radioco Documentation

Release 3.2.2

Iago Veloso Abalo

Jul 13, 2017

Contents

CHAPTER 1

Overview

RadioCo is a broadcasting radio recording scheduling system. RadioCo has been intended to provide a solution for a wide range of broadcast projects, from community to public and commercial stations.

Here are a few of the key features:

- designed to work with any web browser
- drag and drop scheduling calendar interface
- live shows can be recorded and published automatically
- complete authentication system (user accounts, groups, permissions)
- ...and much more

Note: This software is not for live streaming

CHAPTER 2

Release Notes

This document refers to version 3.2.2

CHAPTER 3

Table of contents

Installation Guide

The pages in this section of the documentation are designed to help you get started quickly and show how easy it is to work with RadioCo

The guides follow a logical progression and build on each other, so it's recommended to work through them in the order presented here.

Installing web application

To allow RadioCo to generate correct dates it's necessary to set the timezone variable:

1. Find your timezone in this list
2. Go to the config folder (radioco/configs/base) and create if not exists a local_settings.py in the same directory than settings.py
3. Add to the local settings the variable timezone, for example: TIME_ZONE = "Europe/Madrid"

Installing locally

This tutorial is written for Python 2.7 and Ubuntu 12.04 or later.

Ubuntu

The easiest way of installing the app is using Docker engine, follow the installation steps to install Docker.

Open a terminal and introduce the following commands:

```
sudo apt-get install git-core python-dev python-pip
```

Next, download the project and cd into it:

```
git clone https://github.com/iago1460/django-radio
cd django-radio
```

Install the python invoke library:

```
pip install invoke==0.14.0
```

Execute the next command to deploy the app in docker, this step take some time:

```
inv quickstart
```

Warning: If you have faced the error "ValueError: unknown locale: UTF-8" on MacOS X, execute:

export LC_ALL=en_US.UTF-8 export LANG=en_US.UTF-8

Testing

Let's verify your installation

Now that the server's running, visit http://127.0.0.1:8000/

Warning: Don't use this server in anything resembling a production environment.

Using RadioCo on production

The Internet is a hostile environment. Before deploying this project, you should take some time to review your settings, with security, performance, and operations in mind. Keep in mind this critical settings.

Locally

RadioCo provides a staging environment, safer than the previous one but still insecure, **use at your own risk**.

```
inv docker.setup -e staging
```

Now that the server's running, visit http://127.0.0.1/

To create a superuser you still can use management commands:

```
inv docker.manage -e staging -c "createsuperuser"
```

Installing recorder program

This tutorial is written for Python 2.7 and Ubuntu 12.04 or later.

Installing on Ubuntu

We'll get started by setting up our environment.

```
sudo apt-get install python-dev python-pip python-virtualenv git-core alsa-utils␣
↪vorbis-tools
```

Next, download the project and create the virual environment:

```
git clone https://github.com/iago1460/django-radio-recorder.git
cd django-radio-recorder
```

Create and activate a virtual env:

```
virtualenv venv
source venv/bin/activate
```

Install the requirements:

```
pip install -r requirements.txt
```

Using your favorite text editor, configure the `settings.ini` file

Launch the program

```
python main.py
```

Communication Setup

In your Admin interface go to **Podcast Configuration**, copy the **Recorder token** and put it into the Recorder Program settings:

```
token:8fdde6d703c05773084ea83e5ec2da62637666a0 #for example
```

Modify the **url** in your Recorder Program settings:

```
url:http://yourdomain:80/api/1/
```

Customize

Technical reference material.

RadioCo has a number of settings to configure its behaviour, please read this section carefully.

Application Setup

RadioCo can be adapted to your needs, you have three ways to do it. Editing the Global Settings section in the administration page using your browser or manually overriding the settings.py and overriding templates.

Global settings

Go to the admin section on your browser and edit the information available.

Global Configuration

In this section you can add information related to your site apart of the google analytics id.

Calendar Configuration

Settings related to the calendar, things like the first day of the week

Podcast Configuration

Settings related with the recorder, change here recording delays and get the Recorder token necessary for the recorder programme to work

Site

Change `example.com` with the fully qualified domain name associated with the website. Note that some RSS feed attributes will be incorrect if this value is not properly set up.

Templates

There is a empty folder called templates inside the radioco folder. You should override templates here, make sure to keep the relative path.

For example, to override the episode detail page copy the episode_detail.html file from radioco/apps/programmes/templates/programmes/episode_detail.html to radioco/templates/programmes/episode_detail.html

Settings.py

These settings are available in `settings.py`. Your settings should be in a `local_settings.py` file in the same directory as `settings.py`.

Warning: Your changes on settings should be in `local_settings.py` to avoid conflicts when update, create that file if it's necessary. Be aware that this file is excluded from Git.

USERNAME_RADIOCO_RECORDER

This specifies who is the user of the recorder program:

```
USERNAME_RADIOCO_RECORDER = 'RadioCo_Recorder'
```

Note: It's a good idea change this value for security reasons.

LANGUAGE_CODE

A string representing the language code for this installation. It provides a fallback language in case the user's preferred language can't be determined or is not supported by the website. More info:

```
LANGUAGE_CODE = 'en'
```

PROGRAMME_LANGUAGES

New in version 1.1

Default: A tuple of the following three languages.

This specifies which languages are available for language selection in your programmes:

```
gettext_noop = lambda s: s

PROGRAMME_LANGUAGES = (
    ('es', gettext_noop('Spanish')),
    ('en', gettext_noop('English')),
    ('gl', gettext_noop('Galician')),
)
```

You can see the current list of translated languages by looking in django/conf/global_settings.py (or view the online source).

Disqus

New in version 2.0

Default: Disabled by default.

Add comments to your site with Disqus. Create your account and get your API key.:

```
DISQUS_ENABLE = True
DISQUS_API_KEY = 'YOUR_API_KEY'
DISQUS_WEBSITE_SHORTNAME = 'YOUR_SHORTNAME'
```

Recorder Program Setup

The settings are available in your settings.ini.

API

Api documentation.

Browsable API

Experimental in version 3.0

RadioCo has a Web browsable API, go to http://127.0.0.1:8000/api/2/ in your browser to explore it.

Optionally these endpoints support filtering and ordering in the majority of the exposed fields.

Programmes

Programmes can be filter using after and before parameters as well as Transmissions.

Example query to get all available programmes on New Year's Eve order by name:

```
http://127.0.0.1:8000/api/2/programmes?after=2016-12-31&before=2016-12-31&
↪ordering=name
```

Transmissions

Transmissions are always ordered by date, the after and before parameters are required.

Example query:

```
http://127.0.0.1:8000/api/2/transmissions?after=2016-12-19&before=2016-12-26
```

Also is possible to request the dates in a specific timezone:

```
http://127.0.0.1:8000/api/2/transmissions?timezone=Europe%2FMadrid&after=2016-12-19&
↪before=2016-12-26
```

Finally, there is a endpoint to get the current transmission:

```
http://127.0.0.1:8000/api/2/transmissions/now
```

Radiocom API

New in version 3.1

Radiocom is an Android application for Community Media Stations. RadioCo has the Radiocom API under the following urls:

```
http://127.0.0.1:8000/api/2/radiocom/programmes
http://127.0.0.1:8000/api/2/radiocom/radiostation
http://127.0.0.1:8000/api/2/radiocom/transmissions
http://127.0.0.1:8000/api/2/radiocom/transmissions/now
```

Development & community

RadioCo is an open-source project, and relies on its community of users to keep getting better.

You don't need to be an expert developer to make a valuable contribution - all you need is a little knowledge of the system, and a willingness to follow the contribution guidelines.

Remember that contributions to the documentation are highly prized, and key to the success of the project. Any time and effort you are willing to contribute is greatly appreciated!

Branch policy

- **master**: this is the current stable release, the version released on PyPI.
- **develop**: this branch always reflects a state with the latest delivered development changes for the next release.
- **feature branches**: these are used to develop new features.

Contributing Translation

For translators we have a Transifex account where you can translate the .po files and don't need to install git to be able to contribute. All changes there will be automatically sent to the project.

Contributing Documentation

Perhaps considered "boring" by hard-core coders, documentation is sometimes even more important than code! This is what brings fresh blood to a project, and serves as a reference for old timers. On top of this, documentation is the one area where less technical people can help most - you just need to write semi-decent English. People need to understand you. We don't care about style or correctness.

Documentation should be:

- written using valid Sphinx/restructuredText syntax (see below for specifics) and the file extension should be `.rst`
- written in English (we have standardised on British spellings)
- accessible - you should assume the reader to be moderately familiar with Python and Django, but not anything else. Link to documentation of libraries you use, for example, even if they are "obvious" to you

Merging documentation is pretty fast and painless.

Except for the tiniest of change, we recommend that you test them before submitting.

