
onespacemedia-cms Documentation

Release 1.5.7

Daniel Samuels

July 13, 2015

1	Features enabled by default	3
2	Optional Features	5
3	Integrations	7
3.1	Getting Started	7
3.2	Links Module	9
3.3	Media Module	10
3.4	News Module	11
3.5	Pages Module	14
3.6	Testing	20
3.7	Moderation Plugin	21

A collection of Django extensions that add content-management facilities to Django projects, originally forked from [etianen/cms](#). The team at [Onespacemedia](#) are actively working on the project, adding new features, updating existing ones and keeping it working with the latest versions of Django.

The current version of onespacemedia-cms should always be compatible with the current version of Django.

Note: A lot of the CMS has settings specific to how [Onespacemedia](#) works and as such you will likely to need to change some of them to suit your needs. In addition, this project installs a lot of other packages which you may not need. You may prefer to use the [original repository](#) instead of this one.

Features enabled by default

- Publication controls with online preview.
- Pre-configured WYSIWYG editor widgets (using [Redactor](#))
- Hierarchical page management with no depth limit.
- Image and file management, with easy embedding via WYSIWYG editors or pure model fields.
- Internal / external links in menus (via bundled [\[\[Links app\]\]](#)).
- Simple blog management (via bundled [\[\[News app\]\]](#)).
- Version control and rollback (via [django-reversion](#)).
- Automatic SEO-friendly redirect management (via [django-historylinks](#)).
- Full-text search with relevance ranking (via [django-watson](#)).
- Full ORM caching (via [django-cachalot](#)).
- CSS and JS compression (via [django-compressor](#)).
- Image thumbnailing (via [sorl.thumbnail](#)).
- Deployment and server management (via [server-management](#)).

Optional Features

- FAQs module (via `cms-faqs`)
- Jobs module (via `cms-jobs`)
- People module (via `cms-people`)
- Photoshop-like image editing.
- Admin login via Google+.
- News approval system.

Integrations

The CMS integrates directly with a lot of third-party services, and it's usually worth configuring them to get even greater benefit from the CMS. Currently supported are:

- Adobe Creative SDK (formerly known as Aviary).
- Sentry.
- Google+ authentication API.

3.1 Getting Started

3.1.1 Installation

To install onespacemedia-cms simply run:

```
$ pip install onespacemedia-cms
```

If you want the ability to invite users via the admin, then run:

```
$ pip install onespacemedia-cms[usertools]
```

Once the project has been installed, you can create a project with the following command:

```
$ start_cms_project.py project_name .
```

Where `project_name` is the name of your project and `.` is the current directory (this will cause the `manage.py` to be created in your working directory, rather than in a subfolder).

You should now run the following:

```
$ ./manage.py migrate
$ ./manage.py createsuperuser
```

3.1.2 Configuration

`start_cms_project.py` takes care of a lot of the settings automatically, but it's worth taking a look through the `settings/base.py` and updating any settings which still require values, the key ones being:

- `SITE_NAME`
- `SITE_DOMAIN`

- EMAIL_HOST / EMAIL_HOST_USER / EMAIL_HOST_PASSWORD
- ADMINS
- WHITELISTED_DOMAINS

To configure the various third-party integrations, you will need to generate API keys on each of the platforms and add them to the base settings file.

Google+ Authentication API

1. Log in to the [Google Developers Console](#).
2. Click “Create Project”.
 - (a) Enter your Project Name.
 - (b) (optional) Click “Show advanced options..” and change the data center location.
 - (c) Click “Create”.
3. Click APIs & auth -> Consent screen.
 - (a) Select your Email Address.
 - (b) Enter your Product Name.
 - (c) Click “Save”.
4. Click APIs & auth -> APIs
 - (a) Enable the “Google+ API”.
5. Click APIs & auth -> Credentials
 - (a) Click “Create new Client ID”
 - (b) Enter your domain name into the “Authorized Javascript Origins” textarea.
 - (c) Click “Create Client ID”.
6. Copy the “Client ID” to the setting named `SOCIAL_AUTH_GOOGLE_PLUS_KEY`.
7. Copy the “Client Secret” to the setting named `SOCIAL_AUTH_GOOGLE_PLUS_SECRET`.

