ACP-Calendar Documentation

Release 1.6.0

Luis Carlos Berrocal

Oct 01, 2017

Contents

Contents:

CHAPTER 1

ACP-Calendar

Holiday calendar and date management for the Panama Canal. Includes Panama Canal holidays from 2006 to 2017.

Documentation

The full documentation is at http://django-acp-calendar.readthedocs.io/.

Requirements

Requires

- Python 3.4, 3.5 or 3.6
- Django 1.8.15, 1.9.10, 1.10.7 or 1.11

Quickstart

Install ACP-Calendar

```
$ pip install acp-calendar
```

Open your settings file and include acp_calendar and rest_framework to the THIRD_PARTY_APPS variable on your settings file.

The settings file

```
DJANGO_APPS = (
# Default Django apps:
'django.contrib.auth',
'django.contrib.contenttypes',
```

```
'django.contrib.sessions',
'django.contrib.sites',
'django.contrib.messages',
'django.contrib.staticfiles',

# Useful template tags:
# 'django.contrib.humanize',

# Admin
'django.contrib.admin',
)
THIRD_PARTY_APPS = (
    'crispy_forms',  # Form layouts
    'allauth',  # registration
    'allauth.account',  # registration
    'allauth.socialaccount',  # registration
    'rest_framework',
    'acp_calendar',
)

# Apps specific for this project go here.
LOCAL_APPS = (
    'acp_calendar_project.users',  # custom users app

    # Your stuff: custom apps go here
)

# See: https://docs.djangoproject.com/en/dev/ref/settings/#installed-apps
INSTALLED_APPS = DJANGO_APPS + THIRD_PARTY_APPS + LOCAL_APPS
```

Add the acp_calendar.urls to your urls file.

```
urlpatterns = [
    url(r'^$', TemplateView.as_view(template_name='pages/home.html'), name='home'),
    url(r'^about/$', TemplateView.as_view(template_name='pages/about.html'), name=
↪'about'),

    # Django Admin, use {% url 'admin:index' %}
    url(settings.ADMIN_URL, include(admin.site.urls)),

    # User management
    url(r'^users/', include('acp_calendar_project.users.urls', namespace='users')),
    url(r'^calendar/', include('acp_calendar.urls', namespace='calendar')),
    url(r'^accounts/', include('allauth.urls')),

    # Your stuff: custom urls includes go here


] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Features

Holidays

To get the working days for the Panama Canal between january 1st to january 31st 2016.

```
In [ 3 ]:  import datetime

In [ 4 ]:  start_date = datetime.date(2016, 1,1)

In [ 5 ]:  end_date = datetime.date(2016,1,31)

In [ 6 ]:  working_days = ACPHoliday.get_working_days(start_date, end_date)

In [ 7 ]:  print(working_days)
19
```

Fiscal Year

```
In [ 1 ]:   import datetime

In [ 2 ]:   from acp_calendar.models import FiscalYear

In [ 3 ]:   start_date = datetime.date(2015, 10,1)

In [ 4 ]:   fiscal_year = FiscalYear.create_from_date(start_date)

In [ 5 ]:   print(fiscal_year)
FY16

In [ 6 ]:   fiscal_year.start_date
Out[6]: datetime.date(2015, 10, 1)

In [ 7 ]:   fiscal_year.end_date
Out[7]: datetime.date(2016, 9, 30)
```

Calculator

To access the calculator go to http://<your_host>:<your_port>/calendar/calculator/

docs/images/calculator_01.png

To use the calculator your base.html must have:

- A javascript block at the end of the html
- jQuery (version 2.2.x)
- jQuery ui (version 1.12.x)

Virtual Environment

Use virtualenv to manage a virtual environment.

In a Mac use the following command to create the virtual environment.

```
$ python3 /usr/local/lib/python3.4/site-packages/virtualenv.py --no-site-packages acp_
↪calendar_env
```

Running Tests

Does the code actually work?

```
$ source acp_calendar_env/bin/activate
(acp_calendar_env) $ pip install -r requirements-test.txt
(acp_calendar_env) $ python runtests.py
```

Builds

We are using Travis for continuos integration https://travis-ci.org/luiscberrocal/django-acp-calendar/builds

For coverage we are using coveralls https://coveralls.io/github/luiscberrocal/django-acp-calendar

Run bumpversion

```
$ bumpversion minor
```

Instead of minor you could also use **major** o **patch** depending on the level of the release.

