django-mailbox Documentation

Release 3.3

Adam Coddington

Aug 18, 2017

Contents

How many times have you had to consume some sort of POP3, IMAP, or local mailbox for incoming content, or had to otherwise construct an application driven by e-mail? One too many times, I'm sure.

This small Django application will allow you to specify mailboxes that you would like consumed for incoming content; the e-mail will be stored, and you can process it at will (or, if you're in a hurry, by subscribing to a signal).

Contents:

CHAPTER 1

Installation

1. You can either install from pip:

   ```
   pip install django-mailbox
   ```

   or checkout and install the source from the github repository:

   ```
   git clone https://github.com/coddingtonbear/django-mailbox.git
   cd django-mailbox
   python setup.py install
   ```

2. After you have installed the package, add `django_mailbox` to the `INSTALLED_APPS` setting in your project's `settings.py` file.

3. From your project folder, run `python manage.py migrate django_mailbox` to create the required database tables.

4. Head to your project's Django Admin and create a mailbox to consume.

Note: Once you have entered a mailbox to consume, you can easily verify that you have properly configured your mailbox by either:

- From the Django Admin, using the 'Get New Mail' action from the action dropdown on the Mailbox changelist (http://yourproject.com/admin/django_mailbox/mailbox/).

- *Or* from a shell opened to your project's directory, using the `getmail` management command by running:

  ```
  python manage.py getmail
  ```

Supported Mailbox Types

Django Mailbox supports polling both common internet mailboxes like POP3 and IMAP as well as local file-based mailboxes.

Table 2.1: 'Protocol' Options

Mailbox Type	'Protocol'://	Notes
POP3	`pop3://`	Can also specify SSL with `pop3+ssl://`
IMAP	`imap://`	Can also specify SSL with `imap+ssl://` or STARTTLS with `imap+tls`; additional configuration is also possible: see *POP3 and IMAP Mailboxes* for details.
Gmail IMAP	`gmail+ssl://`	Uses OAuth authentication for Gmail's IMAP transport. See *Gmail IMAP with Oauth2 authentication* for details.
Maildir	`maildir://`	
Mbox	`mbox://`	
Babyl	`babyl://`	
MH	`mh://`	
MMDF	`mmdf://`	
Piped Mail	*empty*	See *Receiving mail directly from Exim4 or Postfix via a pipe*

> **Warning:** Unless you are using IMAP's 'Archive' feature, this will delete any messages it can find in the inbox you specify; do not use an e-mail inbox that you would like to share between applications.

POP3 and IMAP Mailboxes

Mailbox URIs are in the normal URI format:

```
protocol://username:password@domain
```

Basic IMAP Example: `imap://username:password@server`

Basic POP3 Example: `pop3://username:password@server`

Most mailboxes these days are SSL-enabled; if yours use plain SSL add `+ssl` to the protocol section of your URI, but for STARTTLS add `+tls`. Also, if your username or password include any non-ascii characters, they should be URL-encoded (for example, if your username includes an `@`, it should be changed to `%40` in your URI).

For a verbose example, if you have an account named `youremailaddress@gmail.com` with a password of `1234` on GMail, which uses a IMAP server of `imap.gmail.com` (requiring SSL) and you would like to fetch new emails from folder named *Myfolder* and archive them after processing into a folder named `Archived`, you would enter the following as your URI:

```
imap+ssl://youremailaddress%40gmail.com:1234@imap.gmail.com?archive=Archived
```

Additional IMAP Mailbox Features

If you are using an IMAP Mailbox, you have two additional configuration options that you can set by appending parameters to the end of your mailbox URI.

Specifying the source folder

Although by default, Django Mailbox will consume messages from your 'INBOX' folder, you can specify the folder from which you'd like messages consumed by specifying the `folder` URI query parameter; for example, to instead consume from the folder named 'MyFolder', you could add `?folder=MyFolder` to the end of your URI:

```
imap+ssl://youremailaddress%40gmail.com:1234@imap.gmail.com?folder=MyFolder
```

Specifying an archive folder

Django Mailbox will delete messages immediately after processing them, but you can specify an IMAP folder to which the messages should be copied before the original message is deleted.

