Swingtime Documentation

Release 0.2

David Krauth

Oct 05, 2017

Contents

CHAPTER 1

Table of Contents

Introduction

About Swingtime

Welcome

Swingtime is a Django application similar to stripped down version of iCal for Mac OS X or Google Calendar. Swingtime provides an *Event* model that act as metadata container for one or more *Occurrence* objects, which describe specific start and end times.

Swingtime relies heavily upon both the `datetime` standard library package and the `dateutil` package, featuring direct support for the `dateutil.rrule` interface to create occurrences.

A fairly simple example:

```
>>> from datetime import *
>>> from swingtime import models as swingtime
>>> et = swingtime.EventType.objects.create(abbr='work', label='Work Related Events')
>>> evt = swingtime.Event.objects.create(
...     title='New TPS Cover Sheet',
...     description='Kiss off, Lumbergh!',
...     event_type=et
... )
>>> evt.add_occurrences(datetime(2008,12,1,16), datetime(2008,12,1,16,15), count=5)
>>> for o in evt.occurrence_set.all():
...     print o
...
New TPS Cover Sheet: 2008-12-01T16:00:00
New TPS Cover Sheet: 2008-12-02T16:00:00
New TPS Cover Sheet: 2008-12-03T16:00:00
New TPS Cover Sheet: 2008-12-04T16:00:00
New TPS Cover Sheet: 2008-12-05T16:00:00
```

A bit more elaborate example, using the the convenience function *models.create_event()*:

```
>>> # pay day is the last Friday of the month at 5pm
>>> evt = swingtime.create_event(
...     'Pay day',
...     ('pay', 'Payroll'), # alternate means to add EventType on the fly
...     freq=rrule.MONTHLY,
...     byweekday=rrule.FR(-1),
...     until=datetime(2009,4,1),
...     start_time=datetime(2008,12,25,17)
... )
>>> for o in evt.occurrence_set.all():
...     print o
...
Pay day: 2008-12-26T17:00:00
Pay day: 2009-01-30T17:00:00
Pay day: 2009-02-27T17:00:00
Pay day: 2009-03-27T17:00:00
```

Features

- Support for adding complex event occurrences via `dateutil`
- Ready-made *MultipleOccurrenceForm* for handling complex input
- Daily, monthly, and annual view functions
- Grid-based daily view generator, complete with alternating or sequential *EventType* CSS-class handling
- Slightly better than average documentation, a few test cases, and commented code
- Active support (I have to eat my own dogfood)
- Built-in demo project / application

Requirements

- Django 1.0+
- Python 2.5+ (yeah, sorry, Swingtime won't work with 2.3 or 2.4, but I use 2.5+ exclusively; however, should anyone like to contribute the necessary backward compatible patches that don't hamstring any of 2.5+ features, I more than willing to include them).
- The `dateutil` package.

Get Swingtime

Options:

- Swingtime source code
- Gzipped tarball

Settings

Swingtime has it's settings module (`conf/swingtime_settings.py`) that simulates how each Django project's `setting.py` file functions. You can overwrite any or all of the configuration parameters described in

swingtime_settings by creating a file in your own project and referencing that file in your project settings using the name SWINGTIME_SETTINGS_MODULE.

For example, from the demo's configuration:

```
SWINGTIME_SETTINGS_MODULE = 'demo.swingtime_settings'
```

Demo

Swingtime comes with its own demo project and application. The demo is themed as a Karate studio's website and allows you see and interact with the Swingtime application.

A live demo can be found at http://swingtime.gorgeofeternalperil.com or run locally using the built-in Django development server.

Templates

Currently, Swingtime does not include any templates of its own. The demo project provides some sample templates to use as a guide or starting point.

Sample data

Within the Swingtime demo is an app named karate, which defines the custom management command loaddemo. This command will pre-populate your initial database with some events and occurrences based upon the current date and time.

