*exception* `smtplib.`**`SMTPHeloError`**

The server refused our `HELO` message.

*exception* `smtplib.`**`SMTPAuthenticationError`**

SMTP authentication went wrong. Most probably the server didn't accept the username/password combination provided.

**See also:**

**RFC 821 - Simple Mail Transfer Protocol**

Protocol definition for SMTP. This document covers the model, operating procedure, and protocol details for SMTP.

**RFC 1869 - SMTP Service Extensions**

Definition of the ESMTP extensions for SMTP. This describes a framework for extending SMTP with new commands, supporting dynamic discovery of the commands provided by the server, and defines a few additional commands.

## 20.12.1. SMTP Objects

An `SMTP` instance has the following methods:

`SMTP.`**`set_debuglevel`**(*level*)

Set the debug output level. A true value for *level* results in debug messages for connection and for all messages sent to and received from the server.

`SMTP.`**`connect`**([*host*[, *port*]])

Connect to a host on a given port. The defaults are to connect to the local host at the standard SMTP port (25). If the hostname ends with a colon (`':'`) followed by a number, that suffix will be stripped off and the number interpreted as the port number to use. This method is automatically invoked by the constructor if a host is specified during instantiation.

`SMTP.`**`docmd`**(*cmd*[, *argstring*])

Send a command *cmd* to the server. The optional argument *argstring* is simply concatenated to the command, separated by a space.

This returns a 2-tuple composed of a numeric response code and the actual response line (multiline responses are joined into one long line.)

In normal operation it should not be necessary to call this method explicitly. It is used to implement other methods and may be useful for testing private extensions.

If the connection to the server is lost while waiting for the reply, `SMTPServerDisconnected` will be raised.

SMTP.**helo**([*hostname*])

Identify yourself to the SMTP server using HELO. The hostname argument defaults to the fully qualified domain name of the local host. The message returned by the server is stored as the `helo_resp` attribute of the object.

In normal operation it should not be necessary to call this method explicitly. It will be implicitly called by the `sendmail()` when necessary.

SMTP.**ehlo**([*hostname*])

Identify yourself to an ESMTP server using EHLO. The hostname argument defaults to the fully qualified domain name of the local host. Examine the response for ESMTP option and store them for use by `has_extn()`. Also sets several informational attributes: the message returned by the server is stored as the `ehlo_resp` attribute, `does_esmtp` is set to true or false depending on whether the server supports ESMTP, and `esmtp_features` will be a dictionary containing the names of the SMTP service extensions this server supports, and their parameters (if any).

Unless you wish to use `has_extn()` before sending mail, it should not be necessary to call this method explicitly. It will be implicitly called by `sendmail()` when necessary.

SMTP.**ehlo_or_helo_if_needed**()

This method call `ehlo()` and or `helo()` if there has been no previous EHLO or HELO command this session. It tries ESMTP EHLO first.

`SMTPHeloError`

The server didn't reply properly to the HELO greeting.

*New in version 2.6.*

SMTP.**has_extn**(*name*)

Return `True` if *name* is in the set of SMTP service extensions returned by the server, `False` otherwise. Case is ignored.

SMTP.**verify**(*address*)

Check the validity of an address on this server using SMTP VRFY. Returns a tuple consisting of code 250 and a full **RFC 822** address (including human name) if the user address is valid. Otherwise returns an SMTP error code of 400 or greater and an error string.

> **Note:** Many sites disable SMTP VRFY in order to foil spammers.

SMTP.**login**(*user*, *password*)

Log in on an SMTP server that requires authentication. The arguments are the username and the password to authenticate with. If there has been no previous EHLO or HELO command this session, this method tries ESMTP EHLO first. This method will return normally if the authentication was successful, or may raise the following exceptions:

**SMTPHeloError**

The server didn't reply properly to the HELO greeting.

**SMTPAuthenticationError**

The server didn't accept the username/password combination.

**SMTPException**

No suitable authentication method was found.

SMTP.**starttls**([*keyfile*[, *certfile*]])

Put the SMTP connection in TLS (Transport Layer Security) mode. All SMTP commands that follow will be encrypted. You should then call `ehlo()` again.