To archive email messages, add the archive folder name as a query parameter to the URI. For example, if your mailbox has a folder named `myarchivefolder` that you would like to copy messages to after processing, add `?archive=myarchivefolder` to the end of the URI:

```
imap+ssl://youremailaddress%40gmail.com:1234@imap.gmail.com?archive=myarchivefolder
```

Gmail IMAP with Oauth2 authentication

For added security, Gmail supports using OAuth2 for authentication. To handle the handshake and storing the credentials, use python-social-auth.

The Gmail Mailbox is also a regular IMAP mailbox, but the password you specify will be ignored if OAuth2 authentication succeeds. It will fall back to use your specified password as needed.

Build your URI accordingly:

```
gmail+ssl://youremailaddress%40gmail.com:oauth2@imap.gmail.com?archive=Archived
```

Local File-based Mailboxes

If you happen to want to consume a file-based mailbox like an Maildir, Mbox, Babyl, MH, or MMDF mailbox, you can use this too by entering the appropriate 'protocol' in the URI. If you had a maildir, for example, at /var/mail/, you would enter a URI like:

```
maildir:///var/mail
```

Note that there is an additional / in the above URI after the protocol; this is important.

Getting incoming mail

If you are utilizing one of the polling methods above, you will need to periodically poll the mailbox for messages using one of the below methods. If you are receiving mail directly from a mailserver via a pipe – using the `processincomingmessage` management command – you need not concern yourself with this section.

In your code

Mailbox instances have a method named `get_new_mail`; this method will gather new messages from the server.

Using the Django Admin

From the 'Mailboxes' page in the Django Admin, check the box next to each of the mailboxes you'd like to fetch e-mail from, select 'Get new mail' from the action selector at the top of the list of mailboxes, then click 'Go'.

Using a cron job

You can easily consume incoming mail by running the management command named `getmail` (optionally with an argument of the name of the mailbox you'd like to get the mail for).:

```
python manage.py getmail
```

Receiving mail directly from Exim4 or Postfix via a pipe

Django Mailbox's `processincomingmessage` management command accepts, via `stdin`, incoming messages. You can configure Postfix or Exim4 to pipe incoming mail to this management command to import messages directly without polling.

You need not configure mailbox settings when piping-in messages, mailbox entries will be automatically created matching the e-mail address to which incoming messages are sent, but if you would like to specify the mailbox name, you may provide a single argument to the `processincmingmessage` command specifying the name of the mailbox you would like it to use (and, if necessary, create).

Receiving Mail from Exim4

To configure Exim4 to receive incoming mail, start by adding a new router configuration to your Exim4 configuration like:

```
django_mailbox:
  debug_print = 'R: django_mailbox for $localpart@$domain'
  driver = accept
  transport = send_to_django_mailbox
  domains = mydomain.com
  local_parts = emailusernameone : emailusernametwo
```

Make sure that the e-mail addresses you would like handled by Django Mailbox are not handled by another router; you may need to disable some existing routers.

Change the contents of `local_parts` to match a colon-delimited list of usernames for which you would like to receive mail. For example, if one of the e-mail addresses targeted at this machine is `jane@example.com`, the contents of `local_parts` would be, simply `jane`.

Note: If you would like messages addressed to *any* account *@mydomain.com* to be delivered to django_mailbox, simply omit the above `local_parts` setting. In the same vein, if you would like messages addressed to any domain or any local domains, you can omit the `domains` setting or set it to `+local_domains` respectively.

