Postmarker Documentation

Release 0.13.0

Dmitry Dygalo

Nov 25, 2018

Contents

Some basic concepts of Postmarker library:

- Provide as simple as possible interface.
- Use terms from Postmark wherever it may be applicable. The attributes of all classes are provided **as is**, without transformation to snake case. We don't want to introduce new names for existing entities.
- Support as many Python versions as possible.

Contents:

CHAPTER 1

Usage

For the first step you have to initialize the client with a proper API token. Postmarker provides a single `PostmarkClient` class for Server and Account APIs.

```
from postmarker.core import PostmarkClient


>>> postmark = PostmarkClient(server_token='SERVER_TOKEN', account_token='ACCOUNT_
↪TOKEN')
```

The variable `postmark` from the example above will be used in the next examples. By default, logging is configured to be silent. To enable it, pass `verbosity` to client instantiation:

```
>>> postmark = PostmarkClient(server_token='SERVER_TOKEN', account_token='ACCOUNT_
↪TOKEN', verbosity=3)
```

With `verbosity=3` every request and response will be logged to the console.

By default logs are streamed to `sys.stdout` to be in line with the Twelve-Factor App. To change it you could use `logs_stream` parameter in `PostmarkClient`.

For timeouts and max_retries handling there are two parameters - `max_retries` and `timeout`. The `timeout` value is passed to requests.Session.request. `max_retries` is passed to requests.adapters.HTTPAdapter.

Alternatively you could use `PostmarkClient.from_config` method, which is helpful to instantiate `PostmarkClient` from dict-like object, like Tornado or Flask configurations.

```
>>> config = {'POSTMARK_SERVER_TOKEN': 'foo', 'POSTMARK_TIMEOUT': 1}
>>> postmark = PostmarkClient.from_config(config, is_uppercase=True)
```

With `is_uppercase` set to `True` keys will be converted to upper case before lookup in the config object. The default value for `is_uppercase` is `False`. All arguments names from `PostmarkClient.__init__` will be looked up in the config object with default prefix *postmark_*. To use different prefix, just pass `prefix='my_prefix_'` to `PostmarkClient.from_config` method.

```
>>> config = {'my_prefix_server_token': 'foo', 'my_prefix_timeout': 1}
>>> postmark = PostmarkClient.from_config(config, prefix='my_prefix_')
```

CHAPTER 2

Emails

This module provides basic ways to send emails.

2.1 Basic

Sending emails is super easy! Here is the simplest case:

```
>>> postmark.emails.send(
    From='sender@example.com',
    To='receiver@example.com',
    Subject='Postmark test',
    HtmlBody='<html><body><strong>Hello</strong> dear Postmark user.</body></html>'
)
{'ErrorCode': 0,
 'Message': 'OK',
 'MessageID': 'fbff8f29-1742-48ee-93c2-f881c7049402',
 'SubmittedAt': '2016-10-05T06:51:08.9753663-04:00',
 'To': 'receiver@example.com'}
```

Or send `MIMEText`/`MIMEMultipart` instances:

```
>>> message = MIMEText('Text')
>>> message['From'] = 'sender@example.com'
>>> message['To'] = 'receiver@example.com'
>>> postmark.emails.send(message=message)
```

To specify multiple recipients (or `Cc` / `Bcc`) you could pass values as a list or string with comma separated values:

```
>>> postmark.emails.send(
    From='sender@example.com',
    To=['first@example.com', 'second@example.com'],  # The same as 'first@example.com,
↪ second@example.com'
    Subject='Postmark test',
```

(continues on next page)

(continued from previous page)

```
    HtmlBody='<html><body><strong>Hello</strong> dear Postmark user.</body></html>'
)
```

Headers could be specified as a dict:

```
>>> postmark.emails.send(
    From='sender@example.com',
    To='receiver@example.com',
    Headers={'X-Accept-Language': 'en-us, en'},
    Subject='Postmark test',
    HtmlBody='<html><body><strong>Hello</strong> dear Postmark user.</body></html>'
)
```