Adobe Creative SDK

1. Log in to the [Adobe website](#).
2. Click “New Application”.
3. Enter your application name.
4. Select “Web”.
5. Enter a description.
6. Fill out the CAPTCHA.
7. Click “Add Application”.
8. Copy the secret key to `ADOBE_CREATIVE_SDK_CLIENT_SECRET`.
9. Copy the API key to `ADOBE_CREATIVE_SDK_CLIENT_ID`.
10. Set `ADOBE_CREATIVE_SDK_CLIENT_ID` to `True`.

Sentry

1. Log in to [Sentry](#)
2. Create a new Project.
3. Enter the Name.
4. Set Platform to Django.
5. Click “Save Changes”.
6. Copy the DSN from the modal window to the empty string in `settings/production.py`.

3.2 Links Module

The Links module provides a new page content type named “Link” which allows you to have a navigation item without a page associated.

3.2.1 Configuration

Ensure both `cms.apps.pages` and `cms.apps.links` are in your project’s `INSTALLED_APPS` then add the new model to your database with the following:

```
$ ./manage.py migrate
```


3.2.2 Usage

To add a Link to your site simply add a Page. If you have more than one content type you will be shown a page such as this:

[Home](#) › [Pages](#) › [Page](#) › Select page type


Select a type of page from the choices below.

Content



Content

Utilities



Link

If you do not have any other page content types you will be taken straight to the add form. The form itself is very straightforward, simply add the Title for the page and a URL to redirect to.

3.3 Media Module

The media app provides a file and image management interface to the CMS admin. It also integrates with WYSIWYG text editors to provide a file browser and image browser interface that allows images and files to be uploaded directly into the editor.

3.3.1 Models

To make it easier to integrate the media module into your project a selection of models are provided.

class FileRefField

Provides a widget which allows a user to select a file from the media library.

class ImageRefField

The same functionality as the `FileRefField()`, but with the files filtered to only show images.

class VideoFileRefField

The same functionality as the `FileRefField()`, but with the files filtered to only show videos.

class VideoRefField

A `Video` object is a collection of video files and related imagery. You can use it to easily create cross-browser compatible `<video>` tags on the frontend of your website.

3.4 News Module

The News module is a standard news aggregation system which works in two parts. The first part is a “News feed” which is a page content type designated as a container for articles. You can have multiple news feeds in a site and assign articles to any one of them.

3.4.1 Features

- Multiple news feeds per-site.
- Built-in RSS feeds.
- Archive views allowing you to see lists by year, month and date.
- Ability to add articles to categories.
- Ability to associate multiple authors with an article.
- An optional approval system.

The fields available on an Article are:

- Title
- News feed
- Date
- Status (if enabled)
- Image
- Content
- Summary
- Categories
- Authors
- Online / offline toggle

3.4.2 Configuration

Add `cms.apps.news` to your `INSTALLED_APPS` and then add the models to your database with:

```
$ ./manage.py migrate
```

If you would like to enable the approval system set the following setting:

```
NEWS_APPROVAL_SYSTEM = True
```

3.4.3 Usage

First create a News Feed by adding a new page and selecting the “News feed” option then add an Article and select the newly created News feed.

If you have enabled the approval system an additional field appears on the edit form named `status`. By default the status is set to ‘draft’, users without the `news.can_approve_articles` permission can only set the status to either ‘draft’ or ‘submitted’. An article in either of these states will not appear on the front-end website, but is visible within the administration system. Users with the appropriate permission are able to set articles to ‘approved’ and thus make the articles appear on the website.

3.4.4 Overriding templates

The news app has a very granular set of templates, allowing you to change any aspect of the default output with very little effort.