Next, a new transport configuration to your Exim4 configuration:

```
send_to_django_mailbox:
  driver = pipe
  command = /path/to/your/environments/python /path/to/your/projects/manage.py␣
→processincomingmessage
  user = www-data
  group = www-data
  return_path_add
  delivery_date_add
```

Like your router configuration, transport configuration should be altered to match your environment. First, modify the `command` setting such that it points at the proper python executable (if you're using a virtual environment, you'll want to direct that at the python executable in your virtual environment) and project `manage.py` script. Additionally, you'll need to set `user` and `group` such that they match a reasonable user and group (on Ubuntu, `www-data` suffices for both).

Receiving mail from Postfix

Although I have not personally tried using Postfix for this, Postfix is capable of delivering new mail to a script using `pipe`. Please consult the Postfix documentation for pipe here. You may want to consult the above Exim4 configuration for tips.

Subscribing to the incoming mail signal

To subscribe to the incoming mail signal, following this lead:

```python
from django_mailbox.signals import message_received
from django.dispatch import receiver

@receiver(message_received)
def dance_jig(sender, message, **args):
    print "I just recieved a message titled %s from a mailbox named %s" % (message.
→subject, message.mailbox.name, )
```

> **Warning:** As with all django signals, this should be loaded either in an app's models.py or somewhere else loaded early on. If you do not load it early enough, the signal may be fired before your signal handler's registration is processed!

Appendix

Class Documentation

Mailbox

class django_mailbox.models.**Mailbox**(*id*, *name*, *uri*, *from_email*, *active*, *last_polling*)

> **Parameters**
>
> - **id** (`AutoField`) – Id
>
> - **name** (`CharField`) – Name
>
> - **uri** (`CharField`) – Example: imap+ssl://myusername:mypassword@someserver Internet transports include 'imap' and 'pop3'; common local file transports include 'maildir', 'mbox', and less commonly 'babyl', 'mh', and 'mmdf'. Be sure to urlencode your username and password should they contain illegal characters (like @, :, etc).
>
> - **from_email** (`CharField`) – Example: MailBot <mailbot@yourdomain.com>'From' header to set for outgoing email.If you do not use this e-mail inbox for outgoing mail, this setting is unnecessary.If you send e-mail without setting this, your 'From' header will'be set to match the setting *DEFAULT_FROM_EMAIL*.
>
> - **active** (`BooleanField`) – Check this e-mail inbox for new e-mail messages during polling cycles. This checkbox does not have an effect upon whether mail is collected here when this mailbox receives mail from a pipe, and does not affect whether e-mail messages can be dispatched from this mailbox.
>
> - **last_polling** (`DateTimeField`) – The time of last successful polling for messages.It is blank for new mailboxes and is not set for mailboxes that only receive messages via a pipe.

> exception **DoesNotExist**

> exception Mailbox.**MultipleObjectsReturned**

> Mailbox.**active_mailboxes = <django_mailbox.models.ActiveMailboxManager object>**

`Mailbox.`**`archive`**
> Returns (if specified) the folder to archive messages to.

`Mailbox.`**`folder`**
> Returns (if specified) the folder to fetch mail from.

`Mailbox.`**`get_connection`**`()`
> Returns the transport instance for this mailbox.
>
> These will always be instances of *django_mailbox.transports.base.EmailTransport*.

`Mailbox.`**`get_new_mail`**`(`*condition=None*`)`
> Connect to this transport and fetch new messages.

`Mailbox.`**`location`**
> Returns the location (domain and path) of messages.