Attachments could be specified as a list of items in the following forms:

```
# Dictionary
>>> msg_1 = {"Name": "readme.txt", "Content": "dGVzdCBjb250ZW50", "ContentType":
↪"text/plain"}
# Tuple
>>> msg_2 = ("readme.txt", "dGVzdCBjb250ZW50", "text/plain")
# MIMEBase instance
>>> msg_3 = MIMEBase('text', 'plain')
>>> msg_3.set_payload('dGVzdCBjb250ZW50')
>>> msg_3.add_header('Content-Disposition', 'attachment', filename='readme.txt')
# File path string. Content type will be detected automatically
>>> msg_4 = '/home/user/readme.txt'
```

Note! Content should be encoded as Base64 string. Then pass the attachments to *send()*:

```
>>> postmark.emails.send(
    From='sender@example.com',
    To='receiver@example.com',
    Subject='Postmark test',
    HtmlBody='<html><body><strong>Hello</strong> dear Postmark user.</body></html>',
    Attachments=[msg_1, msg_2, msg_3, msg_4]
)
```

Postmarker also supports inline images. Here are the possible usage options:

```
>>> tuple_image = ('image', 'content', 'image/png', 'cid:image@example.com')
>>> mime = MIMEImage('test content', 'png', name='image1.png')
>>> mime.add_header('Content-ID', '<image1@example.com>')
>>> mime_inline = MIMEImage('test content', 'png', name='image3.png')
>>> mime_inline.add_header('Content-ID', '<image3@example.com>')
>>> mime_inline.add_header('Content-Disposition', 'inline', filename='image3.png')
>>> postmark.emails.send(
    From='sender@example.com',
    To='receiver@example.com',
    Subject='Postmark test',
    HtmlBody='<html><body><strong>Hello</strong> dear Postmark user.</body></html>',
    Attachments=[tuple_image, mime, mime_inline]
)
```

To send email in a batch there is *send_batch()* method.

```
>>> postmark.emails.send_batch(
    {
```

(continues on next page)

(continued from previous page)

```
        'From': 'sender@example.com',
        'To': 'receiver@example.com',
        'Subject': 'Postmark test',
        'HtmlBody': '<html><body><strong>Hello</strong> dear Postmark user.</body></
↪html>',
    },
    {
        'From': 'sender2@example.com',
        'To': 'receiver2@example.com',
        'Subject': 'Postmark test 2',
        'HtmlBody': '<html><body><strong>Hello</strong> dear Postmark user.</body></
↪html>',
    }
)
```

You can pass either *Email*/MIMEText/MIMEMultipart instances or dictionaries. Additionally, you may pass extra keywords to use with every email in a batch.

The batch size is not limited, but if the batch has more than 500 emails, then before sending, it will be split into chunks of 500 emails.

Postmark provides an interface to send emails created with a template. Example of usage with Postmarker:

```
>>> postmark.emails.send_with_template(
    TemplateId=123,
    TemplateModel={'username': 'Test'}
    From='sender@example.com',
    To='receiver@example.com',
)
```

Attachments and headers work in the same way as in basic email sending.

2.2 Advanced

To get more flexibility it is possible to use *Email* objects. *EmailManager* uses them internally to send emails. To instantiate one:

```
>>> email = postmark.emails.Email(
    From='sender@example.com',
    To='receiver@example.com',
    Subject='Postmark test',
    HtmlBody='<html><body><strong>Hello</strong> dear Postmark user.</body></html>'
)
```

Then send it:

```
>>> email.send()
```

To specify headers:

```
>>> email['X-Accept-Language'] = 'en-us, en'
```

Also it is possible to remove the header:

```
>>> del email['X-Accept-Language']
```

To add an attachment to an email there is *attach()* method.

```
>>> email.attach(msg_1)
```

To attach multiple attachments, pass them all to *attach()*:

```
>>> email.attach(msg_1, msg_2, msg_3)
```

Also it is possible to attach binary data:

```
>>> content = b'test content'
>>> email.attach_binary(content=content, filename='readme.txt')
```

Batches are available via the *EmailBatch()* constructor.

```
>>> batch = postmark.emails.EmailBatch(email)
>>> len(batch)
1
>>> batch.send()
```

For now, batches expose a very limited interface - only `send()` method and length information via the `len` function.