Running the demo

If you've checked out from directly from the trunk directory or exploded the latest version tarball, you run the demo from anywhere by changing to the demo directory and running:

```
$ python manage.py loaddemo
$ python manage.py runserver
```

Todo List

- Add i18n support / translation support in demo
- Include a setup installer (I'm not doing easy_install!)
- Add weekly view
- Import and export .ics files
- Add *Note* support to demo
- Add more tests
- Port over to github

models — Swingtime Object Model Definitions

Functions

create_event

models.**create_event**(*title*, *event_type*[, *description*, *start_time*, *end_time*, *note*, ***rrule_params*])

Convenience function to create an *Event*, optionally create an *EventType*, and associated *Occurrence* instances. *Occurrence* creation rules match those for *Event.add_occurrences()*.

Returns the newly created *Event* instance.

Parameters

event_type can be either an *EventType* object or 2-tuple of `(abbreviation,label)`, from which an *EventType* is either created or retrieved.

description sets the event's description if not `None`

start_time will default to the current hour if `None`

end_time will default to `start_time` plus *swingtime_settings.DEFAULT_OCCURRENCE_DURATION* hour if `None`

note if not `None`, add a `Note` instance to the new event

rrule_params follows the `dateutils` API (see http://labix.org/python-dateutil)

Example:

```
from datetime import datetime, time
from swingtime import models as swingtime
from dateutil import rrule

event = swingtime.create_event(
    'Beginner Class',
    ('bgn', 'Beginner Classes'),
    description='Open to all beginners',
    start_time=datetime.combine(datetime.now().date(), time(19)),
    count=6,
    byweekday=(rrule.MO, rrule.WE, rrule.FR)
)
```

Classes

Note

class models.**Note**(*django.db.models.Model*)

A generic model for adding simple, arbitrary notes to other models such as `Event` or `Occurrence`.

note
models.TextField

created
models.DateTimeField

EventType

class models.**EventType**(*django.db.models.Model*)
Simple Event classifcation.

abbr
models.CharField

label
models.CharField

Event

class models.**Event**(*django.db.models.Model*)
Container model for general metadata and associated Occurrence entries.

title
models.CharField

description
models.CharField

event_type
models.ForeignKey for EventType

notes
generic.GenericRelation for Note

get_absolute_url()
return ('swingtime-event', [str(self.id)])

add_occurrences(*start_time*, *end_time*[, ***rrule_params*])
Add one or more occurences to the event using a comparable API to dateutil.rrule.

If rrule_params does not contain a freq, one will be defaulted to rrule.DAILY.

Because rrule.rrule returns an iterator that can essentially be unbounded, we need to slightly alter the expected behavior here in order to enforce a finite number of occurrence creation.

If both count and until entries are missing from rrule_params, only a single Occurrence instance will be created using the exact start_time and end_time values.

upcoming_occurrences()
Return all occurrences that are set to start on or after the current time.

next_occurrence()
Return the single occurrence set to start on or after the current time if available, otherwise None.

daily_occurrences([*dt*])
Convenience method wrapping Occurrence.objects.daily_occurrences.

OccurrenceManager

class models.**OccurrenceManager**(*models.Manager*)

daily_occurrences([*dt*, *event*])
Returns a queryset of for instances that have any overlap with a particular day.

Parameters

dt may be either a datetime.datetime, datetime.date object, or `None`. If `None`, default to the current day

event can be an `Event` instance for further filtering

Occurrence

class `models.`**Occurrence**(*django.db.models.Model*)

Represents the start end time for a specific occurrence of a master *Event* object.

start_time
models.DateTimeField

end_time
models.DateTimeField

event
models.ForeignKey - a non-editable Event object

notes
generic.GenericRelation `Note`

get_absolute_url()
'swingtime-occurrence', [str(self.event.id), str(self.id)])

__cmp__()
Compare two `Occurrence` start times

title
Shortcut for the occurrence's `Event.title`

event_type
Shortcut for the occurrence's `EventType`

views — Swingtime Views

Functions

event_listing

`views.`**event_listing**(*request*[, *template='swingtime/event_list.html'*, *events=None*, ***extra_context*])

View all `events`.