`Mailbox.`**`messages`**
> Accessor to the related objects manager on the reverse side of a many-to-one relation.
>
> In the example:

```python
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

> `parent.children` is a `ReverseManyToOneDescriptor` instance.
>
> Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Mailbox.`**`objects = <django.db.models.manager.Manager object>`**

`Mailbox.`**`password`**
> Returns the password to use for fetching messages.

`Mailbox.`**`port`**
> Returns the port to use for fetching messages.

`Mailbox.`**`process_incoming_message`**`(`*message*`)`
> Process a message incoming to this mailbox.

`Mailbox.`**`record_outgoing_message`**`(`*message*`)`
> Record an outgoing message associated with this mailbox.

`Mailbox.`**`type`**
> Returns the 'transport' name for this mailbox.

`Mailbox.`**`use_ssl`**
> Returns whether or not this mailbox's connection uses SSL.

`Mailbox.`**`use_tls`**
> Returns whether or not this mailbox's connection uses STARTTLS.

`Mailbox.`**`username`**
> Returns the username to use for fetching messages.

Message

class `django_mailbox.models.`**`Message`**`(`*id*, *mailbox*, *subject*, *message_id*, *in_reply_to*, *from_header*, *to_header*, *outgoing*, *body*, *encoded*, *processed*, *read*, *eml*`)`

> **Parameters**

- **id** (*AutoField*) – Id

- **mailbox_id** (ForeignKey to [*Mailbox*](#)) – Mailbox

- **subject** (*CharField*) – Subject

- **message_id** (*CharField*) – Message id

- **in_reply_to_id** (ForeignKey to [*Message*](#)) – In reply to

- **from_header** (*CharField*) – From header

- **to_header** (*TextField*) – To header

- **outgoing** (*BooleanField*) – Outgoing

- **body** (*TextField*) – Body

- **encoded** (*BooleanField*) – True if the e-mail body is Base64 encoded

- **processed** (*DateTimeField*) – Processed

- **read** (*DateTimeField*) – Read

- **eml** (*FileField*) – Original full content of message

exception **DoesNotExist**

exception Message.**MultipleObjectsReturned**

Message.**address**

Property allowing one to get the relevant address(es).

In earlier versions of this library, the model had an *address* field storing the e-mail address from which a message was received. During later refactorings, it became clear that perhaps storing sent messages would also be useful, so the address field was replaced with two separate fields.

Message.**attachments**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

Message.**delete**(*args*, ***kwargs*)

Delete this message and all stored attachments.

Message.**eml**

The descriptor for the file attribute on the model instance. Returns a FieldFile when accessed so you can do stuff like:

```
>>> from myapp.models import MyModel
>>> instance = MyModel.objects.get(pk=1)
>>> instance.file.size
```

Assigns a file object on assignment so you can do:

```
>>> with open('/path/to/hello.world', 'r') as f:
...     instance.file = File(f)
```

Message.**from_address**
  Returns the address (as a list) from which this message was received

  ---

  Note: This was once (and probably should be) a string rather than a list, but in a pull request received long, long ago it was changed; presumably to make the interface identical to that of *to_addresses*.

Message.**get_body**()
  Returns the *body* field of this record.

  This will automatically base64-decode the message contents if they are encoded as such.

Message.**get_email_object**()
  Returns an *email.message.Message* instance representing the contents of this message and all attachments.

  See [email.Message.Message] for more information as to what methods and properties are available on *email.message.Message* instances.

  Note: Depending upon the storage methods in use (specifically – whether `DJANGO_MAILBOX_STORE_ORIGINAL_MESSAGE` is set to `True`, this may either create a "re-hydrated" message using stored attachments, or read the message contents stored on-disk.

Message.**get_next_by_processed**(*moreargs*, ***morekwargs*)

Message.**get_previous_by_processed**(*moreargs*, ***morekwargs*)

Message.**html**
  Returns the message body matching content type 'text/html'.

Message.**in_reply_to**
  Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

  In the example:

```python
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

  `child.parent` is a `ForwardManyToOneDescriptor` instance.