CHAPTER 3

Templates

The Templates API is available via the `templates` manager:

```
>>> template = postmark.templates.get(983381)
>>> template
<Template: Test (983381)>
>>> template.edit(Name='New name')
>>> postmark.templates.all()
[<Template: Test1 (983381)>, <Template: TestX (1003801)>]
```

Template validation:

```
>>> postmark.templates.validate(Subject='Test', TextBody='Test')
{
    'AllContentIsValid': True,
    'HtmlBody': None,
    'Subject': {
        'ContentIsValid': True,
        'RenderedContent': 'Test',
        'ValidationErrors': []
    },
    'SuggestedTemplateModel': {},
    'TextBody': {
        'ContentIsValid': True,
        'RenderedContent': 'Test',
        'ValidationErrors': []
    }
}
```

CHAPTER 4

Bounces

Information about bounces is available via the *BounceManager*, which is an attribute of `postmark` instance.

```
>>> postmark.bounces
<BouncesManager>
```

You can get a list of bounces with the *all()* method:

```
>>> postmark.bounces.all()
[<Bounce: 943247350>, <Bounce: 924829573>]
```

Default number of bounces, which will be returned is 500 to fit in single network request. More than 500 items will be fetching in multiple network requests:

```
>>> postmark.bounces.all(count=1001)
[<Bounce: 943247350>, <Bounce: 924829573>, ...]
```

Here it will be 3 network requests, single one is limited by 500 items. To load all available data, pass *None* as *count* value.

Every bounce instance is represented by its ID. You can search be specifying the ID.

```
>>> bounce = postmark.bounces.get(943247350)
>>> bounce
<Bounce: 943247350>
```

Bounce could be activated.

```
>>> bounce.activate()
'OK'
```

Or you can get an SMTP dump if it is available.

```
>>> bounce.dump
'A lot of text'
```

Also, you can access to corresponding `OutboundMessage` instance via `message` property.

```
>>> bounce.message
<Sent message to test@example.com>
```

CHAPTER 5

Inbound messages

The inbound messages API is available via the `messages.inbound` manager:

```
>>> postmark.messages.inbound.all()
[<InboundMessage: Blocked message from test@example.com>, ...]
```

There is a simple way to access the headers in the `InboundMessage` instance:

```
>>> inbound['X-Spam-Status']
No
```

Also, you can convert inbound message to MIME instance:

```
>>> inbound.as_mime()
<email.mime.multipart.MIMEMultipart object at 0x10a232c88>
```

5.1 Attachments

To access the attachments you could use the `Attachments` attribute of the `InboundMessage` which returns a list of attachments.

```
>>> inbound.Attachments
[<Attachment: test.txt>]
```

The interface is the same - all keys-value pairs in JSON data are attributes of the `Attachment`:

```
>>> attachment = inbound.Attachments[0]
>>> attachment.Name
test.txt
```

Besides it, you could do the following with attachments:

- Access the decoded content

- Check it length
- Save it locally
- Convert it to MIME instance

```
>>> len(attachment)
45
>>> attachment.decoded
This is attachment contents, base-64 encoded.
>>> attachment.save('/directory/to/save/')
/directory/to/save/test.txt
>>> attachment.as_mime()
<email.mime.base.MIMEBase at 0x10424b7b8>
```

CHAPTER 6

Outbound messages

The outbound messages API is available via the `messages.outbound` manager:

```
>>> postmark.messages.outbound.all()
[<OutboundMessage: Sent message to test@example.com>, ...]
```

CHAPTER 7

Opens

The opens API handler is located in `messages.outbound.opens` manager.

```
>>> postmark.messages.outbound.opens.all()
[<Open: Open from test@example.com>, ...]
```

Also, you can access to corresponding `OutboundMessage` instance via `message` property.

```
>>> open.message
<Sent message to test@example.com>
```

CHAPTER 8

Sender signatures

The Sender signatures API is available via the `senders` manager:

```
>>> sender_signature = postmark.senders.get(128462)
>>> sender_signature.delete()
```

CHAPTER 9

Server

The Server API is available via the `server` manager:

```
>>> server = postmark.server.get()
>>> server.edit(TrackOpens=True)
```