Template hierarchy

`news/base.html`

Extends: `base.html`

Includes:

- `news/includes/article_category_list.html`
- `news/includes/article_date_list.html`

`news/article_archive.html`

Extends: `news/base.html`

Includes:

- `news/includes/archive_list.html`
- `news/includes/archive_list_item.html`
- `news/includes/article_meta.html`
 - `news/includes/article_date.html`
 - `news/includes/article_category_list.html`

`news/article_archive_day.html`

Extends: `news/article_archive.html`

`news/article_archive_month.html`

Extends: `news/article_archive.html`

news/article_archive_year.html

Extends: news/article_archive.html

news/article_category_archive.html

Extends: news/base.html

Includes:

- news/includes/article_list.html
- news/includes/article_list_item.html
- news/includes/article_meta.html
 - news/includes/article_date.html
 - news/includes/article_category_list.html

news/article_detail.html

Extends: news/base.html

Includes:

- news/includes/article_meta.html
 - news/includes/article_date.html
 - news/includes/article_category_list.html

3.4.5 Template tags

article_list (*context, article_list*)

Renders a list of news articles.

article_url (*context, article, page*)

Renders the URL for an article.

article_list_item (*context, article, page*)

Renders an item in an article list.

article_archive_url (*context, page*)

Renders the URL for the current article archive.

category_url (*context, category, page*)

Renders the URL for the given category.

category_list (*context, category_list*)

Renders a list of categories.

article_year_archive_url (*context, year, page*)

Renders the year archive URL for the given year.

article_day_archive_url (*context, date, page*)

Renders the month archive URL for the given date.

article_date (*context, article*)

Renders a rich date for the given article.

article_category_list (*context, article*)

Renders the list of article categories.

article_meta (*context, article*)

Renders the metadata of an article.

article_date_list (*context, page*)

Renders a list of dates.

article_latest_list (*context, page, limit=5*)

Renders a widget-style list of latest articles.

get_article_latest_list (*context, page, limit=5*)

A wrapper around `article_latest_list` which returns the dictionary rather than render it.

3.4.6 Known issues

- You cannot add fields to News Feeds or Articles easily, you have to copy the application into your project.

3.5 Pages Module

The pages module is a standard CMS module which allows users to add pages to the website. However, it does not do anything by itself, it requires you to add models which extend `ContentBase`.

3.5.1 How it works

In `cms.apps.pages.models` there is a model named `Page`, which contains a set of generic page fields such as `parent`, `publication_date`, `expiry_date` and so on. To be able to add a `Page` to the site, we need a “Content Model”. A Content Model is a way of representing a type of page, this could be a homepage, a standard page, a contact page etc – each Content Model would have it’s own front-end template thus allowing you to provide different layout types, themes etc throughout your site.

The default CMS project contains an example Content Model in `site/models.py` (but is commented out by default to avoid an initial migration being made by mistake). Here’s the same example:

```
from django.db import models

from cms.apps.pages.models import ContentBase

class Content(ContentBase):

    content = models.TextField(
        blank=True,
    )
```

As you can see, this will create a Page type named 'Content' with a textarea named 'content'. In addition to our custom field, we will have the following fields:

- Title
- URL Title (which will automatically populate as you type into the Title field)
- Parent page selector
- Publication date
- Expiry date
- Online status
- Navigation title (if you want a shorter version of the Page name to show in the navigation)
- "Add to navigation" selection, allowing you to add / remove the page from the navigation
- SEO controls:
 - Browser title
 - Page keywords
 - Page description
 - Sitemap priority
 - Sitemap change frequency
 - Allow index
 - Follow links
 - Allow archiving