If `events` is a queryset, clone it. If `None` default to all `Event` objects.

Context parameters:

events an iterable of `Event` objects

extra_context extra variables passed to the template context

event_view

`views.`**event_view**(*request*, *pk*[, *template='swingtime/event_detail.html'*, *event_form_class=forms.EventForm*, *recurrence_form_class=forms.MultipleOccurrenceForm*])

View an `Event` instance and optionally update either the event or its occurrences.

Context parameters:

event the event keyed by `pk`

event_form a form object for updating the event

recurrence_form a form object for adding occurrences

occurrence_view

views.**occurrence_view**(*request*, *event_pk*, *pk*[, *template='swingtime/occurrence_detail.html'*, *form_class=forms.SingleOccurrenceForm*])

View a specific occurrence and optionally handle any updates.

Context parameters:

occurrence the occurrence object keyed by `pk`

form a form object for updating the occurrence

add_event

views.**add_event**(*request*[, *template='swingtime/add_event.html'*, *event_form_class=forms.EventForm*, *recurrence_form_class=forms.MultipleOccurrenceForm*])

Add a new `Event` instance and 1 or more associated `Occurrence` instancess.

Context parameters:

dtstart a datetime.datetime object representing the GET request value if present, otherwise None

event_form a form object for updating the event

recurrence_form a form object for adding occurrences

_datetime_view

views.**_datetime_view**(*request template*, *dt*[, *timeslot_factory=None*, *items=None*, *params=None*])

Build a time slot grid representation for the given datetime `dt`. See utils.create_timeslot_table documentation for items and params.

Context parameters:

day the specified datetime value (dt)

next_day day + 1 day

prev_day day - 1 day

timeslots time slot grid of (time, cells) rows

day_view

views.**day_view**(*request*, *year*, *month*, *day*[, *template='swingtime/daily_view.html'*, ***params*])

See documentation for function‘‘_datetime_view‘‘.

today_view

views.**today_view**(*request*[, *template='swingtime/daily_view.html'*, ***params*])
See documentation for function‘‘_datetime_view‘‘.

year_view

views.**year_view**(*request*, *year*[, *template='swingtime/yearly_view.html'*, *queryset=None*])
Context parameters:

year an integer value for the year in questin

next_year year + 1

last_year year - 1

by_month a sorted list of (month, occurrences) tuples where month is a datetime.datetime object for the first day of a month and occurrences is a (potentially empty) list of values for that month. Only months which have at least 1 occurrence is represented in the list

month_view

views.**month_view**(*request*, *year*, *month*[, *template='swingtime/monthly_view.html'*, *queryset=None*])
Render a tradional calendar grid view with temporal navigation variables.

Context parameters:

today the current datetime.datetime value

calendar a list of rows containing (day, items) cells, where day is the day of the month integer and items is a (potentially empty) list of occurrence for the day

this_month a datetime.datetime representing the first day of the month

next_month this_month + 1 month

last_month this_month - 1 month

forms — Swingtime Forms

Convenience forms for adding and updating `Event` and `Occurrence` objects.

Functions

timeslot_options

forms.**timeslot_options**([*interval=swingtime_settings.TIMESLOT_INTERVAL*,
start_time=swingtime_settings.TIMESLOT_START_TIME,
end_delta=swingtime_settings.TIMESLOT_END_TIME_DURATION,
fmt=swingtime_settings.TIMESLOT_TIME_FORMAT])
Create a list of time slot options for use in swingtime forms.