CHAPTER 10

Status

The Status API is available via the `status` manager:

```
>>> postmark.status.get()
{'lastCheckDate': '2016-10-09T11:05:04Z', 'status': 'UP'}
>>> postmark.status.services
[
    {'status': 'UP', 'url': '/services/api', 'name': 'API Response Time'},
    {'status': 'UP', 'url': '/services/smtp', 'name': 'SMTP Response Time'},
    {'status': 'UP', 'url': '/services/web', 'name': 'Web Response Time'},
    {'status': 'UP', 'url': '/services/inbound', 'name': 'Inbound SMTP Response Time'}
]
>>> postmark.status.availability
{
    'fromDate': '2016-07-11T11:47:10.371Z',
    'percentageUp': 0.9998308505688754,
    'secondsDown': 5280,
    'secondsUp': 31209723,
    'toDate': '2016-10-09T11:47:10.372Z'
}
>>> postmark.status.delivery
[
    {'missing': 0, 'spf': 1, 'spam': 0, 'inbox': 1, 'dkim': 1, 'name': 'AOL'},
    {'missing': 0, 'spf': 1, 'spam': 0, 'inbox': 1, 'dkim': 1, 'name': 'Apple'},
    {'missing': 0, 'spf': 1, 'spam': 0, 'inbox': 1, 'dkim': 1, 'name': 'Gmail'},
    {'missing': 0, 'spf': 1, 'spam': 0, 'inbox': 1, 'dkim': 1, 'name': 'Hotmail'},
    {'missing': 0, 'spf': 1, 'spam': 0, 'inbox': 1, 'dkim': 1, 'name': 'Yahoo!'}
]
```

Incidents are also available:

```
>>> postmark.incidents.last
<Incident: 1965>
>>> postmark.incidents.all()
[<Incident: 1965>, ...]
```

(continues on next page)

(continued from previous page)

```
>>> postmark.incidents.get(1965)
<Incident: 1965>
```

CHAPTER 11

Stats

The Stats API is available via the `stats` manager:

```
>>> postmark.stats.overview(fromdate='2017-02-01')
{
    'BounceRate': 0.0,
    'Bounced': 0,
    'Opens': 50,
    'SMTPApiErrors': 0,
    'Sent': 93,
    'SpamComplaints': 0,
    'SpamComplaintsRate': 0.0,
    'TotalClicks': 0,
    'TotalTrackedLinksSent': 0,
    'Tracked': 93,
    'UniqueLinksClicked': 0,
    'UniqueOpens': 19,
    'WithClientRecorded': 13,
    'WithLinkTracking': 0,
    'WithOpenTracking': 93,
    'WithPlatformRecorded': 19,
    'WithReadTimeRecorded': 19
}
>>> postmark.stats.sends(fromdate='2017-02-01')
{
    'Days': [
        {'Date': '2017-02-01', 'Sent': 2}
    ],
    'Sent': 2
}
```

CHAPTER 12

Inbound rules triggers

The Inbound rules triggers API is available via the `triggers.inboundrules` manager.

```
>>> postmark.triggers.inboundrules.all()
[<InboundRule: 962286>, ...]
```

CHAPTER 13

Tags triggers

The Tags triggers API is available via the `triggers.tags` manager.

```
>>> postmark.triggers.tags.all()
[<Tag: welcome>, ...]
```

CHAPTER 14

Domains

The Domains API is available via the `domains` manager:

```
>>> domain = postmark.domains.get(1)
>>> domain.edit(ReturnPathDomain='pmbounces.example.com')
```

CHAPTER 15

Django email backend

For convenience, Postmarker provides a Django email backend. To use it you have to update your project settings:

```
EMAIL_BACKEND = 'postmarker.django.EmailBackend'
POSTMARK = {
    'TOKEN': '<YOUR POSTMARK SERVER TOKEN>',
    'TEST_MODE': False,
    'VERBOSITY': 0,
}
```

That's it! For every supported Python version, the backend is tested on the latest Django release that supports given the Python version:

- Python 2.7, 3.4, 3.5, 3.6, PyPy - Django 1.10

Example:

```
>>> from django.core.mail import send_mail
>>> send_mail(
    'Subject here',
    'Here is the message.',
    'sender@example.com',
    ['receiver@example.com'],
    html_message='<html></html>'
)
```

Note! The `html_message` argument is available only on Django 1.7+. To use HTML content in Django < 1.7 you should use the `django.core.mail.messages.EmailMultiAlternatives` class directly.