If *keyfile* and *certfile* are provided, these are passed to the `socket` module's `ssl()` function.

If there has been no previous EHLO or HELO command this session, this method tries ESMTP EHLO first.

*Changed in version 2.6.*

**SMTPHeloError**

The server didn't reply properly to the HELO greeting.

**SMTPException**

The server does not support the STARTTLS extension.

*Changed in version 2.6.*

`RuntimeError`

SSL/TLS support is not available to your Python interpreter.

`SMTP.`**`sendmail`**(*from_addr*, *to_addrs*, *msg*[, *mail_options*, *rcpt_options*])

Send mail. The required arguments are an **RFC 822** from-address string, a list of **RFC 822** to-address strings (a bare string will be treated as a list with 1 address), and a message string. The caller may pass a list of ESMTP options (such as `8bitmime`) to be used in `MAIL FROM` commands as *mail_options*. ESMTP options (such as `DSN` commands) that should be used with all `RCPT` commands can be passed as *rcpt_options*. (If you need to use different ESMTP options to different recipients you have to use the low-level methods such as **`mail()`**, **`rcpt()`** and **`data()`** to send the message.)

**Note:** The *from_addr* and *to_addrs* parameters are used to construct the message envelope used by the transport agents. The `SMTP` does not modify the message headers in any way.

If there has been no previous `EHLO` or `HELO` command this session, this method tries ESMTP `EHLO` first. If the server does ESMTP, message size and each of the specified options will be passed to it (if the option is in the feature set the server advertises). If `EHLO` fails, `HELO` will be tried and ESMTP options suppressed.

This method will return normally if the mail is accepted for at least one recipient. Otherwise it will raise an exception. That is, if this method does not raise an exception, then someone should get your mail. If this method does not raise an exception, it returns a dictionary, with one entry for each recipient that was refused. Each entry contains a tuple of the SMTP error code and the accompanying error message sent by the server.

This method may raise the following exceptions:

`SMTPRecipientsRefused`

All recipients were refused. Nobody got the mail. The **`recipients`** attribute of the exception object is a dictionary with information about the refused recipients (like the one returned when at least one recipient was accepted).

`SMTPHeloError`

The server didn't reply properly to the `HELO` greeting.

**`SMTPSenderRefused`**

The server didn't accept the *from_addr*.

**`SMTPDataError`**

The server replied with an unexpected error code (other than a refusal of a recipient).

Unless otherwise noted, the connection will be open even after an exception is raised.

`SMTP.`**`quit`**`()`

Terminate the SMTP session and close the connection. Return the result of the SMTP `QUIT` command.

*Changed in version 2.6:* Return a value.

Low-level methods corresponding to the standard SMTP/ESMTP commands `HELP`, `RSET`, `NOOP`, `MAIL`, `RCPT`, and `DATA` are also supported. Normally these do not need to be called directly, so they are not documented here. For details, consult the module code.

## 20.12.2. SMTP Example

This example prompts the user for addresses needed in the message envelope ('To' and 'From' addresses), and the message to be delivered. Note that the headers to be included with the message must be included in the message as entered; this example doesn't do any processing of the **RFC 822** headers. In particular, the 'To' and 'From' addresses must be included in the message headers explicitly.

```
import smtplib

def prompt(prompt):
    return raw_input(prompt).strip()

fromaddr = prompt("From: ")
toaddrs  = prompt("To: ").split()
print "Enter message, end with ^D (Unix) or ^Z (Windows):"

# Add the From: and To: headers at the start!
msg = ("From: %s\r\nTo: %s\r\n\r\n"
       % (fromaddr, ", ".join(toaddrs)))
while 1:
    try:
        line = raw_input()
    except EOFError:
        break
    if not line:
        break
    msg = msg + line

print "Message length is " + repr(len(msg))

server = smtplib.SMTP('localhost')
server.set_debuglevel(1)
server.sendmail(fromaddr, toaddrs, msg)
server.quit()
```

**Note:** In general, you will want to use the `email` package's features to construct an email message, which you can then convert to a string and send via **`sendmail()`**; see *email: Examples*.