The list is comprised of 2-tuples containing a 24-hour time value and a 12-hour temporal representation of that offset.

timeslot_offset_options

forms.**timeslot_offset_options**([*interval=swingtime_settings.TIMESLOT_INTERVAL*, *start_time=swingtime_settings.TIMESLOT_START_TIME*, *end_delta=swingtime_settings.TIMESLOT_END_TIME_DURATION*, *fmt=swingtime_settings.TIMESLOT_TIME_FORMAT*])

Create a list of time slot options for use in swingtime forms.

The list is comprised of 2-tuples containing the number of seconds since the start of the day and a 12-hour temporal representation of that offset.

Data

default_timeslot_options

forms.**default_timeslot_options**
defaults to `timeslot_options()`

default_timeslot_offset_options

forms.**default_timeslot_offset_options**
defaults to `timeslot_offset_options()`

Classes

MultipleIntegerField

class forms.**MultipleIntegerField**(*django.forms.MultipleChoiceField*)

A form field for handling multiple integers.

def __init__(self, choices, size=None, label=None, widget=None):

if widget is None: widget = forms.SelectMultiple(attrs={'size' : size or len(choices)})

SplitDateTimeWidget

class forms.**SplitDateTimeWidget**(*django.forms.MultiWidget*)

A Widget that splits datetime input into a SelectDateWidget for dates and Select widget for times.

__init__(*attrs=None*)

uses widgets `SelectDateWidget` and `forms.Select(choices=default_timeslot_options`

MultipleOccurrenceForm

class forms.**MultipleOccurrenceForm**(*django.forms.Form*)

day

forms.DateField

start_time_delta

forms.IntegerField

end_time_delta
forms.IntegerField

repeats
forms.ChoiceField

count
forms.IntegerField

until
forms.DateField

freq
forms.IntegerField

interval
forms.IntegerField

week_days
MultipleIntegerField

month_ordinal
forms.IntegerField

month_ordinal_day
forms.IntegerField

each_month_day = MultipleIntegerField(

year_months
MultipleIntegerField

is_year_month_ordinal
forms.BooleanField(required=False)

year_month_ordinal
forms.IntegerField(widget=forms.Select(choices=ORDINAL))

year_month_ordinal_day
forms.IntegerField(widget=forms.Select(choices=WEEKDAY_LONG))

__init__(*[**args*, ***kws*]*)
if `initial` contains `dtstart` - a `datetime.datetime` instance - the appropriate unspecified `initial` will be defaulted for the form.

clean()
populates `cleaned_data` with `start_time` and `end_time` values

save(event):
Returns an `Event` object

EventForm

class `forms.`**EventForm**(*django.forms.ModelForm*)
A simple form for adding and updating Event attributes

SingleOccurrenceForm

class `forms.`**SingleOccurrenceForm**(*django.forms.ModelForm*)
A simple form for adding and updating single Occurrence attributes

utils — Swingtime Utilities

Common features and functions for swingtime

Functions

html_mark_safe

utils.**html_mark_safe**(*func*)
Decorator for functions return strings that should be treated as template safe.

time_delta_total_seconds

utils.**time_delta_total_seconds**(*time_delta*)
Calculate the total number of seconds represented by a `datetime.timedelta` object

month_boundaries

utils.**month_boundaries**([*dt=None*])
Return a 2-tuple containing the datetime instances for the first and last dates of the current month or using `dt` as a reference.

css_class_cycler

utils.**css_class_cycler**()
Return a dictionary keyed by *EventType* abbreviations, whose values are an iterable or cycle of CSS class names.

create_timeslot_table

utils.**create_timeslot_table**([*dt=None*, *items=None*, *start_time=swingtime_settings.TIMESLOT_START_TIME*, *end_time_delta=swingtime_settings.TIMESLOT_END_TIME_DURATION*, *time_delta=swingtime_settings.TIMESLOT_INTERVAL*, *min_columns=swingtime_settings.TIMESLOT_MIN_COLUMNS*, *css_class_cycles=css_class_cycler*, *proxy_class=DefaultOccurrenceProxy*])
Create a grid-like object representing a sequence of times (rows) and columns where cells are either empty or reference a wrapper object for event occasions that overlap a specific time slot.