Documentation markup

Sections

We use Python documentation conventions for section marking:

- `#` with overline, for parts
- `*` with overline, for chapters
- `=`, for sections
- `-`, for subsections
- `^`, for subsubsections
- `"`, for paragraphs

Inline markup

- **use backticks -** **``` ``settings.py`` ```** **- for:**
    - literals

    - filenames
    - names of fields and other items in the Admin interface:

- **use emphasis - `*Home*` around:**
    - the names of available options in the Admin
    - values in or of fields

References

Use absolute links to other documentation pages - `:doc:`/how_to/toolbar`` - rather than relative links - `:doc:`/../toolbar``. This makes it easier to run search-and-replaces when items are moved in the structure.

django-radio

manage module

radioco package

Subpackages

radioco.apps package

Subpackages

radioco.apps.api package

Subpackages

radioco.apps.api.tests package

Submodules

radioco.apps.api.tests.test_api module

class `radioco.apps.api.tests.test_api.`**`TestApi`**

Bases: *`radioco.apps.radioco.test_utils.TestDataMixin`*

`test_api``()`

class `radioco.apps.api.tests.test_api.`**`TestSerializers`**`(`*methodName='runTest'*`)`

Bases: *`radioco.apps.radioco.test_utils.TestDataMixin`*, `django.test.testcases.TestCase`

`test_episode``()`

`test_episode_programme``()`

`test_programme``()`

`test_programme_photo_url``()`

test_schedule()

test_transmission()

radioco.apps.api.tests.test_fullcalendar module

class radioco.apps.api.tests.test_fullcalendar.**TestFullCalendarApi**(*methodName='runTest'*)

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, rest_framework.test.APITestCase

setUp()

test_move_schedule(**args*, ***kwargs*)

test_move_schedule_in_tz(**args*, ***kwargs*)

test_move_schedule_with_schedules(**args*, ***kwargs*)

test_move_schedule_with_schedules_in_tz(**args*, ***kwargs*)

test_schedules_post(**args*, ***kwargs*)

test_schedules_post_in_tz(**args*, ***kwargs*)

radioco.apps.api.tests.test_programmes module

class radioco.apps.api.tests.test_programmes.**TestNotAllowedMethodsProgrammesAPI**(*methodName='run*

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, rest_framework.test.APITestCase

test_episodes_delete()

test_episodes_post()

test_episodes_put()

test_programmes_delete()

test_programmes_post()

test_programmes_put()

class radioco.apps.api.tests.test_programmes.**TestProgrammesAPI**(*methodName='runTest'*)

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, rest_framework.test.APITestCase

setUp()

test_episodes_get_all()

test_episodes_get_by_programme()

test_programmes_after()

test_programmes_before()

test_programmes_between()

test_programmes_get_all()

radioco.apps.api.tests.test_radiocom_api module

class radioco.apps.api.tests.test_radiocom_api.**TestProgramme**(*methodName='runTest'*)

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, rest_framework.test.APITestCase

setUp()

test_programme_in_list()

class radioco.apps.api.tests.test_radiocom_api.**TestTransmission**(*methodName='runTest'*)

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, rest_framework.test.APITestCase

test_transmission_now(**args*, ***keywargs*)

radioco.apps.api.tests.test_radiocom_api.**mock_now**()

radioco.apps.api.tests.test_recorder module

class radioco.apps.api.tests.test_recorder.**PodcastMock**

class radioco.apps.api.tests.test_recorder.**TestProgrammesAPI**(*methodName='runTest'*)

Bases: rest_framework.test.APITestCase

classmethod setUpTestData()

test_recording_schedules(**args*, ***keywargs*)

test_submit_recorder(**args*, ***keywargs*)

radioco.apps.api.tests.test_schedules module

class radioco.apps.api.tests.test_schedules.**TestRestrictedMethodsScheduleAPI**(*methodName='runTest*

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, rest_framework.test.APITestCase

test_schedules_delete()

test_schedules_post()

test_schedules_put()

test_transmissions_delete()

test_transmissions_post()

test_transmissions_put()

class radioco.apps.api.tests.test_schedules.**TestSchedulesAPI**(*methodName='runTest'*)

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, rest_framework.test.APITestCase

test_schedules_get_all()

test_schedules_get_by_calendar()

test_schedules_get_by_nonexisting_calendar()

test_schedules_get_by_nonexisting_programme()

test_schedules_get_by_nonexiting_type()

test_schedules_get_by_programme()

test_schedules_get_by_type()

class radioco.apps.api.tests.test_schedules.**TestTransmissionAPI**(*methodName='runTest'*)

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, rest_framework.test.APITestCase

test_incorrect_transmission_queries()

test_transmission_before(**args*, ***keywargs*)

test_transmission_list_filter_non_active_calendar(**args*, ***keywargs*)

test_transmission_now(**args*, ***keywargs*)

test_transmission_same_day()

test_transmissions_between(**args*, ***keywargs*)

test_transmissions_between_requesting_tz()

test_transmissions_filter_calendar_nonexistend()

radioco.apps.api.tests.test_schedules.**mock_now**()

Module contents

Submodules

radioco.apps.api.recorder_views module

radioco.apps.api.recorder_views.**check_recorder_program**(*user*)

radioco.apps.api.recorder_views.**recording_schedules**(**args*, ***kwargs*)

radioco.apps.api.recorder_views.**submit_recorder**(**args*, ***kwargs*)

radioco.apps.api.serializers module

class radioco.apps.api.serializers.**AbsoluteURLField**(*method_name=None*, *source=None*, *reverse_url=None*, ***kwargs*)

Bases: rest_framework.fields.URLField

A field similar to URLField but returning always the absolute url

For example:

class ExampleSerializer(self): episode_url = AbsoluteURLField() photo_url = AbsoluteURLField(source='photo.url', read_only=True) rss_url = AbsoluteURLField(source='slug', reverse_url='programmes:rss')

to_representation(*value*)

class radioco.apps.api.serializers.**DateTimeFieldTz**(*format=<class rest_framework.fields.empty>*, *input_formats=None*, *default_timezone=None*, **args*, ***kwargs*)

Bases: rest_framework.fields.DateTimeField

Field to display the datetime in the current timezone

to_representation(*date*)

class radioco.apps.api.serializers.**EpisodeSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta

fields = (‘title’, ‘programme’, ‘summary’, ‘issue_date’, ‘season’, ‘number_in_season’)

model

alias of Episode

class radioco.apps.api.serializers.**ProgrammeSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta

fields = (‘id’, ‘slug’, ‘name’, ‘synopsis’, ‘runtime’, ‘photo_url’, ‘rss_url’, ‘language’, ‘category’)

model

alias of Programme

class radioco.apps.api.serializers.**RadiocomConfigurationSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta

fields = (‘station_name’, ‘icon_url’, ‘big_icon_url’, ‘station_photos’, ‘history’, ‘latitude’, ‘longitude’, ‘news_rss’, ‘

model

alias of RadiocomConfiguration

RadiocomConfigurationSerializer.**get_facebook_url**(*obj*)

RadiocomConfigurationSerializer.**get_station_photos**(*obj*)

RadiocomConfigurationSerializer.**get_twitter_url**(*obj*)

class radioco.apps.api.serializers.**RadiocomProgrammeSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

class Meta

fields = (‘id’, ‘name’, ‘description’, ‘logo_url’, ‘rss_url’)

model

alias of Programme

class `radioco.apps.api.serializers.`**`RadiocomTransmissionSerializer`**(*instance=None*, *data=<class rest_framework.fields.empty>*, **kwargs)

Bases: `rest_framework.serializers.Serializer`

class `radioco.apps.api.serializers.`**`ScheduleSerializer`**(*instance=None*, *data=<class rest_framework.fields.empty>*, **kwargs)

Bases: `rest_framework.serializers.ModelSerializer`

class **`Meta`**

`fields` **= ('id', 'programme', 'calendar', 'start', 'runtime', 'title', 'type', 'source')**

`model`

alias of `Schedule`

`ScheduleSerializer.`**`get_title`**(*schedule*)

`ScheduleSerializer.`**`validate`**(*attrs*)

class `radioco.apps.api.serializers.`**`TransmissionSerializer`**(*instance=None*, *data=<class rest_framework.fields.empty>*, **kwargs)

Bases: `rest_framework.serializers.Serializer`

class `radioco.apps.api.serializers.`**`TransmissionSerializerLight`**(*instance=None*, *data=<class rest_framework.fields.empty>*, **kwargs)

Bases: `rest_framework.serializers.Serializer`

`validate`(*attrs*)

radioco.apps.api.urls module

radioco.apps.api.views module

class `radioco.apps.api.views.`**`EpisodeFilter`**(*args, **kwargs)

Bases: `rest_framework.filters.FilterSet`

class **`Meta`**

`fields` **= ('programme',)**

`model`

alias of `Episode`

`EpisodeFilter.`**`base_filters`** **= OrderedDict([(u'programme', <django_filters.filters.CharFilter object>)])**

`EpisodeFilter.`**`declared_filters`** **= OrderedDict([('programme', <django_filters.filters.CharFilter object>)])**

class `radioco.apps.api.views.`**`EpisodeViewSet`**(**kwargs)

Bases: `rest_framework.viewsets.ReadOnlyModelViewSet`

`filter_backends` **= (<class 'rest_framework.filters.DjangoFilterBackend'>, <class 'rest_framework.filters.OrderingFi**

filter_class
alias of *EpisodeFilter*

queryset

serializer_class
alias of EpisodeSerializer

suffix = None

class radioco.apps.api.views.**ProgrammeFilter**(**args*, ***kwargs*)
Bases: rest_framework.filters.FilterSet

class Meta

fields = ('name', 'category')

model
alias of Programme

ProgrammeFilter.**base_filters = OrderedDict([(u'name', <django_filters.filters.CharFilter object>), (u'category'**

