django-reportmail Documentation

Release 1.3

Hiroki KIYOHARA

September 29, 2015

Contents

1 Why django-reportmail **3**

2 Contents **5**

2.1 Installation 5
2.2 Let the hacking begin 6
2.3 Advanced topics 7
2.4 API documentation 9

3 Resources **13**

Python Module Index **15**

Welcome to django-reportmail's documentation. django-reportmail is a django library to send 'report' mail. Almost django management commands used as night batch processing, and then, administrators will want to know the result as mail. If you want to notice results of some django commands, stick with this doc and try django-reportmail.

I recommend that you get started with Installation and then head over to the Let the hacking begin.

CHAPTER 1

Why django-reportmail

Of Cause, you can emit logs and aggregate them by using some another applications like Sentry. But in some cases, you can't deploy them and you should send the report as mail.

A situation like that, django-reportmail will be really helpful for you.

CHAPTER 2

Contents

2.1 Installation

Won't you know about django-reportmail? Continue to read following documentation! If you do, you will learn the way to setup django-reportmail.

2.1.1 How to install

As always, you can install django-reportmail by using *pip*:

```
pip install django-reportmail
```

And then, you need to fix 2 parts of *settings.py*. First, Add a line 'reportmail' to INSTALLED_APPS to register this library for your project:

```
INSTALLED_APPS = (
    ...
    'reportmail',
)
```

And also you need to set 'ADMINS' settings. Because this library will send the report mail to 'ADMINS' on settings.

```
ADMINS = (
    ('Hiroki KIYOHARA', 'hirokiky@gmail.com'),
)

SERVER_EMAIL = 'noreply@example.com'
```

Internally, the reason of setting 'ADMINS' and 'SERVER_EMAIL' is that django-reportmail will send mail by calling *django.core.mail.mail_admins()*. For more detail, please check out the official documentation about *mail_admins*. https://docs.djangoproject.com/en/1.6/topics/email/#mail-admins

2.1.2 Requires

django-reportmail is guaranteed to work correctly on following environments.

Python:

- 2.7
- 3.3

- 3.4
- 3.5

Django:

- 1.6
- 1.7
- 1.8

2.1.3 All set

After setting up the project, you can head over to the Let the hacking begin documentation!

2.2 Let the hacking begin

2.2.1 Basic Usage

It's really easy and simple to use django-reportmail. Only thing you should do is decorating the *handle* method of Django's management command by *reportmail.command.apply_reporter()*:

```
import csv
from django.core.management.base import BaseCommand

from reportmail.command import apply_reporter

class Command(BaseCommand):
    @apply_reporter("Title")
    def handle(self, reporter, filepath, *args, **options):
        for i, l in enumerate(csv.DictReader(open(filepath))):
            reporter.append('Line {}: processed {l}'.format(i+1, l))
```

Then the *handle* method will take a *reportmail.reporter.Reporter* object after *self*. This reporter object is an interface to store messages which you want to notify to administrators.

2.2.2 Reporter

The reporter object provide two method *append()* and *extend()*.

If you want to store a line of message, use *reportmail.reporter.Reporter.append()*. This method will take a string object and store. And if you want to store multiple lines of message, use *reportmail.reporter.Reporter.extend()*. This method will take a list object of strings and store.

Something you should write is storing messages to reporter as same as logging. You will never write another messy codes.

2.2.3 The report

When the command ends, administrators will get report mail. By default, the mail will be like this:

```
Subject:
    Title
Body:
    Report of someapp.management.commands.some_of_your_command
    args: path/to/somecsv.csv
    options:

    result:
    Line1: processed {'somefield': 'somevalue0'}
    Line2: processed {'somefield': 'somevalue1'}
    Line3: processed {'somefield': 'somevalue2'}
    Line4: processed {'somefield': 'somevalue3'}
```

Notice that the subject of the mail is same value of the argument for *apply_reporter* decorator. Actually the first argument of it will used as subject of the mail.

The head of the body is the condition of management command. The first line is the name of command (module path of the command). The second line is command arguments. And the third line is command options.

After them, It is showing the result of command, which is actually strings you stored by calling *append()* or *extend()*.