Currently, there is an assumption that if an occurrence has a `start_time` that falls with the temporal scope of the grid, then that `start_time` will also match an interval in the sequence of the computed row entries.

dt a `datetime.datetime` instance or `None` to default to now

items a queryset or sequence of *Occurrence* instances. If `None`, default to the daily occurrences for `dt`

start_time a `datetime.time` instance, defaulting to *swingtime_settings.TIMESLOT_START_TIME*

end_time_delta a `datetime.timedelta` instance, defaulting to *swingtime_settings.TIMESLOT_END_TIME_DURATION*

time_delta a `datetime.timedelta` instance, defaulting to *swingtime_settings.TIMESLOT_INTERVAL*

min_column the minimum number of columns to show in the table, defaulting to *swingtime_settings.TIMESLOT_MIN_COLUMNS*

css_class_cycles if not `None`, a callable returning a dictionary keyed by desired `EventType` abbreviations with values that iterate over progressive CSS class names for the particular abbreviation; defaults to *css_class_cycler()*

proxy_class a wrapper class for accessing an `Occurrence` object, which should also expose `event_type` and `event_class` attrs, and handle the custom output via its __unicode__ method; defaults to *DefaultOccurrenceProxy*

Classes

BaseOccurrenceProxy

class `utils.`**BaseOccurrenceProxy**(*object*)

A simple wrapper class for handling the representational aspects of an *Occurrence* instance.

DefaultOccurrenceProxy

class `utils.`**DefaultOccurrenceProxy**(*BaseOccurrenceProxy*)

Through the `__unicode__` method, outputs a **safe** string anchor tag for the *Occurrence* instance, followed by simple token placeholders to represent additional slot fillings.

swingtime_settings — Configuration Settings

Settings

`swingtime_settings.`**TIMESLOT_TIME_FORMAT**

A "strftime" string for formatting start and end time selectors in forms. The default format is:

```
'%I:%M %p'
```

`swingtime_settings.`**TIMESLOT_INTERVAL**

Used for creating start and end time form selectors as well as time slot grids. Value should be `datetime.timedelta` value representing the incremental differences between temporal options. The default is:

```
``datetime.timedelta(minutes=15)``
```

`swingtime_settings.`**TIMESLOT_START_TIME**

A datetime.time value indicting the starting time for time slot grids and form selectors. The default is:

```
``datetime.time(9)``
```

`swingtime_settings.`**TIMESLOT_END_TIME_DURATION**

A `datetime.timedelta` value indicating the offset value from `TIMESLOT_START_TIME` for creating time slot grids and form selectors. The purpose for using a time delta is that it possible to span dates. For instance, one could have a starting time of 3pm (15:00) and wish to indicate a ending value 1:30am (01:30), in which case a value of `datetime.timedelta(hours=10.5)` could be specified to indicate that the 1:30 represents the following date's time and not the current date. Default:

```
``datetime.timedelta(hours=+8)``
```

swingtime_settings.**TIMESLOT_MIN_COLUMNS**
Indicates a minimum value (default: `4`) for the number grid columns to be shown in the time slot table.

swingtime_settings.**DEFAULT_OCCURRENCE_DURATION**
Indicate the default length in time for a new occurrence, specifed by using a `datetime.timedelta` object, defaulting to:

```
``datetime.timedelta(hours=+1)``
```

swingtime_settings.**CALENDAR_FIRST_WEEKDAY**
If not `None`, passed to `calendar.setfirstweekday` function. Default: `6`

Changes in Swingtime

Release 0.2 (Decemeber 18, 2008)

- First public release.

CHAPTER 2

Index

- genindex
- modindex
- search

Python Module Index

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M