Message.**incoming_messages = <django_mailbox.models.IncomingMessageManager object>**

Message.**mailbox**
  Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

  In the example:

```python
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

  `child.parent` is a `ForwardManyToOneDescriptor` instance.

Message.**objects = <django.db.models.manager.Manager object>**

Message.**outgoing_messages = <django_mailbox.models.OutgoingMessageManager object>**

Message.**replies**
  Accessor to the related objects manager on the reverse side of a many-to-one relation.

  In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

Message.**reply**(*message*)
> Sends a message as a reply to this message instance.
>
> Although Django's e-mail processing will set both Message-ID and Date upon generating the e-mail message, we will not be able to retrieve that information through normal channels, so we must pre-set it.

Message.**set_body**(*body*)
> Set the *body* field of this record.
>
> This will automatically base64-encode the message contents to circumvent a limitation in earlier versions of Django in which no fields existed for storing arbitrary bytes.

Message.**text**
> Returns the message body matching content type 'text/plain'.

Message.**to_addresses**
> Returns a list of addresses to which this message was sent.

Message.**unread_messages = <django_mailbox.models.UnreadMessageManager object>**

Message Attachment

class django_mailbox.models.**MessageAttachment**(*id*, *message*, *headers*, *document*)

>   **Parameters**
>
>   - **id** (*AutoField*) – Id
>   - **message_id** (ForeignKey to [*Message*](#)) – Message
>   - **headers** (*TextField*) – Headers
>   - **document** (*FileField*) – Document

exception **DoesNotExist**

exception MessageAttachment.**MultipleObjectsReturned**

MessageAttachment.**delete**(**args*, ***kwargs*)
> Deletes the attachment.

MessageAttachment.**document**
> The descriptor for the file attribute on the model instance. Returns a FieldFile when accessed so you can do stuff like:

```
>>> from myapp.models import MyModel
>>> instance = MyModel.objects.get(pk=1)
>>> instance.file.size
```

Assigns a file object on assignment so you can do:

```
>>> with open('/path/to/hello.world', 'r') as f:
...     instance.file = File(f)
```

MessageAttachment.**get_filename**()
> Returns the original filename of this attachment.

MessageAttachment.**items**()

MessageAttachment.**message**
> Accessor to the related object on the forward side of a many-to-one or one-to-one relation.
>
> In the example:

```python
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

> child.parent is a ForwardManyToOneDescriptor instance.

MessageAttachment.**objects = <django.db.models.manager.Manager object>**

Message Storage Details

First, it may be helpful to know a little bit about how e-mail messages are actually sent across the wire:

```
MIME-Version: 1.0
Received: by 10.221.0.211 with HTTP; Sun, 20 Jan 2013 12:07:07 -0800 (PST)
X-Originating-IP: [24.22.122.177]
Date: Sun, 20 Jan 2013 12:07:07 -0800
Delivered-To: test@adamcoddington.net
Message-ID: <CAMdmm+jYCgrxrekAxszmDnBjAytcBym-Ec+uM-+HEtzuKy=M_g@mail.gmail.com>
Subject: Message With Attachment
From: Adam Coddington <test@adamcoddington.net>
To: Adam Coddington <test@adamcoddington.net>
Content-Type: multipart/mixed; boundary=047d7b33dd729737fe04d3bde348


--047d7b33dd729737fe04d3bde348
Content-Type: text/plain; charset=UTF-8

This message has an attachment.

--047d7b33dd729737fe04d3bde348
Content-Type: image/png; name="heart.png"
Content-Disposition: attachment; filename="heart.png"
Content-Transfer-Encoding: base64
X-Attachment-Id: f_hc6mair60

iVBORw0KGgoAAAANSUhEUgAAAA4AAAAOCAYAAAFoTx1HAAAAzUlEQVQoz32RWxXDIBBBEr4NIQEIl
ICESkFAJkRAJSIgEpEQCEqYfu6QUkn7sCcyDGQiSACKSKCAkGwBJwhDwZQNMEiYAIBdQvk7rfaHf
AO8NBJwCxTGhtFgTHVNaNaJeWFu44AXEHzKCktc7zZ0vss+bMoHSiM2b9mQoX1eZCgGqnWskY3gi
XXAAxb8BqFiUgBNY7k49Tu/kV7UKPsefrjEOT9GmghYzrk9V03pjDGYKj3d0c06dKZkpTboRaD9o
B+1m2m81d2Az948xzgdjLaFe95e83AAAAABJRU5ErkJggg==

--047d7b33dd729737fe04d3bde348--
```

Messages are grouped into multiple message payload parts, and should binary attachments exist, they are encoded into text using, generally, base64 or quoted-printable encodings.

