yell Documentation

Release 0.3.2

Alen Mujezinovic

February 26, 2013

CONTENTS

Pluggable notifications for your Python apps.

yell is not a notification storage or delivery backend but a set of APIs that make it easy to add your own delivery mechanisms.

The full documentation is available here.

CHAPTER

ONE

USING NOTIFICATION DECORATORS

```
from yell import notify
from yell.decorators import notification

@notification(name = 'buffalo')
def buffalo_printer(message):
    print message

@notification(name = 'buffalo')
def buffalo_saver(message):
    save(message)

notify("buffalo", _("Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo"))
```

CHAPTER

TWO

USING NOTIFICATION CLASSES

```
from yell import Notification, notify

class Buffalo(Notification):
    name = "buffalo"
    message = _("Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo")

    def notify(self, *args, **kwargs):
        print self.message

class BuffaloEmail(Buffalo):
    def notify(self, *args, **kwargs):
        send_mail("Buffalo", self.message, 'buffalo@example.com', [kwargs.get('user').email])

class BuffaloDatabase(Buffalo):
    def notify(self, *args, **kwargs):
        BuffaloModel.objects.create(user = kwargs.get('user'))

# The default behaviour is to use every notification backend with the same
# name
notify("buffalo", user = User.objects.get(id=1))

# Only send emails
notify("buffalo", user = User.objects.get(id=1), backends = [BuffaloEmail])
```

CHAPTER

THREE

CHANGELOG

v0.2

- Made the API saner to use (*backwards incompatible*):
  - `yell.Yell` became `yell.Notification`
  - `yell.yell` became `yell.notify`
  - `yell.decorators.yelling` became `yell.decorators.notification`

CHAPTER

FOUR

YELL

Pluggable notifications for your Python apps.

yell is not a notification storage or delivery backend but a set of APIs that make it easy to add your own delivery mechanisms.

The full documentation is available here.

4.1 Using notification decorators

```
from yell import notify
from yell.decorators import notification

@notification(name = 'buffalo')
def buffalo_printer(message):
    print message

@notification(name = 'buffalo')
def buffalo_saver(message):
    save(message)

notify("buffalo", _("Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo"))
```

4.2 Using notification classes

```
from yell import Notification, notify

class Buffalo(Notification):
    name = "buffalo"
    message = _("Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo")

    def notify(self, *args, **kwargs):
        print self.message

class BuffaloEmail(Buffalo):
    def notify(self, *args, **kwargs):
        send_mail("Buffalo", self.message, 'buffalo@example.com', [kwargs.get('user').email])

class BuffaloDatabase(Buffalo):
    def notify(self, *args, **kwargs):
```

```
        BuffaloModel.objects.create(user = kwargs.get('user'))

# The default behaviour is to use every notification backend with the same
# name
notify("buffalo", user = User.objects.get(id=1))

# Only send emails
notify("buffalo", user = User.objects.get(id=1), backends = [BuffaloEmail])
```

4.3 Changelog

v0.3

- *backwards incompatible* Guessing the file extension with the `mimetypes` package proved to be inconsistent across systems. `TemplatedEmailBackend` now makes uses explicitly declared file extensions.

v0.2

- Made the API saner to use (*backwards incompatible*):
  - `yell.Yell` became `yell.Notification`
  - `yell.yell` became `yell.notify`
  - `yell.decorators.yelling` became `yell.decorators.notification`

CHAPTER

FIVE

YELL

Pluggable notifications for your Python apps.

yell is not a notification storage or delivery backend but a set of APIs that make it easy to add your own delivery mechanisms.

The full documentation is available here.

5.1 Using notification decorators

```
from yell import notify
from yell.decorators import notification

@notification(name = 'buffalo')
def buffalo_printer(message):
    print message

@notification(name = 'buffalo')
def buffalo_saver(message):
    save(message)

notify("buffalo", _("Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo"))
```

5.2 Using notification classes

```
from yell import Notification, notify

class Buffalo(Notification):
    name = "buffalo"
    message = _("Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo")

    def notify(self, *args, **kwargs):
        print self.message

class BuffaloEmail(Buffalo):
    def notify(self, *args, **kwargs):
        send_mail("Buffalo", self.message, 'buffalo@example.com', [kwargs.get('user').email])

class BuffaloDatabase(Buffalo):
    def notify(self, *args, **kwargs):
```

```
        BuffaloModel.objects.create(user = kwargs.get('user'))

# The default behaviour is to use every notification backend with the same
# name
notify("buffalo", user = User.objects.get(id=1))

# Only send emails
notify("buffalo", user = User.objects.get(id=1), backends = [BuffaloEmail])
```

5.3 Changelog

v0.3

- *backwards incompatible* Guessing the file extension with the `mimetypes` package proved to be inconsistent across systems. `TemplatedEmailBackend` now makes use of explicitly declared file extensions.

v0.2

- Made the API saner to use (*backwards incompatible*):
  - `yell.Yell` became `yell.Notification`
  - `yell.yell` became `yell.notify`
  - `yell.decorators.yelling` became `yell.decorators.notification`