ProgrammeFilter.**declared_filters = OrderedDict()**

class radioco.apps.api.views.**ProgrammeFilterForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*)
Bases: django.forms.forms.Form

base_fields = OrderedDict([('after', <django.forms.fields.DateField object>), ('before', <django.forms.fields.DateFiel

clean()

declared_fields = OrderedDict([('after', <django.forms.fields.DateField object>), ('before', <django.forms.fields.Dat

media

class radioco.apps.api.views.**ProgrammeViewSet**(***kwargs*)
Bases: rest_framework.viewsets.ModelViewSet

filter_backends = (<class 'rest_framework.filters.DjangoFilterBackend'>, <class 'rest_framework.filters.OrderingFi

filter_class
alias of *ProgrammeFilter*

list(*request*, **args*, ***kwargs*)

lookup_field = 'slug'

permission_classes = (<class 'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly'>,)

queryset

serializer_class
alias of ProgrammeSerializer

suffix = None

class radioco.apps.api.views.**RadiocomProgrammeViewSet**(***kwargs*)
Bases: *radioco.apps.api.views.ProgrammeViewSet*

serializer_class
alias of RadiocomProgrammeSerializer

suffix = None

class radioco.apps.api.views.**RadiocomStation**(***kwargs*)
Bases: rest_framework.viewsets.ReadOnlyModelViewSet

list(*request*, **args*, ***kwargs*)

serializer_class
alias of RadiocomConfigurationSerializer

suffix = None

class radioco.apps.api.views.**RadiocomTransmissionViewSet**(***kwargs*)
Bases: *radioco.apps.api.views.TransmissionViewSet*

serializer_class
alias of RadiocomTransmissionSerializer

suffix = None

class radioco.apps.api.views.**ScheduleFilter**(**args*, ***kwargs*)
Bases: rest_framework.filters.FilterSet

class **Meta**

fields = ('programme', 'calendar', 'type')

model
alias of Schedule

ScheduleFilter.**base_filters** = OrderedDict([(u'programme', <django_filters.filters.CharFilter object>), (u'caler

ScheduleFilter.**declared_filters** = OrderedDict([('programme', <django_filters.filters.CharFilter object>)])

class radioco.apps.api.views.**ScheduleViewSet**(***kwargs*)
Bases: rest_framework.viewsets.ModelViewSet

filter_backends = (<class 'rest_framework.filters.DjangoFilterBackend'>, <class 'rest_framework.filters.OrderingFi

filter_class
alias of *ScheduleFilter*

permission_classes = (<class 'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly'>,)

queryset

serializer_class
alias of ScheduleSerializer

suffix = None

class radioco.apps.api.views.**TimezoneForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*)
Bases: django.forms.forms.Form

base_fields = OrderedDict([('timezone', <django.forms.fields.ChoiceField object>)])

clean_timezone()

declared_fields = OrderedDict([('timezone', <django.forms.fields.ChoiceField object>)])

media

x = 'Zulu'

class `radioco.apps.api.views.`**`TransmissionForm`**(*data=None, files=None, auto_id=u'id_%s', prefix=None, initial=None, error_class=<class 'django.forms.utils.ErrorList'>, label_suffix=None, empty_permitted=False*)

Bases: *`radioco.apps.api.views.TimezoneForm`*

`base_fields` **= OrderedDict([('timezone', <django.forms.fields.ChoiceField object>), ('after', <django.forms.fields.Date**

`clean`()

`declared_fields` **= OrderedDict([('timezone', <django.forms.fields.ChoiceField object>), ('after', <django.forms.field**

`media`

class `radioco.apps.api.views.`**`TransmissionOperationViewSet`**(***kwargs*)

Bases: *`radioco.apps.api.viewsets.UpdateOnlyModelViewSet`*

`perform_update`(*serializer*)

`permission_classes` **= (<class 'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly'>,)**

`queryset`

`serializer_class`

alias of `TransmissionSerializerLight`

`suffix` **= None**

`update`(*request, *args, **kwargs*)

class `radioco.apps.api.views.`**`TransmissionViewSet`**(***kwargs*)

Bases: `rest_framework.viewsets.ReadOnlyModelViewSet`

`filter_backends` **= (<class 'rest_framework.filters.DjangoFilterBackend'>,)**

`filter_class`

alias of *`ScheduleFilter`*

`list`(*request, *args, **kwargs*)

`now`(*request*)

`queryset`

`serializer_class`

alias of `TransmissionSerializer`

`suffix` **= None**

radioco.apps.api.viewsets module

class `radioco.apps.api.viewsets.`**`UpdateOnlyModelViewSet`**(***kwargs*)

Bases: `rest_framework.mixins.UpdateModelMixin`, `rest_framework.viewsets.GenericViewSet`

A viewset that provides a update action.

Module contents

radioco.apps.global_settings package

Subpackages

radioco.apps.global_settings.migrations package

Submodules

radioco.apps.global_settings.migrations.0001_initial module

class `radioco.apps.global_settings.migrations.0001_initial.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` = []

`operations` = [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'scroll_time', <django.db.models.fi

radioco.apps.global_settings.migrations.0002_remove_calendarconfiguration_display_next_weeks module

class `radioco.apps.global_settings.migrations.0002_remove_calendarconfiguration_display_next_`

Bases: `django.db.migrations.migration.Migration`

`dependencies` = [(u'global_settings', u'0001_initial')]

`operations` = [<RemoveField name=u'display_next_weeks', model_name=u'calendarconfiguration'>]

radioco.apps.global_settings.migrations.0003_siteconfiguration_footer module

class `radioco.apps.global_settings.migrations.0003_siteconfiguration_footer.`**`Migration`**(*name*, *app_label*

Bases: `django.db.migrations.migration.Migration`

`dependencies` = [(u'global_settings', u'0002_remove_calendarconfiguration_display_next_weeks')]

`operations` = [<AddField field=<django.db.models.fields.TextField>, preserve_default=True, name=u'footer', model_na

radioco.apps.global_settings.migrations.0004_auto_20150606_1335 module

class `radioco.apps.global_settings.migrations.0004_auto_20150606_1335.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` = [(u'global_settings', u'0003_siteconfiguration_footer')]

`operations` = [<AddField field=<django.db.models.fields.TextField>, preserve_default=True, name=u'address', model_

radioco.apps.global_settings.migrations.0005_auto_20150606_1415 module

class `radioco.apps.global_settings.migrations.0005_auto_20150606_1415.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` = [(u'global_settings', u'0004_auto_20150606_1335')]

operations = [<RenameField new_name=u'about_footer', model_name=u'siteconfiguration', old_name=u'footer'>, <A

radioco.apps.global_settings.migrations.0006_auto_20160116_1509 module

class `radioco.apps.global_settings.migrations.0006_auto_20160116_1509.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'global_settings', u'0005_auto_20150606_1415')]**

`operations` **= [<AlterField field=<django.db.models.fields.CharField>, name=u'google_analytics_id', model_name=u'si**

radioco.apps.global_settings.migrations.0007__v3_0__calendarconfiguration_tweaks module

class `radioco.apps.global_settings.migrations.0007__v3_0__calendarconfiguration_tweaks.`**`Migrat`**

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'global_settings', u'0006_auto_20160116_1509'), (u'radioco', u'0001__v3_0__mysql_timezone')]**

`operations` **= [<RemoveField name=u'scroll_time', model_name=u'calendarconfiguration'>, <AddField field=<django.**

radioco.apps.global_settings.migrations.0008__v3_0__radiocomconfiguration module

class `radioco.apps.global_settings.migrations.0008__v3_0__radiocomconfiguration.`**`Migration`**(*nam* *app*

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'global_settings', u'0007__v3_0__calendarconfiguration_tweaks')]**

`operations` **= [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'station_name', <django.db.model**

Module contents

radioco.apps.global_settings.templatetags package

Submodules

radioco.apps.global_settings.templatetags.global_settings_extras module

`radioco.apps.global_settings.templatetags.global_settings_extras.`**`get_global_model`**(*model_path*)

Module contents

Submodules

radioco.apps.global_settings.admin module

class `radioco.apps.global_settings.admin.`**`PodcastConfigurationAdmin`**(*model*, *admin_site*)

Bases: *`radioco.apps.global_settings.admin.SingletonModelAdmin`*

media

readonly_fields = **['recorder_token']**

class radioco.apps.global_settings.admin.**SingletonModelAdmin**(*model*, *admin_site*)

Bases: django.contrib.admin.options.ModelAdmin

change_form_template = **'global_settings/admin_change_form.html'**

change_view(*request*, *object_id*, *form_url=''*, *extra_context=None*)

get_urls()

has_add_permission(*request*)

has_delete_permission(*request*, *obj=None*)

media

object_history_template = **'global_settings/admin_object_history.html'**

response_change(*request*, *obj*)

radioco.apps.global_settings.models module

class radioco.apps.global_settings.models.**CalendarConfiguration**(*id*, *slot_duration*, *first_day*, *min_time*, *max_time*)

Bases: *radioco.apps.global_settings.models.SingletonModel*

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

exception CalendarConfiguration.**MultipleObjectsReturned**

Bases: django.core.exceptions.MultipleObjectsReturned

CalendarConfiguration.**get_first_day_display**(**moreargs*, ***morekwargs*)

CalendarConfiguration.**objects** = **<radioco.apps.global_settings.models.SingletonModelManager object>**

class radioco.apps.global_settings.models.**PodcastConfiguration**(*id*, *url_source*, *start_delay*, *end_delay*, *next_events*)

Bases: *radioco.apps.global_settings.models.SingletonModel*

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

exception PodcastConfiguration.**MultipleObjectsReturned**

Bases: django.core.exceptions.MultipleObjectsReturned

PodcastConfiguration.**objects** = **<radioco.apps.global_settings.models.SingletonModelManager object>**

PodcastConfiguration.**recorder_token**

class `radioco.apps.global_settings.models.`**`RadiocomConfiguration`**`(`*id, station_name, icon_url, big_icon_url, history, latitude, longitude, news_rss, station_photos, stream_url*`)`

Bases: *`radioco.apps.global_settings.models.SingletonModel`*

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `RadiocomConfiguration.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