Earlier versions of this library would preserve the above text verbatim in the database, but neither of the above encodings are very efficient methods of storing binary data, and databases aren't really ideal for storing large chunks of binary data anyway.

Modern versions of this library (>=2.1) will walk through the original message, write `models.MessageAttachment` records for each non-text attachment, and alter the message body removing the original payload component, but writing a custom header providing the library enough information to re-build the message in the event that one needs a python `email.message.Message` object.

```
MIME-Version: 1.0
Received: by 10.221.0.211 with HTTP; Sun, 20 Jan 2013 12:07:07 -0800 (PST)
X-Originating-IP: [24.22.122.177]
Date: Sun, 20 Jan 2013 12:07:07 -0800
Delivered-To: test@adamcoddington.net
Message-ID: <CAMdmm+jYCgrxrekAxszmDnBjAytcBym-Ec+uM-+HEtzuKy=M_g@mail.gmail.com>
Subject: Message With Attachment
From: Adam Coddington <test@adamcoddington.net>
To: Adam Coddington <test@adamcoddington.net>
Content-Type: multipart/mixed; boundary=047d7b33dd729737fe04d3bde348


--047d7b33dd729737fe04d3bde348
Content-Type: text/plain; charset=UTF-8

This message has an attachment.


--047d7b33dd729737fe04d3bde348
X-Django-Mailbox-Interpolate-Attachment: 1308



--047d7b33dd729737fe04d3bde348--
```

The above payload is what would continue to be stored in the database. Although in this constructed example, this reduces the message's size only marginally, in most instances, attached files are much larger than the attachment shown here.

Note: Email message bodies are `base-64` encoded when stored in the database.

Although the attachment is no longer preserved in the message body above, and only the `X-Django-Mailbox-Interpolate-Attachment: 1308` header remains in the place of the original attachment, the attachment was stored in a `django_mailbox.MesageAttachment` record:

Field	Value	Description
Pri-mary Key	`1308`	Uniquely generated for each attachment.
Head-ers	`Content-Type: image/png; name="heart.png" Content-Disposition: attachment; filename="heart.png" Content-Transfer-Encoding: base64 X-Attachment-Id: f_hc6mair60`	Raw headers from the actual message's payload part.
File	`(binary file object)`	References a stored-on-disk binary file corresponding with this attachment.

And were one to run the `django_mailbox.Message` instance's `get_email_object` method, the following message will be returned:

```
MIME-Version: 1.0
Received: by 10.221.0.211 with HTTP; Sun, 20 Jan 2013 12:07:07 -0800 (PST)
X-Originating-IP: [24.22.122.177]
Date: Sun, 20 Jan 2013 12:07:07 -0800
```

```
Delivered-To: test@adamcoddington.net
Message-ID: <CAMdmm+jYCgrxrekAxszmDnBjAytcBym-Ec+uM-+HEtzuKy=M_g@mail.gmail.com>
Subject: Message With Attachment
From: Adam Coddington <test@adamcoddington.net>
To: Adam Coddington <test@adamcoddington.net>
Content-Type: multipart/mixed; boundary=047d7b33dd729737fe04d3bde348

--047d7b33dd729737fe04d3bde348
Content-Type: text/plain; charset=UTF-8

This message has an attachment.

--047d7b33dd729737fe04d3bde348
Content-Type: image/png; name="heart.png"
Content-Disposition: attachment; filename="heart.png"
X-Attachment-Id: f_hc6mair60
Content-Transfer-Encoding: base64

iVBORw0KGgoAAAANSUhEUgAAAA4AAAAOCAYAAAFoTx1HAAAAzUlEQVQoz32RWxXDIBBEr4NIQEIl
ICESkFAJkRAJSIgEpEQCEqYfu6QUkn7sCcyDGQiSACKSKCAkGwBJwhDwZQNMEiYAIBdQvk7rfaHf
AO8NBJwCxTGhtFgTHVNaNaJeWFu44AXEHzKCktc7zZ0vss+bMoHSiM2b9mQoX1eZCgGqnWskY3gi
XXAAxb8BqFiUgBNY7k49Tu/kV7UKPsefrjEOT9GmghYzrk9V03pjDGYKj3d0c06dKZkpTboRaD9o
B+1m2m81d2Az948xzgdjLaFe95e83AAAAABJRU5ErkJggg==

--047d7b33dd729737fe04d3bde348--
```