For testing purposes, there is the `TEST_MODE` option. When it is set to `True` all interactions will be conducted with a special testing API token - `POSTMARK_API_TEST`.

To globally turn on `TrackOpens` feature, set `TRACK_OPENS` to `True`.

CHAPTER 16

Messages

To mark messages with tags you could use `postmarker.django.PostmarkEmailMessage` / `postmarker.django.PostmarkEmailMultiAlternatives` instead of `EmailMessage` / `EmailMultiAlternatives` from Django. Example:

```
>>> from postmarker.django import PostmarkEmailMessage
>>> PostmarkEmailMessage(
    'Subject', 'Body', 'sender@example.com', ['receiver@example.com'],
    tag='Test tag'
).send()
```

You can get the same feature for your own classes with `PostmarkEmailMixin`.

CHAPTER 17

Signals

There are three signals, which are emitted:

- `postmarker.django.pre_send` is emitted just before the send;
- `postmarker.django.post_send` is emitted just after the send;
- `postmarker.django.on_exception` is emitted in case of exception.

`pre_send` and `post_send` signals contain `messages` kwarg which contains all MIME messages, that are sent. `post_send` signal has `response` kwargs, which contains decoded response from Postmark. `on_exception` contains not prepared email instances in `raw_messages` kwarg and exception instance in `exception`.

CHAPTER 18

Tornado helper

There is a mixin for Tornado support. You could implement request handler like this:

```
from postmarker.tornado import PostmarkMixin
from tornado.web import RequestHandler


class EmailHandler(PostmarkMixin, RequestHandler):

    def post(self):
        # Awesome stuff here
        # ...
        # Send single email
        self.send(
            From='sender@example.com',
            To='receiver@example.com',
            Subject='Postmark test',
            HtmlBody='<html><body><strong>Hello</strong> dear Postmark user.</body></
↪html>'
        )
        # Send batch
        self.send_batch(
            {
                'From': 'sender@example.com',
                'To': 'receiver@example.com',
                'Subject': 'Postmark test',
                'HtmlBody': '<html><body><strong>Hello</strong> dear Postmark user.</
↪body></html>',
            },
            {
                'From': 'sender2@example.com',
                'To': 'receiver2@example.com',
                'Subject': 'Postmark test 2',
                'HtmlBody': '<html><body><strong>Hello</strong> dear Postmark user.</
↪body></html>',
```

(continues on next page)

(continued from previous page)

```
            }
        )
        # Or do whatever you want to do with postmark
        bounces = self.postmark_client.bounces.all()
```

To make it work, define `postmark_server_token` option:

```
from tornado.web import Application

app = Application(
    [
        (r'/send/', EmailHandler),
    ],
    postmark_server_token='YOUR_API_TOKEN'
)
```

And run your app! That's it. All possible options have the same name as in `PostmarkClient` prefixed with `postmark_`:

- `postmark_server_token`
- `postmark_account_token`
- `postmark_verbosity`
- `postmark_max_retries`
- `postmark_timeout`

CHAPTER 19

Spam check API

You can use Postmark's SpamAssassin filter API with a simple call:

```
>>> response = postmark.spamcheck('Raw email dump')
>>> print(response['report'])
pts rule name               description
---- ---------------------- --------------------------------------------------
 1.3 RDNS_NONE              Delivered to internal network by a host with no rDNS
 1.0 BODY_URI_ONLY          Message body is only a URI in one line of text or for
                            an image
 2.0 URI_ONLY_MSGID_MALF    URI only + malformed message ID
>>> print(response['score'])
3.8
```

CHAPTER 20

Webhooks

20.1 Inbound

20.1.1 Basic

The webhook data could automatically decoded from JSON with `postmark.messages.inbound.InboundMessage`.

```
>>> with open('/path/to/raw_content.json') as fd:
        data = fd.read()
>>> inbound = postmark.messages.inbound.InboundMessage(data)
```

It works in the same way as a regular `InboundMessage`.