`RadiocomConfiguration.`**`objects`** **= <radioco.apps.global_settings.models.SingletonModelManager object>**

class `radioco.apps.global_settings.models.`**`SingletonModel`**`(`**args, **kwargs*`)`

Bases: `django.db.models.base.Model`

class **`Meta`**

`abstract` **= False**

`SingletonModel.`**`delete`**`(`**args, **kwargs*`)`

classmethod `SingletonModel.`**`get_global`**`()`

`SingletonModel.`**`objects`**

`SingletonModel.`**`save`**`(`**args, **kwargs*`)`

class `radioco.apps.global_settings.models.`**`SingletonModelManager`**

Bases: `django.db.models.manager.Manager`

`get``(`**args, **kwargs*`)`

class `radioco.apps.global_settings.models.`**`SiteConfiguration`**`(`*id, site_name, about_footer, more_about_us, google_analytics_id, address, twitter_address, facebook_address*`)`

Bases: *`radioco.apps.global_settings.models.SingletonModel`*

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `SiteConfiguration.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

`SiteConfiguration.`**`objects`** **= <radioco.apps.global_settings.models.SingletonModelManager object>**

Module contents

radioco.apps.programmes package

Subpackages

radioco.apps.programmes.migrations package

Submodules

radioco.apps.programmes.migrations.0001_initial module

class `radioco.apps.programmes.migrations.0001_initial.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'auth', u'__first__')]**

`operations` **= [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'title', <django.db.models.fields.Cl**

radioco.apps.programmes.migrations.0002_change_language_choices module

class `radioco.apps.programmes.migrations.0002_change_language_choices.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'programmes', u'0001_initial')]**

`operations` **= [<RunPython <function lowercase_language>>, <AlterField field=<django.db.models.fields.CharField>, r**

`radioco.apps.programmes.migrations.0002_change_language_choices.`**`lowercase_language`**(*apps*, *schema_ed*

radioco.apps.programmes.migrations.0003_change_textfield_to_richtextfield module

class `radioco.apps.programmes.migrations.0003_change_textfield_to_richtextfield.`**`Migration`**(*nam* *app*

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'programmes', u'0002_change_language_choices')]**

`operations` **= [<AlterField field=<ckeditor.fields.RichTextField>, preserve_default=True, name=u'summary', model_na**

radioco.apps.programmes.migrations.0004_change_photo_url module

class `radioco.apps.programmes.migrations.0004_change_photo_url.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'programmes', u'0003_change_textfield_to_richtextfield')]**

`operations` **= [<RunPython <function change_photo_url>>]**

`radioco.apps.programmes.migrations.0004_change_photo_url.`**`change_photo_url`**(*apps*, *schema_editor*)

radioco.apps.programmes.migrations.0005_auto_20150531_1734 module

class `radioco.apps.programmes.migrations.0005_auto_20150531_1734.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'programmes', u'0004_change_photo_url')]**

`operations` **= [<AlterField field=<django.db.models.fields.related.OneToOneField>, preserve_default=True, name=u'ep**

radioco.apps.programmes.migrations.0006_auto_20160104_2029 module

class `radioco.apps.programmes.migrations.0006_auto_20160104_2029.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'programmes', u'0005_auto_20150531_1734')]**

`operations` **= [<AlterField field=<django.db.models.fields.files.ImageField>, name=u'photo', model_name=u'programm**

radioco.apps.programmes.migrations.0007_change_default_image module

class `radioco.apps.programmes.migrations.0007_change_default_image.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'programmes', u'0006_auto_20160104_2029'), (u'users', u'0003_auto_20160104_2029')]**

`operations` **= [<RunPython <function change_programmes>>, <RunPython <function change_users>>]**

`radioco.apps.programmes.migrations.0007_change_default_image.`**`change_programmes`**(*apps*, *schema_editor*)

`radioco.apps.programmes.migrations.0007_change_default_image.`**`change_users`**(*apps*, *schema_editor*)

radioco.apps.programmes.migrations.0008_auto_20160116_1509 module

class `radioco.apps.programmes.migrations.0008_auto_20160116_1509.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'programmes', u'0007_change_default_image')]**

`operations` **= [<AlterField field=<django.db.models.fields.CharField>, name=u'language', model_name=u'programme**

radioco.apps.programmes.migrations.0009__v3_0__small_tweaks module

class `radioco.apps.programmes.migrations.0009__v3_0__small_tweaks.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'programmes', u'0008_auto_20160116_1509'), (u'radioco', u'0001__v3_0__mysql_timezone')]**

`operations` **= [<AlterField field=<django.db.models.fields.DateField>, name=u'end_date', model_name=u'programme'**

radioco.apps.programmes.migrations.0010__v3_2__convert_role module

class radioco.apps.programmes.migrations.0010__v3_2__convert_role.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [(u'programmes', u'0009__v3_0__small_tweaks')]

operations = [<AlterField field=<django.db.models.fields.CharField>, name=u'role', model_name=u'participant'>, <A

radioco.apps.programmes.migrations.0010__v3_2__convert_role.**convert_role_choice_to_value**(*ap sc*

radioco.apps.programmes.migrations.0011__v3_2__ensure_one_user_role module

class radioco.apps.programmes.migrations.0011__v3_2__ensure_one_user_role.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [(u'programmes', u'0010__v3_2__convert_role')]

operations = [<AlterUniqueTogether unique_together=set([(u'person', u'episode')]), name=u'participant'>, <AlterUni

Module contents

Submodules

radioco.apps.programmes.admin module

class radioco.apps.programmes.admin.**NonStaffEpisodeAdmin**(*model*, *admin_site*)

Bases: django.contrib.admin.options.ModelAdmin

fields = ['programme', 'title', 'summary', 'issue_date', 'season', 'number_in_season']

form

alias of *NonStaffEpisodeAdminForm*

formfield_for_foreignkey(*db_field*, *request=None*, ***kwargs*)

get_queryset(*request*)

get_readonly_fields(*request*, *obj=None*)

inlines = [<class 'radioco.apps.programmes.admin.NonStaffParticipantInline'>, <class 'radioco.apps.programmes.adm

list_display = ('__unicode__', 'season', 'number_in_season', 'issue_date', 'programme')

list_filter = ['issue_date', <class 'radioco.apps.programmes.admin.OwnEpisodeProgrammeListFilter'>, <class 'rad

list_select_related = True

media

ordering = ['-season', '-number_in_season']

save_formset(*request*, *form*, *formset*, *change*)

save_model(*request*, *obj*, *form*, *change*)

search_fields = ['programme__name']

class radioco.apps.programmes.admin.**NonStaffEpisodeAdminForm**(**args*, ***kwargs*)

Bases: django.forms.models.ModelForm

class Meta

fields = '__all__'

model

alias of Episode

NonStaffEpisodeAdminForm.**base_fields = OrderedDict([('title', <django.forms.fields.CharField object>), ('peo**

NonStaffEpisodeAdminForm.**clean_programme**()

NonStaffEpisodeAdminForm.**declared_fields = OrderedDict()**

NonStaffEpisodeAdminForm.**media**

class radioco.apps.programmes.admin.**NonStaffParticipantInline**(*parent_model*, *admin_site*)

Bases: django.contrib.admin.options.StackedInline

extra = 0

form

alias of *NonStaffParticipantInlineForm*

formset

alias of *NonStaffParticipantInlineFormset*

get_formset(*request*, *obj=None*, ***kwargs*)

get_queryset(*request*)

media

model

alias of Participant

class radioco.apps.programmes.admin.**NonStaffParticipantInlineForm**(**args*, ***kwargs*)

Bases: django.forms.models.ModelForm

base_fields = OrderedDict()

clean()

declared_fields = OrderedDict()

media

class radioco.apps.programmes.admin.**NonStaffParticipantInlineFormset**(*data=None*, *files=None*, *instance=None*, *save_as_new=False*, *prefix=None*, *queryset=None*, ***kwargs*)

Bases: django.forms.models.BaseInlineFormSet

class radioco.apps.programmes.admin.**NonStaffProgrammeAdmin**(*model*, *admin_site*)

Bases: django.contrib.admin.options.ModelAdmin

get_fieldsets (*request*, *obj=None*)

get_prepopulated_fields (*request*, *obj=None*)

get_queryset (*request*)

inlines = [<class 'radioco.apps.programmes.admin.NonStaffRoleInline'>]

list_display = ('name', '_runtime', 'current_season', 'category', 'start_date', 'end_date')

list_filter = ['_runtime', 'category']

media

save_formset (*request*, *form*, *formset*, *change*)

search_fields = ['name']

class `radioco.apps.programmes.admin.`**NonStaffRoleInline** (*parent_model*, *admin_site*)

Bases: `django.contrib.admin.options.StackedInline`

extra = 0

form

alias of *NonStaffRoleInlineForm*

formset

alias of *NonStaffRoleInlineFormset*

get_formset (*request*, *obj=None*, ***kwargs*)

get_queryset (*request*)

list_select_related = True

media

model

alias of `Role`

class `radioco.apps.programmes.admin.`**NonStaffRoleInlineForm** (**args*, ***kwargs*)

Bases: `django.forms.models.ModelForm`

base_fields = OrderedDict()

clean ()

Check unique together: person, role, programme

declared_fields = OrderedDict()

media

class `radioco.apps.programmes.admin.`**NonStaffRoleInlineFormset** (*data=None*, *files=None*, *instance=None*, *save_as_new=False*, *prefix=None*, *queryset=None*, ***kwargs*)

Bases: `django.forms.models.BaseInlineFormSet`

class `radioco.apps.programmes.admin.`**OwnEpisodeIssueDateListFilter** (*request*, *params*, *model*, *model_admin*)