If we wanted to have another page type with a different set of fields (or even the same fields) we simply have to add another model which extends `ContentBase`, like so:

```
from django.db import models

from cms.apps.pages.models import ContentBase

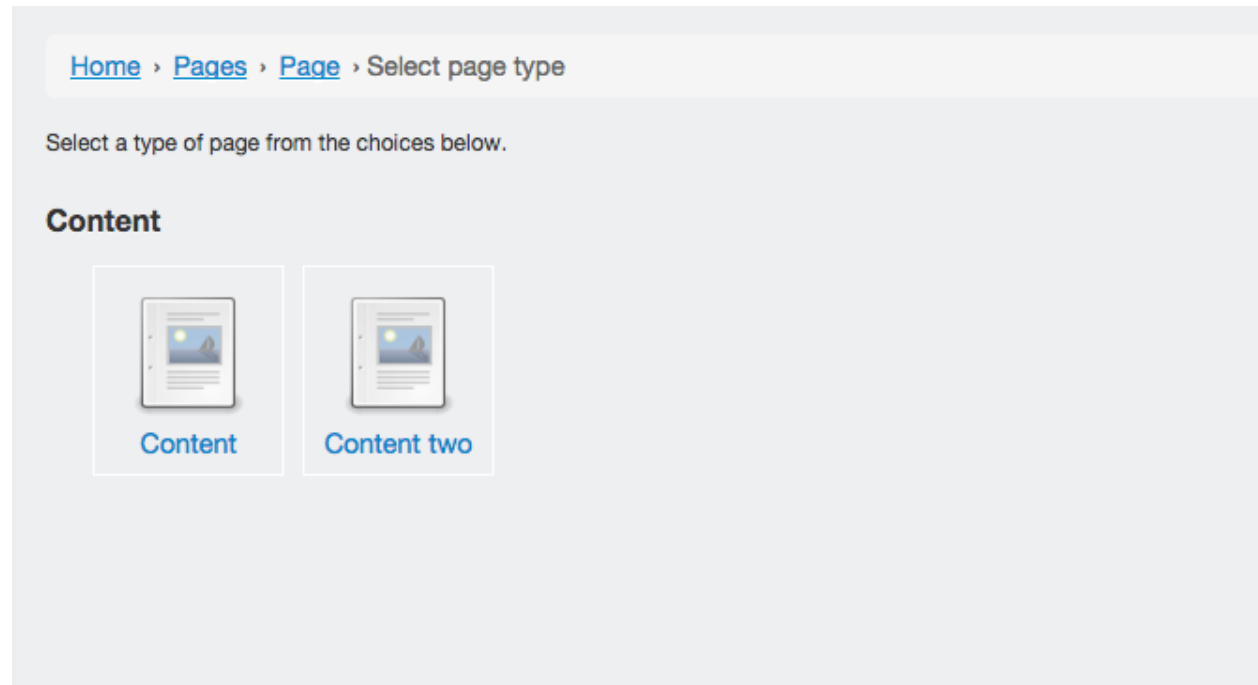

class Content(ContentBase):

    content = models.TextField(
        blank=True,
    )


class ContentTwo(ContentBase):

    content = models.TextField(
        blank=True,
    )
```

With this structure in place, we would then get a choice of content types when adding a page.



3.5.2 Context processor

The pages module automatically adds a variable named `pages` to your template context, this gives you access to the page data and content for the current page and the homepage. Let's assume your model looks like this:

```
from django.db import models

from cms.apps.pages.models import ContentBase

class Content(ContentBase):

    introduction = models.TextField(
        blank=True,
    )
```

You can access the page data like this:

```
<!-- The Page object -->
{{ pages.current }}

<!-- The Content model (which extends ContentBase) -->
{{ pages.current.content }}

<!-- Fields on the Page model -->
{{ pages.current.title }}
{{ pages.current.slug }}

<!-- Fields on the Content model -->
{{ pages.current.content.introduction }}
```

The `content` attribute on the `Page` model is a method which performs a `ContentType` lookup against the content ID allowing access to the fields of the `Content` model.

