

---

# **flask-mail Documentation**

***Release 0.9.1***

**Dan Jacob**

December 26, 2016



<b>1</b>	<b>Links</b>	<b>3</b>
<b>2</b>	<b>Installing Flask-Mail</b>	<b>5</b>
<b>3</b>	<b>Configuring Flask-Mail</b>	<b>7</b>
<b>4</b>	<b>Sending messages</b>	<b>9</b>
<b>5</b>	<b>Bulk emails</b>	<b>11</b>
<b>6</b>	<b>Attachments</b>	<b>13</b>
<b>7</b>	<b>Unit tests and suppressing emails</b>	<b>15</b>
<b>8</b>	<b>Header injection</b>	<b>17</b>
<b>9</b>	<b>Signalling support</b>	<b>19</b>
<b>10</b>	<b>API</b>	<b>21</b>
	<b>Python Module Index</b>	<b>23</b>



One of the most basic functions in a web application is the ability to send emails to your users.

The **Flask-Mail** extension provides a simple interface to set up SMTP with your [Flask](#) application and to send messages from your views and scripts.



---

### Links

---

- [documentation](#)
- [source](#)
- [changelog](#)





---

## Installing Flask-Mail

---

Install with **pip** and **easy\_install**:

```
pip install Flask-Mail
```

or download the latest version from version control:

```
git clone https://github.com/mattupstate/flask-mail.git
cd flask-mail
python setup.py install
```

If you are using **virtualenv**, it is assumed that you are installing flask-mail in the same virtualenv as your Flask application(s).



---

## Configuring Flask-Mail

---

**Flask-Mail** is configured through the standard Flask config API. These are the available options (each is explained later in the documentation):

- **MAIL\_SERVER** : default **'localhost'**
- **MAIL\_PORT** : default **25**
- **MAIL\_USE\_TLS** : default **False**
- **MAIL\_USE\_SSL** : default **False**
- **MAIL\_DEBUG** : default **app.debug**
- **MAIL\_USERNAME** : default **None**
- **MAIL\_PASSWORD** : default **None**
- **MAIL\_DEFAULT\_SENDER** : default **None**
- **MAIL\_MAX\_EMAILS** : default **None**
- **MAIL\_SUPPRESS\_SEND** : default **app.testing**
- **MAIL\_ASCII\_ATTACHMENTS** : default **False**

In addition the standard Flask **TESTING** configuration option is used by **Flask-Mail** in unit tests (see below).

Emails are managed through a **Mail** instance:

```
from flask import Flask
from flask_mail import Mail

app = Flask(__name__)
mail = Mail(app)
```

In this case all emails are sent using the configuration values of the application that was passed to the **Mail** class constructor.

Alternatively you can set up your **Mail** instance later at configuration time, using the **init\_app** method:

```
mail = Mail()

app = Flask(__name__)
mail.init_app(app)
```

In this case emails will be sent using the configuration values from Flask's **current\_app** context global. This is useful if you have multiple applications running in the same process but with different configuration options.



---

## Sending messages

---

To send a message first create a Message instance:

```
from flask_mail import Message

@app.route("/")
def index():

    msg = Message("Hello",
                  sender="from@example.com",
                  recipients=["to@example.com"])
```

You can set the recipient emails immediately, or individually:

```
msg.recipients = ["you@example.com"]
msg.add_recipient("somebodyelse@example.com")
```

If you have set MAIL\_DEFAULT\_SENDER you don't need to set the message sender explicitly, as it will use this configuration value by default:

```
msg = Message("Hello",
              recipients=["to@example.com"])
```

If the sender is a two-element tuple, this will be split into name and address:

```
msg = Message("Hello",
              sender=("Me", "me@example.com"))

assert msg.sender == "Me <me@example.com>"
```

The message can contain a body and/or HTML:

```
msg.body = "testing"
msg.html = "<b>testing</b>"
```

Finally, to send the message, you use the Mail instance configured with your Flask application:

```
mail.send(msg)
```



---

## Bulk emails

---

Usually in a web application you will be sending one or two emails per request. In certain situations you might want to be able to send perhaps dozens or hundreds of emails in a single batch - probably in an external process such as a command-line script or cronjob.

In that case you do things slightly differently:

```
with mail.connect() as conn:
    for user in users:
        message = '...'
        subject = "hello, %s" % user.name
        msg = Message(recipients=[user.email],
                      body=message,
                      subject=subject)

        conn.send(msg)
```

The connection to your email host is kept alive and closed automatically once all the messages have been sent.

Some mail servers set a limit on the number of emails sent in a single connection. You can set the max amount of emails to send before reconnecting by specifying the **MAIL\_MAX\_EMAILS** setting.





---

## Attachments

---

Adding attachments is straightforward:

```
with app.open_resource("image.png") as fp:
    msg.attach("image.png", "image/png", fp.read())
```

See the [API](#) for details.

