Django Localized Recurrence Documentation

Release 2.1.0

Erik Swanson

Sep 27, 2017

Contents

1 Table of Contents **3**
- 1.1 Installation . . . 3
- 1.2 Quickstart and Basic Usage . . . 3
- 1.3 Examples . . . 5
- 1.4 Code Documentation . . . 7
- 1.5 Contributing . . . 7

Python Module Index **9**

Localized Recurrences help you keep track of events that recur in your users local times.

CHAPTER 1

Table of Contents

Installation

Installation with Pip

Django Localized Recurrence is available on PyPi it can be installed using `pip`:

```
pip install django-localized-recurrence
```

Use with Django

To use Localized Recurrence with django, first be sure to install it and/or include it in your `requirements.txt` Then include `'localized_recurrence'` in `settings.INSTALLED_APPS`. After it is included in your installed apps, run:

```
./manage.py migrate localized_recurrence
```

if you are using South. Otherwise run:

```
./manage.py syncdb
```

Quickstart and Basic Usage

Django localized recurrence allows you to store a single instance of the `LocalizedRecurrence` model for an event that occurs regularly. Localized recurrences also automatically ensure that the events remain consistent for the users local times.

The basic usage of the library comes in two steps:

1. Create a recurrence to keep track of a recurring event.

2. When appropriate, check if the event is due to be acted on, and take the appropriate action.

Creating a recurrence

Creating a recurring event is as simple as creating an instance of `localized_recurrence.models.LocalizedRecurrence` using the standard `create` method of django model managers. The following is an example of a daily recurring event, at 3:00 PM in the user's local time in the Eastern United States.

```
from datetime import timedelta

from localized_recurrence.models import LocalizedRecurrence

LocalizedRecurrence.objects.create(
    interval='DAY',
    offset=timedelta(hours=15),
    timezone='US/Eastern',
)
```

Once a localized recurrence is created, it is simply a static object in the database. However, it comes with methods that make it extremely easy to know if the recurrence is due to be acted on.

Acting on a recurrence

Django localized-recurrence does not specify a method for checking when recurrences are due. The user of this app is in complete control of how these recurrences are to be checked. For example, they could be checked in the view code when a user loads a page, or in a celery beat task.

In order to support a broad range of use cases, localized-recurrence limits itself to two actions.

1. Checking when the event is next scheduled to recur.
2. Updating the event to have occured in this interval.

Acting on a single recurrence object

Given a `LocalizedRecurrence` object, called, say `my_daily_event`, checking when an object is next scheduled to recurr is as simple as checking the `next_scheduled` property of a recurrence instance, which stores the time, in UTC, of when it is next due.

```
if my_daily_event.next_scheduled < datetime.utcnow():
    # Process the event / Do stuff.
```

Then, once you are done processing the event, its schedule needs to be updated so that it will not be due to be scheduled until its interval has passed. This is as simple as calling the `update_schedule` method on the instance.

```
my_daily_event.update_schedule()
```

Calling this method updates the `next_scheduled` field on the model in a way that makes sure it will recur only at the appropriate time for its interval and timezone.

Acting on many recurrence objects

To find all the `LocalizedRecurrence` instance which are due we can use django's built in ORM tools to filter based on the current UTC time.

```
past_due = LocalizedRecurrence.objects.filter(next_scheduled__lte=datetime.utcnow())
```

Then, after taking whatever action goes along with an event, we need to update the database so that the types of checks we showed above will only return `True` in the next interval for the recurrence.

For a queryset, such as `past_due` above, this is as simple as:

```
past_due.update_schedule()
```

With that call, django-localized-recurrence takes care of any local time changes in the interval, and sets the `next_scheduled` field of each object to the time, in UTC, of the event, as the user would expect it for their local time.

Examples

A Calendar Event Example

In this example we create a basic Calendar event, which store recurring events. The benefits of using a localized recurrence in this way are two fold. First, that you don't have to store a separate entry for every time the event happens, only one localized recurrence describing how the event recurs. Second, the code for keeping track of the conversion between a user's local time and UTC, even across daylight savings time boundaries is automatically handled by the recurrence updates.