3.5.3 Template tags

A collection of template tags are included with the pages module, mostly for the purposes of simplifying SEO. You can load them into the template like this:

```
{% load pages %}
```

navigation (*context, pages, section=None*)

Renders the site navigation using the template specified at `pages/navigation.html`. By default this is just an unordered list with each navigation item as a list item. The simplest usage is like this:

```
{% navigation pages.homepage.navigation %}
```

Which would produce an output like this:

```
<ul>
  <li>
    <a href="/page-1/">Page One</a>
  </li>

  <li>
    <a href="/page-2/">Page Two</a>
  </li>
</ul>
```

If you would like the “base page” (the page that the navigation is being based off) to be included in the navigation simply add the `section` kwarg:

```
{% navigation pages.homepage.navigation section=pages.homepage %}
```

The output of this would be:

```
<ul>
  <li>
    <a class="here" href="/">Homepage</a>
  </li>

  <li>
    <a href="/page-1/">Page One</a>
  </li>

  <li>
    <a href="/page-2/">Page Two</a>
  </li>
</ul>
```

get_navigation (*context, pages, section=None*)

This is a wrapper around `navigation`, but returns the navigation list instead of rendering it out to the page.

page_url (*page, view_func=None, *args, **kwargs*)

Gets the URL of a Page’s view function.

TODO: Expand on this.

meta_description (*context, description=None*)

Renders the content of the meta description tag for the current page:

```
{% meta_description %}
```

You can override the meta description by setting a context variable called `meta_description`:

```
{% with "foo" as meta_description %}
  {% meta_description %}
{% endwith %}
```

You can also provide the meta description as an argument to this tag:

```
{% meta_description "foo" %}
```

meta_robots (*context*, *index=None*, *follow=None*, *archive=None*)

Renders the content of the meta robots tag for the current page:

```
{% meta_robots %}
```

You can override the meta robots by setting boolean context variables called `robots_index`, `robots_archive` and `robots_follow`:

```
{% with 1 as robots_follow %}
  {% meta_robots %}
{% endwith %}
```

You can also provide the meta robots as three boolean arguments to this tag in the order ‘index’, ‘follow’ and ‘archive’:

```
{% meta_robots 1 1 1 %}
```

title (*context*, *browser_title=None*)

Renders the title of the current page:

```
{% title %}
```

You can override the title by setting a context variable called `title`:

```
{% with "foo" as title %}
  {% title %}
{% endwith %}
```

You can also provide the title as an argument to this tag:

```
{% title "foo" %}
```

breadcrumbs (*context*, *page=None*, *extended=False*)

Renders the breadcrumbs trail for the current page:

```
{% breadcrumbs %}
```

To override and extend the breadcrumb trail within page applications, add the `extended` flag to the tag and add your own breadcrumbs underneath:

```
{% breadcrumbs extended=1 %}
```

header (*context*, *page_header=None*)

Renders the header for the current page:

```
{% header %}
```

You can override the page header by providing a header or title context variable. If both are present, then header overrides title:

```
{% with "foo" as header %}
    {% header %}
{% endwith %}
```

You can also provide the header as an argument to this tag:

```
{% header "foo" %}
```

3.5.4 FAQs

Can I change the content type after the page has been created?

Yes, but it has risks. Changing the content type will cause you to lose data in any fields which don't exist in the new model, that is to say that if your structure looks like this:

```
class Content(ContentBase):

    content = models.TextField(
        blank=True,
    )

class ContentTwo(ContentBase):

    content = models.TextField(
        blank=True,
    )
```

You can switch without issue as they have the same fields, however if your models look like this:

```
class Content(ContentBase):

    content = models.TextField(
        blank=True,
    )

class ContentTwo(ContentBase):

    description = models.TextField(
        blank=True,
    )
```

You would lose the data in the content field (on save) if you switched the content type from Content to ContentTwo.

If you still want to change the content type, then it's reasonably simple.

1. Go to the create page of the content type you want to change to. Copy the number from the ?type=XX portion of the URL.
2. Go to the edit page of the page you wish to switch.
3. Add ?type=XX to the end of the URL.