20.2 Delivery

For delivery webhook there is a wrapper - `Delivery` with the regular model interface as in any other model (e.g. `InboundMessage`):

```
>>> from postmarker.models.emails import Delivery
>>> with open('/path/to/raw_content.json') as fd:
        data = fd.read()
>>> hook = Delivery.from_json(data)
>>> hook.Recipient
john@example.com
```

20.3 Open

Opens could be processed by `postmark.messages.outbound.opens.Open`:

```
>>> with open('/path/to/raw_content.json') as fd:
        data = fd.read()
>>> open = postmark.messages.outbound.opens.Open(data)
```

20.4 Bounce

For bounce webhook processing there is `Bounce` constructor in `bounces` manager. It constructs new `Bounce` instance from given JSON string.

```
>>> with open('/path/to/raw_content.json') as fd:
        data = fd.read()
>>> bounce = postmark.bounces.Bounce(data)
>>> bounce.activate()
'OK'
```

Another way to parse a bounce - use `Bounce.from_json` method:

```
>>> from postmarker.models.bounces import Bounce
>>> with open('/path/to/raw_content.json') as fd:
        data = fd.read()
>>> bounce = Bounce.from_json(data)
```

But in this case, there is no possibility to work with the bounce - only parsed data will be available.

CHAPTER 21

Testing

Postmarker provides two *pytest* fixtures:

- *postmark_request* - to mock all requests to Postmark API.
- *postmark_client* - an instance of `PostmarkClient` with *postmark_request* fixture applied.

CHAPTER 22

API Reference

22.1 Email

class postmarker.models.emails.**EmailManager**(*client*)

Sends emails via Postmark REST API.

Email(*From*, *To*, *Cc=None*, *Bcc=None*, *Subject=None*, *Tag=None*, *HtmlBody=None*, *TextBody=None*, *Metadata=None*, *ReplyTo=None*, *Headers=None*, *TrackOpens=None*, *TrackLinks='None'*, *Attachments=None*)

Constructs *Email* instance.

Returns *Email*

EmailBatch(**emails*)

Constructs *EmailBatch* instance.

Returns *EmailBatch*

EmailTemplate(*TemplateId*, *TemplateModel*, *From*, *To*, *TemplateAlias=None*, *Cc=None*, *Bcc=None*, *Subject=None*, *Tag=None*, *ReplyTo=None*, *Headers=None*, *TrackOpens=None*, *TrackLinks='None'*, *Attachments=None*, *InlineCss=True*)

Constructs *EmailTemplate* instance.

Returns *EmailTemplate*

send(*message=None*, *From=None*, *To=None*, *Cc=None*, *Bcc=None*, *Subject=None*, *Tag=None*, *HtmlBody=None*, *TextBody=None*, *Metadata=None*, *ReplyTo=None*, *Headers=None*, *TrackOpens=None*, *TrackLinks='None'*, *Attachments=None*)

Sends a single email.

Parameters

- **message** – *Email* or `email.mime.text.MIMEText` instance.
- **From** (*str*) – The sender email address.
- **To** (*str or list*) – Recipient's email address. Multiple recipients could be specified as a list or string with comma separated values.

- **Cc** (*str or list*) – Cc recipient's email address. Multiple Cc recipients could be specified as a list or string with comma separated values.
- **Bcc** (*str or list*) – Bcc recipient's email address. Multiple Bcc recipients could be specified as a list or string with comma separated values.
- **Subject** (*str*) – Email subject.
- **Tag** (*str*) – Email tag.
- **HtmlBody** (*str*) – HTML email message.
- **TextBody** (*str*) – Plain text email message.
- **ReplyTo** (*str*) – Reply To override email address.
- **Headers** (*dict*) – Dictionary of custom headers to include.
- **TrackOpens** (*bool*) – Activate open tracking for this email.
- **TrackLinks** (*str*) – Activate link tracking for links in the HTML or Text bodies of this email.
- **Attachments** (*list*) – List of attachments.

Returns Information about sent email.

Return type *dict*

send_batch(**emails*, ***extra*)
Sends an email batch.

Parameters

- **emails** – *Email* instances or dictionaries
- **extra** – dictionary with extra arguments for every message in the batch.

class postmarker.models.emails.**Email**(***kwargs*)

attach(**payloads*)
Appends given payloads to the current payload.