Bases: `django.contrib.admin.filters.SimpleListFilter`

lookups (*request*, *model_admin*)

parameter_name = 'date'

queryset (*request*, *queryset*)

title = <django.utils.functional.__proxy__ object>

class `radioco.apps.programmes.admin.`**OwnEpisodeProgrammeListFilter** (*request*, *params*, *model*, *model_admin*)

Bases: `django.contrib.admin.filters.SimpleListFilter`

Check people in programmes besides episodes, better performance

lookups (*request*, *model_admin*)

parameter_name = 'programme'

queryset (*request*, *queryset*)

title = <django.utils.functional.__proxy__ object>

class `radioco.apps.programmes.admin.`**PodcastInline** (*parent_model*, *admin_site*)

Bases: `django.contrib.admin.options.StackedInline`

extra = 0

inline_classes = ('grp-collapse grp-open',)

media

model

alias of `Podcast`

radioco.apps.programmes.feeds module

class `radioco.apps.programmes.feeds.`**ProgrammeFeed**

Bases: `django.contrib.syndication.views.Feed`

description (*programme*)

feed_extra_kwargs (*programme*)

get_object (*request*, *slug*)

item_description (*podcast*)

item_enclosure_length (*podcast*)

item_enclosure_mime_type (*podcast*)

item_enclosure_url (*podcast*)

item_extra_kwargs (*podcast*)

item_keywords (*podcast*)

item_pubdate (*podcast*)

item_title (*podcast*)

items (*programme*)

link (*programme*)

title (*programme*)

class `radioco.apps.programmes.feeds.`**`RssProgrammeFeed`**

Bases: *`radioco.apps.programmes.feeds.ProgrammeFeed`*

`feed_type`

alias of *`iTunesFeed`*

`item_guid`(*podcast*)

class `radioco.apps.programmes.feeds.`**`iTunesFeed`**(*title*, *link*, *description*, *language=None*, *author_email=None*, *author_name=None*, *author_link=None*, *subtitle=None*, *categories=None*, *feed_url=None*, *feed_copyright=None*, *feed_guid=None*, *ttl=None*, ***kwargs*)

Bases: `django.utils.feedgenerator.Rss201rev2Feed`

`add_item_elements`(*handler*, *item*)

`add_root_elements`(*handler*)

`rss_attributes`()

radioco.apps.programmes.models module

class `radioco.apps.programmes.models.`**`Episode`**(*id*, *title*, *programme*, *summary*, *issue_date*, *season*, *number_in_season*)

Bases: `django.db.models.base.Model`

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `Episode.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

`Episode.`**`get_absolute_url`**()

`Episode.`**`objects`** **= <radioco.apps.programmes.models.EpisodeManager object>**

`Episode.`**`participant_set`**

`Episode.`**`people`**

`Episode.`**`podcast`**

`Episode.`**`programme`**

`Episode.`**`runtime`**

`Episode.`**`summary_text`**

class `radioco.apps.programmes.models.`**`EpisodeManager`**

Bases: `django.db.models.manager.Manager`

`create_episode`(*date*, *programme*, *last_episode=None*, *episode=None*)

static **`last`**(*programme*)

static **`unfinished`**(*programme*, *after=None*)

class `radioco.apps.programmes.models.`**`Participant`**(*id*, *person*, *episode*, *role*, *description*)

Bases: `django.db.models.base.Model`

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

exception Participant.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Participant.**episode**

Participant.**objects = <django.db.models.manager.Manager object>**

Participant.**person**

class radioco.apps.programmes.models.**Podcast**(*episode*, *url*, *mime_type*, *length*, *duration*)
Bases: django.db.models.base.Model

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Podcast.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Podcast.**episode**

Podcast.**get_absolute_url**()

Podcast.**objects = <django.db.models.manager.Manager object>**

class radioco.apps.programmes.models.**Programme**(*id*, *name*, *synopsis*, *photo*, *language*, *current_season*, *category*, *slug*, *_runtime*, *start_date*, *end_date*)
Bases: django.db.models.base.Model

CATEGORY_CHOICES = ((‘Arts’, <django.utils.functional.__proxy__ object>), (‘Business’, <django.utils.functional.__pro

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Programme.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Programme.**announcers**

Programme.**end_dt**

Programme.**episode_set**

Programme.**get_absolute_url**()

Programme.**get_category_display**(**moreargs*, ***morekwargs*)

Programme.**get_language_display**(**moreargs*, ***morekwargs*)

Programme.**objects = <django.db.models.manager.Manager object>**

Programme.**photo**
Just like the FileDescriptor, but for ImageFields. The only difference is assigning the width/height to the width_field/height_field, if appropriate.

Programme.**rearrange_episodes**(*after*, *calendar*)
Update the issue_date of episodes from a given date

Programme.**role_set**

Programme.**runtime**

Programme.**save**(**args*, ***kwargs*)

Programme.**schedule_set**

Programme.**start_dt**

Programme.**synopsis_text**

class radioco.apps.programmes.models.**Role**(*id*, *person*, *programme*, *role*, *description*, *date_joined*)

Bases: django.db.models.base.Model

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

exception Role.**MultipleObjectsReturned**

Bases: django.core.exceptions.MultipleObjectsReturned

Role.**get_next_by_date_joined**(**moreargs*, ***morekwargs*)

Role.**get_previous_by_date_joined**(**moreargs*, ***morekwargs*)

Role.**objects = <django.db.models.manager.Manager object>**

Role.**person**

Role.**programme**

radioco.apps.programmes.models.**update_schedule_if_dt_has_changed**(*sender*, *instance*, ***kwargs*)

radioco.apps.programmes.models.**update_schedule_performance**(*programme*)

radioco.apps.programmes.urls module

radioco.apps.programmes.utils module

radioco.apps.programmes.views module

radioco.apps.programmes.views.**episode_detail**(*request*, *slug*, *season_number*, *episode_number*)

radioco.apps.programmes.views.**programme_detail**(*request*, *slug*)

Module contents

radioco.apps.radioco package

Subpackages

radioco.apps.radioco.management package

Subpackages

radioco.apps.radioco.management.commands package

Submodules

radioco.apps.radioco.management.commands.create_example_data module

class `radioco.apps.radioco.management.commands.create_example_data.`**`Command`**(*stdout=None*, *stderr=None*, *no_color=False*)

Bases: `django.core.management.base.BaseCommand`

`handle`(**args*, ***options*)

Module contents

Module contents

radioco.apps.radioco.migrations package

Submodules

radioco.apps.radioco.migrations.0001__v3_0__mysql_timezone module

class `radioco.apps.radioco.migrations.0001__v3_0__mysql_timezone.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u’schedules’, u‘0001_initial’), (u’programmes’, u‘0008_auto_20160116_1509’), (u’global_settings’, u‘**

`operations` **= [<RunPython <function migrate_datetime_to_utc>>]**

`radioco.apps.radioco.migrations.0001__v3_0__mysql_timezone.`**`migrate_datetime_to_utc`**(*apps*, *schema_ed*

version 3 introduces timezone support, this migration will fix the Episode issue_date Since Postgres is the only database which is already storing the datetimes in utc we don’t need to run on it

Module contents

radioco.apps.radioco.tests package

Submodules

radioco.apps.radioco.tests.tests module

class `radioco.apps.radioco.tests.tests.`**`RadioIntegrationTests`**(*methodName=’runTest’*)

Bases: *`radioco.apps.radioco.test_utils.TestDataMixin`*, `django.test.testcases.TestCase`

`test_index`()

class `radioco.apps.radioco.tests.tests.`**`UtilsTest`**(*methodName=’runTest’*)

Bases: `django.test.testcases.TestCase`

`test_dictionary_key`()

Making sure that is safe to use a dt tz aware object in a different timezone to access a dictionary

`test_example_data`()

Running example data function, If nothing crash we are happy

Module contents

Submodules

radioco.apps.radioco.context_processors module

`radioco.apps.radioco.context_processors.`**`settings`**(*request*)

radioco.apps.radioco.test_utils module

class `radioco.apps.radioco.test_utils.`**`TestDataMixin`**

Bases: `object`

classmethod **`setUpTestData`**()

`radioco.apps.radioco.test_utils.`**`create_test_data`**()

`radioco.apps.radioco.test_utils.`**`spain_tz`**()

radioco.apps.radioco.tz_utils module

class `radioco.apps.radioco.tz_utils.`**`GMT`**(*seconds*)

Bases: `dateutil.tz.tz.tzoffset`

GMT implementation, it has a fixed offset

`localize`(*dt*, *is_dst=False*)

Convert naive time to local time

`normalize`(*dt*, *is_dst=False*)

Correct the timezone information on the given datetime

`radioco.apps.radioco.tz_utils.`**`fix_recurrence_date`**(*start_dt*, *dt*)

Fix for django-recurrence 1.3 rdates and exdates needs a datetime, we are combining the date with the time from start_date.