We start by defining a model with a foreign key to `LocalizedRecurrence`.

```
from django.contrib.auth.models import User
from django.db import models

from localized_recurrence import LocalizedRecurrence

class RecurringCalendarEvent(models.Model):
    user = models.ForeignKey(User)
    event_name = models.CharField(max_length=120)
    event_description = models.TextField()
    recurrence = models.ForeignKey(LocalizedRecurrence)

    objects = RecurringCalendarEventManager()
```

To go along with the event model, we create a manager that can create the localized recurrence and event at the same time.

```
class RecurringCalendarEventManager(models.Manager):
    def create_event(self, name, description, user, timezone, offset, interval):
        recurrence = LocalizedRecurrence.objects.create(
            interval=interval,
            offset=time,
            timezone=timezone
        )
        event = self.create(
            user=user,
            event_name=name,
            description=description,
            recurrence=recurrence
        )
        return event
```

Then, in a file `views.py` we can create two views. The first is a view that is intended to show a simple calendar but that first checks to see if there are any events that are due to be shown the user. It does this by filtering on the `next_scheduled` field of the associated `LocalizedRecurrence` objects.

```
from datetime import datetime

from django.shortcuts import redirect
from django.views.generic import TemplateView

class CalendarView(TemplateView):
    template_name = 'calendar/full_calendar.html'

    def get(self, request, *args, **kwargs):
        events_past_due = RecurringCalendarEvent.objects.filter(
            user=self.request.user,
            recurrence__next_scheduled__lte=datetime.utcnow()
        )
        if events_past_due.count() > 0:
            redirect('calendar.event_notification')
        else:
            return super(CalendarView, self).get(request, *args, **kwargs)
```

The second view (also assumed to be in the `views.py` file) is the view that displays any of the events that are past due. In this view, the `get_context_data` takes care of both passing the events to the template, but also updating the `LocalizedRecurrence` objects so that their `next_scheduled` fields are automatically set to the appropriate time in the future.

```
class CalendarNotification(TemplateView):
    template_name = 'calendar/event_notification.html'

    def get_context_data(self):
        context = super(CalendarNotification, self)
        events_past_due = RecurringCalendarEvent.objects.filter(
            user=self.request.user,
            recurrence__next_scheduled__lte=datetime.utcnow()
        )
        LocalizedRecurrence.objects.filter(
            id__in=[event.recurrence for event in events_past_due]
        ).update_schedule()
        context['events_past_due'] = events_past_due
        return context
```

Then all that's left is presenting this information in an attractive manner.

In this usage of the LocalizedRecurrence objects, checking the recurrences depends on the user actually visiting a page to hit the code path. It would also be possible to check if the recurrences are past due in a separate task, like the celery-beat scheduler.

Code Documentation

Models

Fields

class localized_recurrence.fields.**DurationField**(**args*, ***kwargs*)

A field to store durations of time with accuracy to the second.

A Duration Field will automatically convert between python timedelta objects and database integers.

Duration fields can be used to define fields in a django model

```
class Presentations(models.Model):
    length = DurationField()
    speaker = models.ForeignKey(User)
    location = models.ForeignKey(Location)
```

Given such a model, storing a duration in the database is as simple as passing in a `timedelta` object

```
>>> Presentations.objects.create(
...     length=datetime.timedelta(minutes=45),
...     speaker=User.objects.get(email='MrT@example.com'),
...     location=wrestle_mania_ring,
... )
```

The timedeltas are stored as seconds in the backend, so sub-second accuracy is lost with this field.

Contributing

Contributions and issues are most welcome! All issues and pull requests are handled through github on the ambitioninc repository. Also, please check for any existing issues before filing a new one. If you have a great idea but it involves big changes, please file a ticket before making a pull request! We want to make sure you don't spend your time coding something that might not fit the scope of the project.

Running the tests

To get the source source code and run the unit tests, run:

```
$ git clone git://github.com/ambitioninc/django-localized-recurrence.git
$ cd django-localized-recurrence
$ virtualenv env
$ . env/bin/activate
$ python setup.py install
$ coverage run setup.py test
$ coverage report
```

While 100% code coverage does not make a library bug-free, it significantly reduces the number of easily caught bugs! Please make sure coverage is at 100% before submitting a pull request.

Code Quality

For code quality, and style consistency please run flake8:

```
$ pip install flake8
$ flake8 .
```

This project treats all flake8 warnings as errors. They should be fixed before submitting a pull request.

Code Styling

Please arrange imports with the following style

```
# Standard library imports
import os

# Third party package imports
from mock import patch
from django.conf import settings

# Local package imports
from localized_recurrence.version import __version__
```

Please follow Google's python style guide wherever possible.

Building the docs

When in the project directory:

```
$ pip install -r requirements/docs.txt
$ python setup.py build_sphinx
$ open docs/build/html/index.html
```

Release Checklist

To create a new release, please go through each step of the following checklist:

- Bump version in localized_recurrence/version.py
- Git tag the version
- Upload to pypi:

  ```
  pip install wheel
  python setup.py sdist bdist_wheel upload
  ```

Vulnerability Reporting

For any security issues, please do NOT file an issue or pull request on github! Please contact security@ambition.com with the GPG key provided on Ambition's website.

Python Module Index

l

localized_recurrence.fields, 7

Index

D

DurationField (class in localized_recurrence.fields), 7

L

localized_recurrence.fields (module), 7