Parameters **payloads** (*dict*, *tuple*, *list*, *MIMEBase*) –

Returns None.

22.2 Bounce

class postmarker.models.bounces.**BounceManager**(*client*)
Encapsulates logic about bounces.

Bounce(*json*)
Constructs new Bounce instance from JSON-encoded string. Intended to use for bounce webhook processing.

Parameters **json** – *str*

Returns *Bounce*

activate(*id*)
Activates a bounce.

Parameters **id** (*int*) – Bounce ID.

Returns Activation result and bounce data.

Return type *dict*

all(*count=500*, *offset=0*, *type=None*, *inactive=None*, *emailFilter=None*, *tag=None*, *messageID=None*, *fromdate=None*, *todate=None*)
Returns many bounces.

Parameters

- **count** (*int*) – Number of bounces to return per request.
- **offset** (*int*) – Number of bounces to skip.
- **type** (*str*) – Filter by type of bounce.
- **inactive** (*bool*) – Filter by emails that were deactivated by Postmark due to the bounce.
- **emailFilter** (*str*) – Filter by email address.
- **tag** (*str*) – Filter by tag.
- **messageID** (*str*) – Filter by messageID.
- **fromdate** (*date*) – Filter messages starting from the date specified (inclusive).
- **todate** (*date*) – Filter messages up to the date specified (inclusive).

Returns A list of *Bounce* instances.

Return type *list*

deliverystats
Returns number of inactive emails and list of bounce types with total counts.

Return type *dict*

get(*id*)
Returns a single bounce.

Parameters **id** (*int*) – Bounce ID.

Return type *Bounce*

get_dump(*id*)
Gets an SMTP data dump.

Parameters **id** (*int*) – Bounce ID.

Returns A dump of SMTP data if it is available.

Return type *str* or *None*

model
alias of *Bounce*

tags
A list of tags that have generated bounces for a given server.

Return type *list*

class postmarker.models.bounces.**Bounce**(*manager=None*, ***kwargs*)
Bounce model.

activate()
Activates the bounce instance and updates it with the latest data.

Returns Activation status.

Return type *str*

dump
Gets SMTP data dump.

Returns Dump of SMTP data if it is available.

Return type *str* or *None*

22.3 Server

class postmarker.models.server.**ServerManager**(*client*)
Lets you get or edit details of the specific server.

model
alias of *Server*

class postmarker.models.server.**Server**(*manager=None*, ***kwargs*)

CHAPTER 23

Changelog

23.1 Unreleased

23.2 ‘0.13.0‘_ - 2018-11-25

23.2.1 Added

- Support for Python 3.7. #170
- Support for *Metadata* option. #168

23.2.2 Changed

- Stream logs to *sys.stdout* by default. #159

23.2.3 Removed

- Support for Python 2.6, 3.2 and 3.3.

23.3 0.12.2 - 2018-11-05

23.3.1 Changed

- Make *mock* package optional on Python 2. #158, #162

23.4 0.12.1 - 2018-11-05

23.4.1 Changed

- Better handling of exceptions that happen during response parsing. #163

23.5 0.12.0 - 2018-06-12

23.5.1 Added

- Support for *TemplateAlias*. #150

23.5.2 Fixed

- Processing of alternatives together with attachments. #148
- Processing of *message/rfc822* attachments.

23.6 0.11.3 - 2017-11-08

23.6.1 Added

- Ability to convert inbound messages to MIME instances. #90

23.6.2 Fixed

- Fix missed *mock* dependency for Python 2. #145

23.7 0.11.2 - 2017-05-14

23.7.1 Added

- Alternative instantiation method - `from_config`.