Note: Note that although the above is functionally identical to the originally received message, there were changes in the order of headers in rehydrated message components, and whitespace changes are also possible (but not shown above).

Settings

- DJANGO_MAILBOX_ADMIN_ENABLED

    - Default: `True`

    - Type: `boolean`

    - Controls whether mailboxes appear in the Django Admin.

- DJANGO_MAILBOX_STRIP_UNALLOWED_MIMETYPES

    - Default: `False`

    - Type: `boolean`

    - Controls whether or not we remove mimetypes not specified in DJANGO_MAILBOX_PRESERVED_MIMETYPES from the message prior to storage.

- DJANGO_MAILBOX_ALLOWED_MIMETYPES

    - Default `['text/html', 'text/plain']`

    - Type: `list`

    - Should DJANGO_MAILBOX_STRIP_UNALLOWED_MIMETYPES be `True`, this is a list of mimetypes that will not be stripped from the message prior to processing attachments. Has no effect unless

> `DJANGO_MAILBOX_STRIP_UNALLOWED_MIMETYPES` is set to `True`.

- `DJANGO_MAILBOX_TEXT_STORED_MIMETYPES`

    - Default: `['text/html', 'text/plain']`

    - Type: `list`

    - A list of mimetypes that will remain stored in the text body of the message in the database. See *Message Storage Details*.

- `DJANGO_MAILBOX_ALTERED_MESSAGE_HEADER`

    - Default: `X-Django-Mailbox-Altered-Message`

    - Type: `string`

    - Header to add to a message payload part in the event that the message cannot be reproduced accurately. Possible values include:

        * `Missing`: The message could not be reconstructed because the message payload component (stored outside this database record) could not be found. This will be followed by a semicolon (`;`) and a short, more detailed description of which record was not found.

        * `Stripped` The message could not be reconstructed because the message payload component was intentionally stripped from the message body prior to storage. This will be followed by a semicolon (`;`) and a short, more detailed description of why this payload component was stripped.

- `DJANGO_MAILBOX_ATTACHMENT_INTERPOLATION_HEADER`

    - Default: `X-Django-Mailbox-Interpolate-Attachment`

    - Type: `string`

    - Header to add to the temporary 'dehydrated' message body in lieu of a non-text message payload component. The value of this header will be used to 'rehydrate' the message into a proper e-mail object in the event of a message instance's `get_email_object` method being called. Value of this field is the primary key of the `django_mailbox.MessageAttachment` instance currently storing this payload component's contents.

- `DJANGO_MAILBOX_ATTACHMENT_UPLOAD_TO`

    - Default: `mailbox_attachments/%Y/%m/%d/`

    - Type: `string`

    - Attachments will be saved to this location. Specifies the `upload_to` setting for the attachment FileField. For more on FileFields and upload_to, see the Django docs

- `DJANGO_MAILBOX_MAX_MESSAGE_SIZE`

    - Default: `False`

    - Type: `integer`

    - If this is set, it will be read as a number of bytes. Any messages above that size will not be downloaded. `2000000` is 2 Megabytes.

- `DJANGO_MAILBOX_STORE_ORIGINAL_MESSAGE`

    - Default: `False`

    - Type: `boolean`

    - Controls whether or not we store original messages in `eml` field

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index