django-notify Documentation

Release 0.1.0

Pete Wicken

Jun 27, 2017

Contents:

django-notify is a Django plug-in for automatically publishing AMQP events when models are modified.

Adding a notification event is as simple as adding an `@notify` decorator to your model class. By default, the decorator will publish an event for any `create`, `read`, `update` and `delete` (CRUD) action on your model.

```
@notify
class MyModel(models.Model):
    pass
```

This behaviour can be modified to selectively act upon any combination of CRUD events:

```
@notify(when='CD')
class MyModel(models.Model):
    pass
```

To specify which fields should be included in the AMQP body; add it to fields:

```
@notify(when='CRUD', fields=('pk', 'name'))
class MyModel(models.Model):
    name = models.CharField()
```

That's it! For further configuration options such as generating custom routing keys for your events, see the :doc: *concepts* page.

CHAPTER 1

Ideas

A place for future project ideas.

CHAPTER 2

Concepts

Custom defined dynamic routing keys

In order to get around redefining routing keys for each model every time you call `@notify`, or having one static global one used for everything, we have settings options to allow you to dynamically set them using basic string formatting.

For example; you can set this as your routing key format definition:

```
DJANGO_NOTIFY_ROUTING_KEY = 'model.{}.{}.{}'
```

Which is saying, always start the routing key with the string literal `model` and then delimit three additional bits of data with `.`.

To define the 'additional bits of data', we can use the following settings variable which will be used to format the routing key. The order of the items in the tuple dictate the order of interpolation:

```
DJANGO_NOTIFY_ROUTING_KEY_ARGS = ('name', 'author', 'pk')
```

The strings defined in here are class attributes of the class - So in Django that means model fields. Here is an example of what this would generate:

```
model.my_book_name.joe_bloggs.123
```

This is easy, but most people will want a little more structure and context for their routing keys. Introducing, routing key formatters! Out of the box, django-notify provides you with some pretty simple but useful formatters. Here is an example of using one:

```
from django_notify import formatters

DJANGO_NOTIFY_ROUTING_KEY_ARGS = (formatters.EventType, 'name', 'pk')
```

This will format into your routing key a contextual model event type:

```
model.create.my_book_name.123
```

Cool, but we can do better. You can define your own key formatter by inheriting from the `BaseFormatter`.

```
from django_notify import formatters

class ModificationTime(formatters.BaseFormatter):

    def __init__(self, **kwargs):
        super(ModificationTime, self).__init__(kwargs)

    def formatted_result(self):
         return str(kwargs['model']['modified_at'])
```

Another thing to note is, as the strings we pass into our routing key arguments are just class attributes, we can sneak in some arguments that access more fundamental object fields. For example, let's say I want the actual model class name in the routing key? Simple python always wins.

```
DJANGO_NOTIFY_ROUTING_KEY_ARGS = (formatters.EventType, '__class__.__name__', 'pk')
```

```
model.create.MyModel.123
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search