23.8 0.11.1 - 2017-05-10

23.8.1 Added

- Test helpers. #112

23.9 0.11.0 - 2017-05-02

23.9.1 Added

- `message` property for `Bounce`, `Delivery` and `Open` classes to access corresponding `OutboundMessage` instance. #119
- An ability to control timeout and retries behaviour. #82
- Signal for exceptions in Django backend. #126
- Tornado helper. #85

23.10 0.10.1 - 2017-04-03

23.10.1 Fixed

- Fix Bcc ignoring in Django backend. #135

23.11 0.10.0 - 2017-03-30

23.11.1 Added

- Short-circuit send of empty batches in Django backend. #123

23.11.2 Changed

- `OutboundMessageManager.get_details` and `InboundMessageManager.get_details` were methods were renamed to `get`. Now they returns `OutboundMessage` and `InboundMessage` instances respectively. #125
- Renamed `token` kwarg in *PostmarkClient* to `server_token`. #130

23.11.3 Fixed

- Fix counting of successfully sent messages in Django backend. #122
- Propagate API exceptions in Django backend. #128

23.12 0.9.2 - 2017-03-29

23.12.1 Fixed

- Remove stale files from the package.

23.13 0.9.1 - 2017-03-29

23.13.1 Fixed

- Fix packaging issue.

23.14 0.9.0 - 2017-03-28

23.14.1 Added

- Ability to load all items without specifying exact *count* value. #106
- Delivery webhook wrapper. #95
- Open webhook wrapper. #96
- Bounce webhook wrapper. #97

23.14.2 Changed

- `postmarker.webhooks.InboundWebhook` class was superseded by `postmark.messages.inbound.InboundMessage` constructor, which works in the same way.

23.14.3 Fixed

- Fix PyPI package display. #116

23.15 0.8.1 - 2017-03-15

23.15.1 Fixed

- Fix needless requests when *count* is more than number of available items. #107

23.16 0.8.0 - 2017-03-13

23.16.1 Added

- Ability to download more than 500 items. #70
- *pre_send* and *post_send* Django signals. #83
- Inbound rules triggers API. #75
- Tags triggers API. #74

23.16.2 Changed

- Output logs stream to default `sys.stderr`. #102

23.17 0.7.2 - 2017-03-11

23.17.1 Fixed

- Fix Django backend crash with attachments. #98

23.18 0.7.1 - 2017-03-10

23.18.1 Added

- *VERBOSITY* option to the Django backend. #92

23.19 0.7.0 - 2017-03-02

23.19.1 Added

- Stats API. (#72)
- Sender Signatures API. (#73)
- Messages API. (#71)
- Inbound webhook wrapper. (#87)

23.20 0.6.2 - 2017-01-02

23.20.1 Fixed

- Fix Unicode string handling on Python 2. #78

23.21 0.6.1 - 2017-01-01

23.21.1 Fixed

- Fix handling of *quoted-printable* payload. #76

23.22 0.6.0 - 2016-12-05

23.22.1 Added

- Link tracking support. #62
- Spam check API support. #57
- Inline images support. #52

- Domains API. #64

23.22.2 Changed

- Better exceptions handling. #50

23.23 0.5.3 - 2016-10-27

23.23.1 Added

- Tags for Django messages. #59

23.24 0.5.2 - 2016-10-27

23.24.1 Fixed

- Fix headers decoding. #60

23.25 0.5.1 - 2016-10-18

23.25.1 Fixed

- Fix invalid messages count in email batches. #55

23.25.2 Changed

- Better Django support. #51

23.26 0.5.0 - 2016-10-15

23.26.1 Added

- Status API. #39
- Custom user agent. #43
- Jython support. #13
- Handling more than 500 emails in batches. #46
- Templates API. #15

23.27 0.4.0 - 2016-10-09

23.27.1 Added

- Python 3.2 support. #38

23.27.2 Removed

- `ServerClient` & `AccountClient` were removed. #41

23.28 0.3.1 - 2016-10-08

23.28.1 Changed

- Move repo.

23.29 0.3.0 - 2016-10-07

23.29.1 Added

- Pass extra settings to Django backend. #29
- Testing feature for `Django` backend. #27
- Logging. #19
- Server API. #14
- Improved attachments support. #23
- Improved MIME messages support. #28

23.30 0.2.0 - 2016-10-07

23.30.1 Added

- Django email backend. #16
- Support for `MIMEText` sending. #25
- Batch emailing implementation. #12
- Ability to remove headers from email message. #24
- Improved attachments interface. #18
- Support for sending single email. #11

23.31 0.1.1 - 2016-10-05

23.31.1 Fixed

- Fix packaging issue

23.32 0.1.0 - 2016-10-05

- Initial release.

CHAPTER 24

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index