Return: A datetime in the default timezone with the offset required to work in the recurrence

`radioco.apps.radioco.tz_utils.`**`fix_recurrence_dst`**(*dt*)

Fix for django-recurrence 1.3 Function to fix a datetime tz aware with an incorrect offset

Returns: A datetime in the same timezone but with the offset fixed

`radioco.apps.radioco.tz_utils.`**`get_active_timezone`**()

Same method as timezone.get_current_timezone but returning utc if nothing was set

`radioco.apps.radioco.tz_utils.`**`recurrence_after`**(*recurrence*, *after_dt*, *start_dt*)

Fix for django-recurrence 1.3 Avoid outputting a impossible dt

`radioco.apps.radioco.tz_utils.`**`recurrence_before`**(*recurrence*, *before_dt*, *start_dt*)

Fix for django-recurrence 1.3 Avoid outputting a impossible dt

`radioco.apps.radioco.tz_utils.`**`transform_datetime_tz`**(*dt*, *tz=None*)

Transform a datetime in other timezone to the current one

`radioco.apps.radioco.tz_utils.`**`transform_dt_to_default_tz`**(*dt*)

Transform a datetime in other timezone to the current one

radioco.apps.radioco.utils module

class `radioco.apps.radioco.utils.`**`DeletePermissionMixin`**

Bases: `object`

`dispatch`(*request*, **args*, ***kwargs*)

`model` = **None**

class `radioco.apps.radioco.utils.`**`GetObjectMixin`**

Bases: `django.views.generic.detail.SingleObjectMixin`

`dispatch`(**args*, ***kwargs*)

`object` = **None**

`radioco.apps.radioco.utils.`**`check_delete_permission`**(*user*, *model*)

`radioco.apps.radioco.utils.`**`create_example_data`**()

`radioco.apps.radioco.utils.`**`field_has_changed`**(*_object*, *field*)

class `radioco.apps.radioco.utils.`**`memorize`**(*func*)

Bases: `dict`

A simple cache system, use as decorator

radioco.apps.radioco.views module

`radioco.apps.radioco.views.`**`index`**(*request*)

`radioco.apps.radioco.views.`**`user_logout`**(*request*)

Module contents

radioco.apps.schedules package

Subpackages

radioco.apps.schedules.migrations package

Submodules

radioco.apps.schedules.migrations.0001_initial module

class `radioco.apps.schedules.migrations.0001_initial.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` = **[(u'programmes', u'0001_initial')]**

`operations` = **[<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'day', <django.db.models.fields.In**

radioco.apps.schedules.migrations.0002__v3_0__renaming_calendar_model module

class `radioco.apps.schedules.migrations.0002__v3_0__renaming_calendar_model.`**`Migration`**(*name*, *app_label*

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'schedules', u'0001_initial'), (u'radioco', u'0001__v3_0__mysql_timezone')]**

`operations` **= [<RenameField new_name=u'calendar', model_name=u'schedule', old_name=u'schedule_board'>, <Ren**

`radioco.apps.schedules.migrations.0002__v3_0__renaming_calendar_model.`**`migrate_board`**(*apps*, *schema_e*

Renaming previous Calendars and creating a new one

radioco.apps.schedules.migrations.0003__v3_0__create_excludeddates_model module

class `radioco.apps.schedules.migrations.0003__v3_0__create_excludeddates_model.`**`Migration`**(*name* *app_l*

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'schedules', u'0002__v3_0__renaming_calendar_model')]**

`operations` **= [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'datetime', <django.db.models.fiel**

radioco.apps.schedules.migrations.0004__v3_0__adding_rrules module

class `radioco.apps.schedules.migrations.0004__v3_0__adding_rrules.`**`Migration`**(*name*, *app_label*)

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'schedules', u'0003__v3_0__create_excludeddates_model')]**

`operations` **= [<AddField field=<django.db.models.fields.DateTimeField>, name=u'end_date', model_name=u'schedule**

`radioco.apps.schedules.migrations.0004__v3_0__adding_rrules.`**`migrate_daily_recurrences`**(*apps*, *schema*

Migration to convert weekly recurrences into new complex recurrences

radioco.apps.schedules.migrations.0005__v3_0__migrating_schedules_to_unique_calendar module

class `radioco.apps.schedules.migrations.0005__v3_0__migrating_schedules_to_unique_calendar.`**`Mi`**

Bases: `django.db.migrations.migration.Migration`

`dependencies` **= [(u'schedules', u'0004__v3_0__adding_rrules'), (u'programmes', u'0009__v3_0__small_tweaks')]**

`operations` **= [<RunPython <function migrate_schedules>>, <RemoveField name=u'day', model_name=u'schedule'>, <**

`radioco.apps.schedules.migrations.0005__v3_0__migrating_schedules_to_unique_calendar.`**`calcul`**

Calculation of the last end date to improve performance

`radioco.apps.schedules.migrations.0005__v3_0__migrating_schedules_to_unique_calendar.`**`calcul`**

Calculation of the first start date to improve performance

`radioco.apps.schedules.migrations.0005__v3_0__migrating_schedules_to_unique_calendar.`**`migrat`**

Final Migration to v3.0

Before this migration Calendar (ScheduleBoard) had a date limit and all the schedules were repeated weekly We want to move constraint dates from Calendar to programmes and clone the active schedules into the active calendar

radioco.apps.schedules.migrations.0005__v3_0__migrating_schedules_to_unique_calendar.**recurr**

Fix for django-recurrence 1.3 Avoid outputing a non possible dt

radioco.apps.schedules.migrations.0005__v3_0__migrating_schedules_to_unique_calendar.**recurr**

Fix for django-recurrence 1.3 Avoid outputing a non possible dt

Module contents

radioco.apps.schedules.tests package

Submodules

radioco.apps.schedules.tests.test_recurrences module

class radioco.apps.schedules.tests.test_recurrences.**EmptyRecurrenceTests**(*methodName='runTest'*)

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

Tests to check django-recurrence behaviour when there are no recurrences

setUp()

test_after()

test_after_inclusive()

test_before()

test_before_inclusive()

test_no_values()

class radioco.apps.schedules.tests.test_recurrences.**RecurrenceTests**(*methodName='runTest'*)

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

Tests to check django-recurrence behaviour

setUp()

test_after()

test_after_inclusive()

test_before_dt()

test_before_inclusive()

test_impossible_recurrence_after()

Testing error calling after and function wrapper to solve it (recurrence_after)

test_impossible_recurrence_before()
Testing error calling before and function wrapper to solve it (recurrence_before)

test_last_date_is_equals_before_limit()

test_start_date_is_before_start_limit()

test_when_date_doesnt_exist_on_february()

radioco.apps.schedules.tests.test_schedules module

class radioco.apps.schedules.tests.test_schedules.**CalendarAdminTests**(*methodName='runTest'*)
Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

setUp()

test_clone_calendar()

class radioco.apps.schedules.tests.test_schedules.**CalendarManagerTests**(*methodName='runTest'*)
Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

setUp()

test_current()

test_current_no_date()

class radioco.apps.schedules.tests.test_schedules.**CalendarModelTests**(*methodName='runTest'*)
Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

test_str()

class radioco.apps.schedules.tests.test_schedules.**CalendarValidationTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_only_one_calendar_active()

class radioco.apps.schedules.tests.test_schedules.**ScheduleBetweenTests**(*methodName='runTest'*)
Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

setUp()

test_between_includes_started_episode()

test_dates_between_includes_started_episode()

class radioco.apps.schedules.tests.test_schedules.**ScheduleModelTests**(*methodName='runTest'*)
Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

setUp()

test_date_after()

test_date_after_exclude()

test_date_before()

test_dates_between()

test_dates_between_complex_ruleset()

test_dates_between_earlier_end_by_programme()

test_dates_between_later_start_by_programme()

test_end_gt_calendar()

test_recurrence_rules()

test_runtime()

test_runtime_not_set()

test_save_rearrange_episodes(**args*, ***keywargs*)

test_start()

test_start_lt_calendar()

test_unicode()

class radioco.apps.schedules.tests.test_schedules.**ScheduleUtilsTests**(*methodName='runTest'*)
Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

setUp()

test_available_dates_after()

test_available_dates_none()

test_rearrange_episodes()

test_rearrange_episodes_new_schedule(**args*, ***keywargs*)

test_rearrange_only_non_emited_episodes(**args*, ***keywargs*)

class radioco.apps.schedules.tests.test_schedules.**ScheduleValidationTests**(*methodName='runTest'*)
Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

test_fields()

class radioco.apps.schedules.tests.test_schedules.**TransmissionModelTests**(*methodName='runTest'*)
Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

setUp()

test_at()

test_between()

test_between_by_queryset()

test_end()

test_episode__url()

test_name()

test_programme_url()

test_slug()

test_start()

radioco.apps.schedules.tests.test_schedules.**mock_now**(*dt=datetime.datetime(2014, 1, 1, 13, 30, tzinfo=<UTC>)*)

radioco.apps.schedules.tests.test_timezone module

class radioco.apps.schedules.tests.test_timezone.**ScheduleModelTests**(*methodName='runTest'*)

Bases: *radioco.apps.radioco.test_utils.TestDataMixin*, django.test.testcases.TestCase

setUp()

test_CEST_transition()

test_CET_transition()

test_cleaned_internal_recurrence_dates()

test_transform_dt_to_default_tz()

radioco.apps.schedules.tests.test_timezone.**test_CET_transitions**(*self*)

Module contents

Submodules

radioco.apps.schedules.admin module

class radioco.apps.schedules.admin.**CalendarAdmin**(*model*, *admin_site*)

Bases: django.contrib.admin.options.ModelAdmin

actions = ['clone_calendar', 'set_active']

clone_calendar(*request*, *queryset*)

list_display = ('name', 'is_active')

list_filter = ['is_active']

media

ordering = ['name']

search_fields = ['name']

set_active(*request*, *queryset*)

class radioco.apps.schedules.admin.**ScheduleAdmin**(*model*, *admin_site*)