If some unexpected error occurred while processing the command, *apply_reporter* will catch the error and report it and it's traceback.

2.2.4 Aborting report

If you want to abort to send any reports (mails), call *abort()* of *Reporter*. It's useful in case you don't want to get any messages.

```
class Command(BaseCommand):
    @apply_reporter("Title")
    def handle(self, reporter, *args, **options):
        ...
        reporter.abort()  # Nothing to say
```

2.2.5 Always in motion is the future...

You learned basic usage of django-reportmail. But sometimes it's not enough to address some sort of customising.

On the next, you can learn advanced topics like changing mail templates, or changing way to report. Let's continue Advanced topics.

2.3 Advanced topics

In this page, you can learn more about *django-reportmail*. I'll show you helpful topics when you should address some sort of customising especially, appearing at your job.

2.3.1 How to change the mail template

This section you can learn the way to change template used for rendering report mail. To change the template, you can simply apply the path to template as the 'template' argument for *apply_reporter()*:

```
from django.core.management.base import BaseCommand
from reportmail.command import apply_reporter

class Command(BaseCommand):
    @apply_reporter("Title", template='yourapp/dataimport_report.txt')
    def handle(self, reporter, *args, **options):
        pass
```

By default, it uses *reportmail/command_report.txt*.

This template will take these values as context:

- stored_text: list of messages you stored.
- args: arguments of command calling.
- options: option arguments of command calling and some value of enviroments.
- command: module path for this command.

If you want to add another value, you can simply set item for the reporter.context atribute:

```
>>> reporter.context['some_additional_value'] = 'Hi, there'
```

2.3.2 How to change the way to report

Sometimes you want to change the way to report instead of admin mails. To change, you can simply pass the *committer* argument for *apply_reporter()*. For instance, If you want the reporter to output result to standard output, you can use *reportmail.reporter.console_committer()* like this:

```
from reportmail.reporter import console_committer

>>> class Command(BaseCommand):
...     @apply_reporter("Title", committer=console_committer)
...     def handle(self, reporter, *args, **kwargs):
```

django-reportmail provides two committer functions from it's own:

- *reportmail.reporter.admin_mail_committer()*: sending as admin mail (default committer)
- *reportmail.reporter.manager_mail_committer()*: sending as manager mail
- *reportmail.reporter.console_committer()*: printing out to the standard output

Or, you can simply set a 'Committer' function to the `reportmail.reporter.Reporter.committer` attribute. The committer is the function which to get 'subject' and 'body' string as positional argument and cause some side-effects:

```
>>> def my_committer(subject, body):
...     print(subject)
...     print(body)
>>> reporter.committer = my_committer
```

Notice that the implementation of this *my_committer* function is actually same with *console_committer*. It's not so complex to create committers. Try it cheerfully if you want it.

2.3.3 Conclusion

You've already learned about django-reportmail good enough. If you need some reference for this linbrary, please refer API documentation. This will useful when you want to remind behaviors of each components.

If you've read whole documentation and have some questions or opinions, please raise a new issue at django-reportmail repository

2.4 API documentation

2.4.1 reportmail.command module

reportmail.command.**apply_reporter**(*subject*, *template='reportmail/command_report.txt'*, *committer=None*, *reporter_cls=<class 'reportmail.reporter.Reporter'>*, *additional_context=None*)

Adding a reporting feature for django command

You can use this as decorator for Command.handle. and decorated handle() will get admin mail reporter object after *self*:

```
@apply_reporter("Title of report", 'path/to/template.txt')
def handle(self, reporter, *args, **options):
    ...
```

By default, *apply_reporter* will use the *reportmail/command_report.txt* template. To change the template, you can put same name template.

This decorator provide these additional values for template as context:

- args: arguments of command calling.
- options: option arguments of command calling and some value of enviroments.
- command: module path for this command.

Notice that if the decorated command raises an exception, It will caught it to add the traceback to report mail. After added the error message, raised exception will be reraised.

Parameters

- **subject** – Title of report
- **template** – Template to use rendering
- **committer** – Committer function to be passed for the reporter.

2.4.2 reportmail.reporter module

A module for reporting.