At this point you will be looking at the fieldset for the new content type, but you will not have applied the changes. If you're happy with the way your data looks hit Save and the changes will be saved.

Can I change the `ModelAdmin` fieldsets of a model admin view?

Yes. Simply add the `fieldsets` tuple to your model rather than your `admin.py`.

Can I set a `filter_horizontal` on a content model `ManyToManyField`?

Yes. Simply add the `filter_horizontal` tuple to your model rather than your `admin.py`.

Can I add inline model admins to content models?

Yes. In your `admin.py` add code that looks like this:

```
from django.contrib import admin

from .models import Content, CarouselSlide

from cms.apps.pages.admin import page_admin


class CarouselSlideInline(admin.StackedInline):
    model = CarouselSlide

page_admin.register_content_inline(Content, CarouselSlideInline)
```

3.6 Testing

A very high test coverage is maintained in the project, typically at 99%+. This helps us evaluate how the CMS works across various Python and Django versions. As the CMS is not a Django project as such, you are unable to test it directly and instead have to create a CMS project and run the test from there.

3.6.1 Configuration

Install the Django version you would like to test against, then install the CMS with the test packages enabled and then create a new project:

```
$ pip install Django==1.x.x
$ pip install onespacemedia-cms[testing]
$ start_cms_project.py testing . --without-people --without-faqs --without-jobs --skip-frontend
```

You can then run the tests with:

```
$ ./manage.py test cms
```

If you would like coverage information you can use:

```
$ coverage run --source=cms --omit='*migrations*' manage.py test cms && coverage html
```

This will output a set of HTML files which you can open with:

```
$ open htmlcov/index.html
```

3.7 Moderation Plugin

Built into the CMS is a fully featured moderation system which allows any model in your project to be controlled by a status system. Models which utilise the moderation system gain a field named `status` which has three possible values: “Draft”, “Submitted for approval” or “Approved”. Objects are only visible on the front-end of the website when they are marked as “Approved”.

Adding the moderation system to a model will create a new permission to be created named “Can approve items”. Users will need to have this permission to be able to publish items to the website by setting the object status to “Approved”, users without the permission will only be able to set the object status to “Draft” or “Submitted for approval”.

3.7.1 Adding the moderation system to a model

There are a few steps required to integrate the moderation system with your models. You will need to modify your `models.py` to look like this:

```
from cms.plugins.moderation.models import ModerationBase

class MyModel(ModerationBase):

    # If you wish to set Meta settings, you need to extend ModerationBase.Meta.
    class Meta(ModerationBase.Meta):
        pass
```

To integrate the moderation system with the Django admin system modify your `admin.py` to use this structure:

```
from django.contrib import admin

from .models import MyModel

from cms.plugins.moderation.admin import MODERATION_FIELDS, ModerationAdminBase

class MyModelAdmin(ModerationAdminBase):

    # If your fieldsets variable only contains MODERATION_FIELDS, you can omit
    # this variable entirely as this configuration is the default.
    fieldsets = (
        MODERATION_FIELDS,
    )

admin.site.register(MyModel, MyModelAdmin)
```

If your model already existed before adding the moderation system you will need to run `./manage.py update_permissions` (and probably restart the server) before they appear.

A

article_archive_url(), 13
article_category_list(), 14
article_date(), 14
article_date_list(), 14
article_day_archive_url(), 13
article_latest_list(), 14
article_list(), 13
article_list_item(), 13
article_meta(), 14
article_url(), 13
article_year_archive_url(), 13

B

breadcrumbs(), 18

C

category_list(), 13
category_url(), 13

F

FileRefField (built-in class), 10

G

get_article_latest_list(), 14
get_navigation(), 17

H

header(), 18

I

ImageRefField (built-in class), 10

M

meta_description(), 17
meta_robots(), 18

N

navigation(), 17

P

page_url(), 17

T

title(), 18

V

VideoFileRefField (built-in class), 10
VideoRefField (built-in class), 10