Bases: django.contrib.admin.options.ModelAdmin

change_list_template = 'admin/schedules/calendar.html'

changelist_view(*request*, *extra_context=None*)

fieldsets = ((None, {'fields': ('programme', 'type', 'start_dt', 'recurrences')}), (<django.utils.functional.__proxy__ ob

has_add_permission(*request*)

media

readonly_fields = ('effective_start_dt', 'effective_end_dt', 'source', 'from_collection')

radioco.apps.schedules.forms module

class `radioco.apps.schedules.forms.`**`DeleteScheduleForm`**(*has_recurrences*, **args*, ***kwargs*)

Bases: `django.forms.forms.Form`

`DELETE_ALL` = **'all'**

`DELETE_ONLY_THIS` = **'only_this'**

`DELETE_THIS_AND_FOLLOWING` = **'this_and_following'**

`base_fields` = **OrderedDict([('transmission_dt', <django.forms.fields.DateTimeField object>), ('schedule', <django.for**

`declared_fields` = **OrderedDict([('transmission_dt', <django.forms.fields.DateTimeField object>), ('schedule', <djan**

`media`

radioco.apps.schedules.models module

class `radioco.apps.schedules.models.`**`Calendar`**(*id*, *name*, *is_active*)

Bases: `django.db.models.base.Model`

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `Calendar.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

classmethod `Calendar.`**`get_active`**()

`Calendar.`**`objects`** = **<django.db.models.manager.Manager object>**

`Calendar.`**`rearrange_episodes`**()

`Calendar.`**`save`**(**args*, ***kwargs*)

`Calendar.`**`schedule_set`**

class `radioco.apps.schedules.models.`**`CalendarManager`**

Bases: `django.db.models.manager.Manager`

`current`()

class `radioco.apps.schedules.models.`**`ExcludedDates`**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`

Helper to improve performance

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `ExcludedDates.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

`ExcludedDates.`**`date`**

`ExcludedDates.`**`get_new_excluded_datetime`**(*new_dt*)

Returns: A new dt to be excluded in that date

`ExcludedDates.`**`get_next_by_datetime`**(**moreargs*, ***morekwargs*)

`ExcludedDates.`**`get_previous_by_datetime`**(**moreargs*, ***morekwargs*)

`ExcludedDates.`**`objects`** = **<django.db.models.manager.Manager object>**

ExcludedDates.**schedule**

class radioco.apps.schedules.models.**Schedule**(*id*, *programme*, *type*, *calendar*, *recurrences*, *start_dt*, *effective_start_dt*, *effective_end_dt*, *from_collection*, *source*)

Bases: django.db.models.base.Model

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

exception Schedule.**MultipleObjectsReturned**

Bases: django.core.exceptions.MultipleObjectsReturned

Schedule.**calendar**

Schedule.**child_schedules**

Schedule.**date_after**(*after*)

Schedule.**date_before**(*before*)

Schedule.**dates_between**(*after*, *before*)

Return a sorted list of dates between after and before

Schedule.**exclude_date**(*dt*)

Schedule.**excludeddates_set**

Schedule.**from_collection**

Schedule.**get_next_by_start_dt**(**moreargs*, ***morekwargs*)

Schedule.**get_previous_by_start_dt**(**moreargs*, ***morekwargs*)

static Schedule.**get_schedule_which_excluded_dt**(*programme*, *dt*)

Schedule.**get_type_display**(**moreargs*, ***morekwargs*)

Schedule.**has_recurrences**()

Schedule.**include_date**(*dt*)

Schedule.**objects = <django.db.models.manager.Manager object>**

Schedule.**programme**

Schedule.**recurrences**

A placeholder class that provides a way to set the attribute on the model.

Schedule.**runtime**

Schedule.**save**(**args*, ***kwargs*)

Schedule.**schedule_set**

Schedule.**source**

class radioco.apps.schedules.models.**Transmission**(*schedule*, *date*, *episode=None*)

Bases: object

Temporal object generated according to recurrence rules or schedule information It contains concrete dates

classmethod at(*at*)

classmethod between(*after*, *before*, *schedules=None*)

Return a tuple of Schedule and Transmissions sorted by date

end

episode_url

name

programme_url

slug

radioco.apps.schedules.models.**calculate_effective_schedule_end_dt**(*schedule*)
Calculation of the last end date to improve performance

radioco.apps.schedules.models.**calculate_effective_schedule_start_dt**(*schedule*)
Calculation of the first start date to improve performance

radioco.apps.schedules.urls module

radioco.apps.schedules.utils module

radioco.apps.schedules.utils.**next_dates**(*calendar*, *programme*, *after*)
Returns: A generator with the next dates of a given programme

radioco.apps.schedules.views module

class radioco.apps.schedules.views.**DeleteScheduleView**(***kwargs*)
Bases: *radioco.apps.radioco.utils.GetObjectMixin*, *radioco.apps.radioco.utils.DeletePermissionMixin*, django.views.generic.edit.FormView

form_class
alias of DeleteScheduleForm

form_valid(*form*)

get(*request*, **args*, ***kwargs*)

get_context_data(***kwargs*)

get_form_kwargs()

get_initial()

model
alias of Schedule

schedule_id = None

template_name = 'admin/schedules/delete_modal.html'

transmission_dt = None

radioco.apps.schedules.views.**schedule_list**(*request*)

Module contents

radioco.apps.users package

Subpackages

radioco.apps.users.migrations package

Submodules

radioco.apps.users.migrations.0001_initial module

class radioco.apps.users.migrations.0001_initial.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'auth', u'__first__')]

operations = [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'bio', <django.db.models.fields.Tex

radioco.apps.users.migrations.0002_change_textfield_to_richtextfield module

class radioco.apps.users.migrations.0002_change_textfield_to_richtextfield.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'users', u'0001_initial')]

operations = [<AlterField field=<ckeditor.fields.RichTextField>, preserve_default=True, name=u'bio', model_name=u

radioco.apps.users.migrations.0003_auto_20160104_2029 module

class radioco.apps.users.migrations.0003_auto_20160104_2029.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'users', u'0002_change_textfield_to_richtextfield')]

operations = [<AlterField field=<django.db.models.fields.files.ImageField>, name=u'avatar', model_name=u'userprofi

radioco.apps.users.migrations.0004__v3_0__ckeditor_upload_field module

class radioco.apps.users.migrations.0004__v3_0__ckeditor_upload_field.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'users', u'0003_auto_20160104_2029'), (u'radioco', u'0001__v3_0__mysql_timezone')]

operations = [<AlterField field=<ckeditor_uploader.fields.RichTextUploadingField>, name=u'bio', model_name=u'use

Module contents

Submodules

radioco.apps.users.admin module

class radioco.apps.users.admin.**NonStaffUserProfileForm**(**args*, ***kwargs*)
Bases: django.forms.models.ModelForm

base_fields = OrderedDict([('username', <django.forms.fields.CharField object>), ('first_name', <django.forms.fields

clean_username()

declared_fields = OrderedDict([('username', <django.forms.fields.CharField object>), ('first_name', <django.forms

media

save(*force_insert=False*, *force_update=False*, *commit=True*)

class radioco.apps.users.admin.**SingletonProfileAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

change_view(*request*, *object_id*, *extra_context=None*)

fields = ['username', 'first_name', 'last_name', 'email', 'bio', 'avatar', 'display_personal_page']

form
alias of *NonStaffUserProfileForm*

get_urls()

has_add_permission(*request*)

has_delete_permission(*request*, *obj=None*)

media

response_change(*request*, *obj*)

save_model(*request*, *obj*, *form*, *change*)

class radioco.apps.users.admin.**UserProfileAdmin**(*model*, *admin_site*)
Bases: django.contrib.auth.admin.UserAdmin

inlines = (<class 'radioco.apps.users.admin.UserProfileInline'>,)

list_filter = ('is_staff', 'is_superuser', 'is_active', 'groups', 'userprofile__display_personal_page')

media

class radioco.apps.users.admin.**UserProfileInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.StackedInline

can_delete = False

exclude = ('slug',)

extra = 1

inline_classes = ('grp-collapse grp-open',)

media

model
alias of UserProfile

radioco.apps.users.models module

class radioco.apps.users.models.**UserProfile**(*id*, *user*, *bio*, *avatar*, *display_personal_page*, *slug*)
Bases: django.db.models.base.Model

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception UserProfile.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

UserProfile.**avatar**

Just like the FileDescriptor, but for ImageFields. The only difference is assigning the width/height to the width_field/height_field, if appropriate.

UserProfile.**get_absolute_url**()

UserProfile.**objects = <django.db.models.manager.Manager object>**

UserProfile.**save**(**args*, ***kwargs*)

UserProfile.**user**

radioco.apps.users.models.**save_slug**(*sender*, *instance=None*, ***kwargs*)

radioco.apps.users.tests module

class radioco.apps.users.tests.**UserProfileMethodTests**(*methodName='runTest'*)

Bases: django.test.testcases.TestCase

test_get_announcers_and_profile()

test_save()

radioco.apps.users.urls module

radioco.apps.users.views module

class radioco.apps.users.views.**UsersView**(***kwargs*)

Bases: django.views.generic.list.ListView

get_queryset()

model

alias of UserProfile

radioco.apps.users.views.**userprofile_detail**(*request*, *slug*)

Module contents

Module contents

radioco.configs package

Subpackages

radioco.configs.base package

Submodules

radioco.configs.base.manage module

radioco.configs.base.settings module

radioco.configs.base.settings.**gettext_noop**(*s*)

radioco.configs.base.test_runner module

class radioco.configs.base.test_runner.**MyTestSuiteRunner**(**args*, ***kwargs*)
Bases: django.test.runner.DiscoverRunner

radioco.configs.base.urls module

Module contents

radioco.configs.development package

Submodules

radioco.configs.development.manage module

radioco.configs.development.settings module

Module contents

radioco.configs.heroku package

Submodules

radioco.configs.heroku.manage module

radioco.configs.heroku.settings module

radioco.configs.heroku.wsgi module

Module contents

radioco.configs.staging package

Submodules

radioco.configs.staging.manage module

radioco.configs.staging.settings module

radioco.configs.staging.wsgi module

Module contents

Module contents

Module contents

tasks package

Submodules

tasks.docker module

tasks.docs module

tasks.heroku module

tasks.locale module

tasks.radioco module

tasks.utils module

Module contents

Release notes & upgrade information

Some versions of RadioCo present more complex upgrade paths than others, and some **require** you to take action. It is strongly recommended to read the release notes carefully when upgrading.