Reporter(subject, template[, base_context, ...])	An object to store result messages and send messages by using committer.
console_committer(subject, body)	One of committers to send messages to standard output.
admin_mail_committer(subject, body)	One of committers to send messages to Admin Mails.
manager_mail_committer(subject, body)	One of committers to send messages to Manager Mails.

Committers is callable to make some side-effect telling result message for administrators. Internally Reporter uses Committer to tell messages. So committers are totally separated from reporters and reporter delegates the sending processing to committers.

class reportmail.reporter.**Reporter**(*subject*, *template*, *base_context=None*, *committer=None*)

An object to store result messages and send messages by using committer.

The API of Reporter is quite simple. You can store messages as same way as list, like this:

```
>>> reporter = Reporter()
>>> reporter.append("The first line")
>>> reporter.append("The second line")
>>> reporter.commit()
```

When the *commit()* method is called, stored messages will be sent to administrators. You can also use committer as a context manager. If you do, you won't need to call *commit()* method explicitly.

```
>>> with Reporter() as reporter:
>>>     reporter.append("The first line")
>>>     reporter.append("The second line")
```

This way is better and easier to read. so I recommend to use Reporter as context manager. Notice that the reporter won't handle exceptions by default. If you want reporter to catch exceptions and report about it, write the explicit code like this:

```
>>> import traceback
>>> with Reporter() as reporter:
>>>     try:
>>>         # do_something()
>>>         reporter.append("Success")
>>>     except Exception as e:
>>>         reporter.append(str(e) + traceback.format_exc())
>>>         raise
```

Parameters

- **subject** (*str*) – A subject of message. This value will be deliver for committer directly.
- **template** (*str*) – A string to specify a template to be used for build result message.
- **base_context** (*dict*) – Base context will be provided for the template. By default, empty dict will be used.
- **committer** (*callable*) – Committer function. By default, admin_mail_committer will be used.

abort()
Aborting commit of this reporter.

If this method is called, *self.commit()* will no longer send results.

refs https://github.com/hirokiky/django-reportmail/issues/7

append(*text*)
Storing a line of message

Parameters **text** (*str*) – A string of message to store

commit()
A interface to send the report

Internally, this method will call *self.committer* by passing *self.subject and result of 'self.render()*.

extend(*text_list*)
Storing some lines of messages

Parameters **text** (*list*) – A list of Some messages to store

render()
Rendering result by using stored messages

The context for template will contain messages you stored as 'stored_text' value. And also it contains values from *base_context* of constructing.

reportmail.reporter.**admin_mail_committer**(*subject*, *body*)
One of committers to send messages to Admin Mails.

This committer depends on django's django.core.mail.mail_admins. So you need to set 'ADMINS' of the settings file. Notice that thin committer will fail silently to avoid causing unexpected error while sending admin mails.

This committer will simply use the subject as mail subject, and use body as mail body.

reportmail.reporter.**console_committer**(*subject*, *body*)
One of committers to send messages to standard output.

This committer will simply output the message, separating subject and body by breaking.

reportmail.reporter.**manager_mail_committer**(*subject*, *body*)
One of committers to send messages to Manager Mails.

This committer depends on django's django.core.mail.mail_managers. So you need to set 'MANAGERS' of the settings file. Notice that thin committer will fail silently to avoid causing unexpected error while sending manager mails.

This committer will simply use the subject as mail subject, and use body as mail body.

CHAPTER 3

Resources

- Documentation
- Github
- PyPI

r

reportmail.command, 9
reportmail.reporter, 9

A

abort() (reportmail.reporter.Reporter method), 10
admin_mail_committer() (in module reportmail.reporter), 11
append() (reportmail.reporter.Reporter method), 10
apply_reporter() (in module reportmail.command), 9

C

commit() (reportmail.reporter.Reporter method), 10
console_committer() (in module reportmail.reporter), 11

E

extend() (reportmail.reporter.Reporter method), 10

M

manager_mail_committer() (in module reportmail.reporter), 11

R

render() (reportmail.reporter.Reporter method), 10
Reporter (class in reportmail.reporter), 9
reportmail.command (module), 9
reportmail.reporter (module), 9