If `MAIL_ASCII_ATTACHMENTS` is set to **True**, filenames will be converted to an ASCII equivalent. This can be useful when using a mail relay that modify mail content and mess up Content-Disposition specification when filenames are UTF-8 encoded. The conversion to ASCII is a basic removal of non-ASCII characters. It should be fine for any unicode character that can be decomposed by NFKD into one or more ASCII characters. If you need romanization/transliteration (i.e.  $\beta \rightarrow ss$ ) then your application should do it and pass a proper ASCII string.



---

## Unit tests and suppressing emails

---

When you are sending messages inside of unit tests, or in a development environment, it's useful to be able to suppress email sending.

If the setting `TESTING` is set to `True`, emails will be suppressed. Calling `send()` on your messages will not result in any messages being actually sent.

Alternatively outside a testing environment you can set `MAIL_SUPPRESS_SEND` to **False**. This will have the same effect.

However, it's still useful to keep track of emails that would have been sent when you are writing unit tests.

In order to keep track of dispatched emails, use the `record_messages` method:

```
with mail.record_messages() as outbox:

    mail.send_message(subject='testing',
                      body='test',
                      recipients=emails)

    assert len(outbox) == 1
    assert outbox[0].subject == "testing"
```

The **outbox** is a list of `Message` instances sent.

The `blinker` package must be installed for this method to work.

Note that the older way of doing things, appending the **outbox** to the `g` object, is now deprecated.



---

## Header injection

---

To prevent [header injection](#) attempts to send a message with newlines in the subject, sender or recipient addresses will result in a `BadHeaderError`.



---

## Signalling support

---

New in version 0.4.

**Flask-Mail** now provides signalling support through a `email_dispatched` signal. This is sent whenever an email is dispatched (even if the email is not actually sent, i.e. in a testing environment).

A function connecting to the `email_dispatched` signal takes a `Message` instance as a first argument, and the Flask app instance as an optional argument:

```
def log_message(message, app):  
    app.logger.debug(message.subject)  
  
email_dispatched.connect(log_message)
```





**class** flask\_mail.**Mail** (*app=None*)

Manages email messaging

**Parameters** **app** – Flask instance

**connect** ()

Opens a connection to the mail host.

**send** (*message*)

Sends a single message instance. If TESTING is True the message will not actually be sent.

**Parameters** **message** – a Message instance.

**send\_message** (*\*args, \*\*kwargs*)

Shortcut for send(msg).

Takes same arguments as Message constructor.

**Versionadded** 0.3.5

**class** flask\_mail.**Attachment** (*filename=None, content\_type=None, data=None, disposition=None, headers=None*)

Encapsulates file attachment information.

**Versionadded** 0.3.5

**Parameters**

- **filename** – filename of attachment
- **content\_type** – file mimetype
- **data** – the raw file data
- **disposition** – content-disposition (if any)

**class** flask\_mail.**Connection** (*mail*)

Handles connection to host.

**send** (*message, envelope\_from=None*)

Verifies and sends message.

**Parameters**

- **message** – Message instance.
- **envelope\_from** – Email address to be used in MAIL FROM command.

**send\_message** (\*args, \*\*kwargs)

Shortcut for send(msg).

Takes same arguments as Message constructor.

**Versionadded** 0.3.5

```
class flask_mail.Message(subject='', recipients=None, body=None, html=None, sender=None,
                        cc=None, bcc=None, attachments=None, reply_to=None,
                        date=None, charset=None, extra_headers=None, mail_options=None,
                        rcpt_options=None)
```

Encapsulates an email message.

#### Parameters

- **subject** – email subject header
- **recipients** – list of email addresses
- **body** – plain text message
- **html** – HTML message
- **sender** – email sender address, or **MAIL\_DEFAULT\_SENDER** by default
- **cc** – CC list
- **bcc** – BCC list
- **attachments** – list of Attachment instances
- **reply\_to** – reply-to address
- **date** – send date
- **charset** – message character set
- **extra\_headers** – A dictionary of additional headers for the message
- **mail\_options** – A list of ESMTP options to be used in MAIL FROM command
- **rcpt\_options** – A list of ESMTP options to be used in RCPT commands

**add\_recipient** (recipient)

Adds another recipient to the message.

**Parameters** **recipient** – email address of recipient.

**attach** (filename=None, content\_type=None, data=None, disposition=None, headers=None)

Adds an attachment to the message.

#### Parameters

- **filename** – filename of attachment
- **content\_type** – file mimetype
- **data** – the raw file data
- **disposition** – content-disposition (if any)

**f**

flask-mail, 3  
flask\_mail, 21



## A

`add_recipient()` (`flask_mail.Message` method), [22](#)

`attach()` (`flask_mail.Message` method), [22](#)

`Attachment` (class in `flask_mail`), [21](#)

## C

`connect()` (`flask_mail.Mail` method), [21](#)

`Connection` (class in `flask_mail`), [21](#)

## F

`flask-mail` (module), [1](#)

`flask_mail` (module), [21](#)

## M

`Mail` (class in `flask_mail`), [21](#)

`Message` (class in `flask_mail`), [22](#)

## S

`send()` (`flask_mail.Connection` method), [21](#)

`send()` (`flask_mail.Mail` method), [21](#)

`send_message()` (`flask_mail.Connection` method), [21](#)

`send_message()` (`flask_mail.Mail` method), [21](#)