It goes without saying that you should **backup your database** before embarking on any process that makes changes to your database.

1.1 release notes

What's new in 1.1

- Numerous updates to the documentation
- Updates to facilitate the starting-up (radioco_recorder_user and a empty shedule_board are self-created)
- Added custom languages to programmes (by default, all django languages)

How this affects you

If you're starting with a new installation, you don't need to worry about this. Don't even bother reading this section; it's for upgraders.

Activate your virtualenv and do the following in your project main directory:

```
pip install django-radio==1.1
python manage.py migrate
```

1.1.1 release notes

What's new in 1.1.1

- Added CKEditor to programme's synopsis, episode's summary and user's biography
- Added default settings to the project
- Added customisable footer

How this affects you

If you're starting with a new installation, you don't need to worry about this. Don't even bother reading this section; it's for upgraders.

Activate your virtualenv and do the following in your project main directory:

```
pip install django-radio==1.1.1
python manage.py collectstatic
python manage.py migrate
```

In your settings.py import default settings:

```
from radio.settings_base import *
```

1.2 release notes

What's new in 1.2

- Fixed project urls when RadioCo is deployed in a subdirectory
- Fixed default url of photos when STATIC_URL is not "/static/"
- Fixed errors 404 when users don't have a personal webpage.

How this affects you

If you're starting with a new installation, you don't need to worry about this. Don't even bother reading this section; it's for upgraders.

Activate your virtualenv and do the following in your project main directory:

```
pip install django-radio==1.2
python manage.py migrate
```

2.1 release notes

What's new in 2.1

- New layout! The whole site has been redesigned
- Added image versions, RadioCo will adapt your images to fit in the new template
- Added comments support.

- Added analytics support.

How this affects you

If you’re starting with a new installation, you don’t need to worry about this. Don’t even bother reading this section; it’s for upgraders.

You need to replace your current source with the content of https://github.com/iago1460/django-radio. To setup your settings please read the configuration section.

You should be able to keep your current database but make sure to create a backup before start.

```
pip install -r radio/configs/common/requirements.txt
python manage.py collectstatic
python manage.py migrate
```

3.0 release notes

What’s new in 3.0

In this version RadioCo has almost been rewritten completely, internally the major improvements are:

- Use of time-zone-aware datetime objects (more info)
- Added good amount of tests
- Changed calendar to use API methods
- Using bower to download js and css dependencies
- Renamed internal folders and models
  - django-radio/radio –> django-radio/radioco
  - django-radio/radio/configs/common –> django-radio/radioco/configs/base
  - ScheduleBoard model has been renamed to Calendar
  - dashboard app has been merged into schedules

Special mention to @stefan-walluhn - Radio Corax who helped in this release, specifically adding Bower, Rest Framework API, tests and complex recurrence rules.

And the visible features are:

- Added download button to episodes (#38 Credit to @AlexCortinas)
- Removed unnecessary Scroll configuration setting (#13 Credit to @AlexCortinas)
- Improvements in title representation of episodes when not set (#17 Credit to @txenoo)
- Removed warnings when compiling (#32 Thanks to @txenoo)
- Added Radiocom endpoints (Thanks to @pablogrela)
- Added upload feature to CKEditor (#22)
- Added new calendar setting slotDuration
- Added a suite of helper tasks using invoke (Easy deploy to docker)
- Added public API to access to programmes, episodes, schedules and transmissions (#2)
- The schedule manager has received important updates (#20, #4)

Now managing schedules should be a lot more simple. From the manager is possible to create and edit programmes in addition to schedules.

The forced weekly recurrence has been removed in favor of complex recurrence rules, having a unique calendar should be enough.

The Calendar model doesn't have date constraints anymore and only one can be active at the same time. If you want to restrict a programme you should do it filling the start and end date fields inside itself.

How this affects you

If you're starting with a new installation, you don't need to worry about this. Don't even bother reading this section; it's for upgraders.

Please **make sure of having a copy of your database** before even started to look to the next steps.

If you have custom code or templates I recommend putting them in a safe place, special mention to local_settings.py files, your are going to need to move them to the correct folder.

Since this migration introduces timezone support, it's necessary to set the timezone variable:

1. Find your timezone in this list
2. Go to the config folder and create if not exists a local_settings.py in the same directory than settings.py
3. Add to the local settings the variable timezone, for example: TIME_ZONE = "Europe/Madrid"

Warning: Using an incorrect timezone will cause incorrect dates, you should set/override that setting in your local_settings.py.

Warning: If you have a database backend different to PostgreSQL you must convert your data from local time to UTC. Fortunately for you a migration was written to fix episode issue_dates, other dates will suffer alterations.

Warning: We strongly recommend to use PostgreSQL as backend. Other database engines **will not be supported** in the future.

You have to install bower before running the following commands in the project root folder:

```
bower install
pip install -r radioco/configs/base/requirements.txt
python manage.py collectstatic
python manage.py migrate
```

After the migration process you will find that there is a new calendar set to active that contains all your schedules and your other calendars have been renamed with the prefix "Legacy - ". Feel free of delete these, they are not require anymore.

Note: Warning messages could appear during the migration process, it's usually safe to ignore them.

3.0.1 release notes

What's new in 3.0.1

- Fixing invoke commands.
- Simplified docker setup.

Take a look to our installation section.

How this affects you

If you're starting with a new installation, you don't need to worry about this. Don't even bother reading this section; it's for upgraders.

You need to replace your current source with the content of https://github.com/iago1460/django-radio. To setup your settings please read the configuration section.

This update doesn't require any further steps.

3.1 release notes

What's new in 3.1

- Fixing staging docker port issue. (Thanks to @mmontes11 for report)
- Adding stable Radiocom API
- Solving ticket #43
- Upgrading libraries to last version
- Small fixes to invoke scripts
- Improving API (Outputting absolute urls and adding some fields)

How this affects you

If you're starting with a new installation, you don't need to worry about this. Don't even bother reading this section; it's for upgraders.

You need to replace your current source with the content of https://github.com/iago1460/django-radio. To setup your settings please read the configuration section.

You should be able to keep your current database but make sure to create a backup before start.

```
pip install -r radioco/configs/base/requirements.txt
python manage.py collectstatic
python manage.py migrate  # check the following note in case of error
```

Note: When migrating from version 3.0 to version 3.1 the following error will appear during the migration process *django.db.utils.ProgrammingError: relation "recurrence_date" already exists* to solve this please run the following command:

```
python manage.py migrate --fake-initial
```

Or if you are using our docker setup:

```
inv docker.clean -e staging
inv docker.setup -e staging  # check the following note in case of error
```

Note: If a migration error appear during the setup please run:

```
inv docker.manage -e staging -c "migrate --fake-initial"
```

3.2 release notes

What's new in 3.2

- Solving ticket #5
- Improving rss (adding images and fixing itunes encoding)
- Added rss autodiscovery
- Documenting django.contrib.sites
- Improving ORM queries
- Adding cache to singleton models using memcached
- Improve performance in staging config using nginx cache, uwsgi and caching templates
- Updating Spanish and Galician translations
- Fixing progress bar in the main page

How this affects you

If you're starting with a new installation, you don't need to worry about this. Don't even bother reading this section; it's for upgraders.

You need to replace your current source with the content of https://github.com/iago1460/django-radio. To setup your settings please read the configuration section.

You should be able to keep your current database but make sure to create a backup before start.

Warning: You must ensure your episodes and programmes only have one role per user. A database constraint has been put in place as part of this migration.

Warning: This migration depends on `settings.LANGUAGE_CODE`, it will convert the role field into a editable CharField using your preferred language. Note you can change this value temporally to run this migration. More info.

```
pip install -r radioco/configs/base/requirements.txt
python manage.py collectstatic
python manage.py migrate
```

Note: If you have a custom production environment please consider to apply our staging setup to improve performance.

Or if you are using our docker setup:

```
inv docker.clean -e staging
inv docker.setup -e staging
```

3.2.1 release notes

What's new in 3.2.1

- Making rss links protocol aware so they can be opened by podcast apps.

How this affects you

If you're starting with a new installation, you don't need to worry about this. Don't even bother reading this section; it's for upgraders.

You need to replace your current source with the content of https://github.com/iago1460/django-radio. To setup your settings please read the configuration section.

This update doesn't require any further steps.

3.2.2 release notes

What's new in 3.2.2

- Fixing bug that made impossible deleting inline objects.
- Removing undocumented switch user functionality.
- Tweaking calendar css to improve readability.
- Filtering for active programmes in the home page.

Thanks to @beaulalume for reporting the previous issues.

How this affects you

If you're starting with a new installation, you don't need to worry about this. Don't even bother reading this section; it's for upgraders.

You need to replace your current source with the content of https://github.com/iago1460/django-radio. To setup your settings please read the configuration section.

You should be able to keep your current database but make sure to create a backup before start.

```
python manage.py collectstatic
```

Or if you are using our docker setup:

```
inv docker.clean -e staging
inv docker.setup -e staging
```

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index