```
python setup.py sdist bdist_wheel

python setup.py register -r pypitest

python setup.py sdist upload -r pypitest
```

Check https://testpypi.python.org/pypi/acp-calendar/

```
python setup.py register -r pypi

python setup.py sdist upload -r pypi
```

Development

There is a project to use to develop and view the acp_calendar app it is at https://github.com/luiscberrocal/acp-calendar-dev-project

Credits

Tools used in rendering this package:

- Cookiecutter
- **'cookiecutter-pypackage'_**

CHAPTER 2

Installation

At the command line:

```
$ easy_install acp-calendar
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv acp-calendar
$ pip install acp-calendar
```

CHAPTER 3

Usage

To use ACP-Calendar in a project:

```
import acp_calendar
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/luiscberrocal/acp-calendar/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

ACP-Calendar could always use more documentation, whether as part of the official ACP-Calendar docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/luiscberrocal/acp-calendar/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *acp-calendar* for local development.

1. Fork the *acp-calendar* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/acp-calendar.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv acp-calendar
$ cd acp-calendar/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 acp_calendar tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/luiscberrocal/acp-calendar/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_acp_calendar
```

CHAPTER 5

Credits

Development Lead

- Luis Carlos Berrocal <luis.berrocal.1942@gmail.com>

Contributors

- Ewen Chou <ewenchou>

CHAPTER 6

History

0.2.2 (2016-04-10)

- First release on PyPI.

CHAPTER 7

API documentation

API documentation

acp_calendar.models module

The `FiscalYear` class

class acp_calendar.models.**FiscalYear**(*year*, ***kwargs*)

This class represents a Pancama Canal Fiscal year which start on October first of the previous year and ends on September 30th of the current year. For example fiscal year 2016 starts on October 1st, 2015 and ends September 30th 2016

static create_from_date(*cdate*, ***kwargs*)

Creates a Fiscal year object for a date.

Parameters

- **cdate** – Date or datetime object within the fiscal year
- **kwargs** – Same kwargs as for the constructor

Returns a FiscalYear object

static current_fiscal_year(***kwargs*)

Create a fiscal year object for current date :param kwargs: Same kwargs as for the constructor :return: FiscalYear object for current date

months_in_fiscal_year()

Gets a tuple of tuple containing the month number and the year of the months in the fiscal year.

Returns a tuple containing 12 tuples. Each tuple contains 2 integer, the first one is the month the

second one is the year.

The `HolidayType` model

class acp_calendar.models.**HolidayType**(**args*, ***kwargs*)
Model for Holiday type.

The `ACPHoliday` model

class acp_calendar.models.**ACPHoliday**(**args*, ***kwargs*)
Model for a non working day in the Panama Canal due to a holiday. This model contains all the logic for the working days calculations.

static convert_to_date(*study_date*)
Converts String to date to a date object.

Parameters **study_date** – date to be processed

Returns date object

static days_in_range_generator(*start_date*, *end_date*)
Creates a generator that contains all date dates between start_date and end_date

Parameters

- **start_date** – Start date in string or date
- **end_date** – End date in string or date

Returns A generator containing all dates in range

static get_working_days(*start_date*, *end_date*, ***kwargs*)
Calculates the amount of working days between start date and end date. It will calculate all days that are not saturday or sunday and then substract the holiday in the range if they exist.

Parameters

- **start_date** –
- **end_date** –
- **kwargs** –

Returns Number of working days between the star date and the end date

static get_working_days_for_month(*year*, *month*)
Calculate the amount of working days in a month

Parameters

- **year** – Year
- **month** – month

Returns Number of working days

static validate_dates(*start_date*, *end_date*)
Validates three rules: 1. End date is not before start date 2. End date cannot occur after oldest holiday in database 3. Start date cannot occur before the first holiday in database

Will raise an ACPCalendarException if one of these rules is broken.

Parameters

- **start_date** – Start date
- **end_date** – End date

static **working_delta**(*start_date*, *working_days*)
Calculates the date based on a start date and the number of working days in the future

Parameters

- **start_date** – Start date
- **working_days** – Number of woking days to the date we are interested

Returns Date that is n working days from start date.

acp_calendar.management.commands.acp_holidays module

The acp_holidays command

acp_calendar.views module

The CalendarView View

The Calculator View

Python Module Index

a

Index