django-basic-cms Documentation

Release 0.1.5

YD-Technology

September 07, 2013

CONTENTS

Welcome on the documentation of the simple multilingual Django Basic CMS (package name: django-basic-cms). You track the latest changes to the code base on the github project page. To get more information about this CMS and its feature go to the *Introduction section*.

To get the source code with git use:

```
git clone git://github.com/YD-Technology/django-basic-cms.git django-basic-cms
```

To get installations instructions go to the *Installation section*.

The latest changelog can be found here *Changelog section*.

CHAPTER

ONE

TABLE OF CONTENT

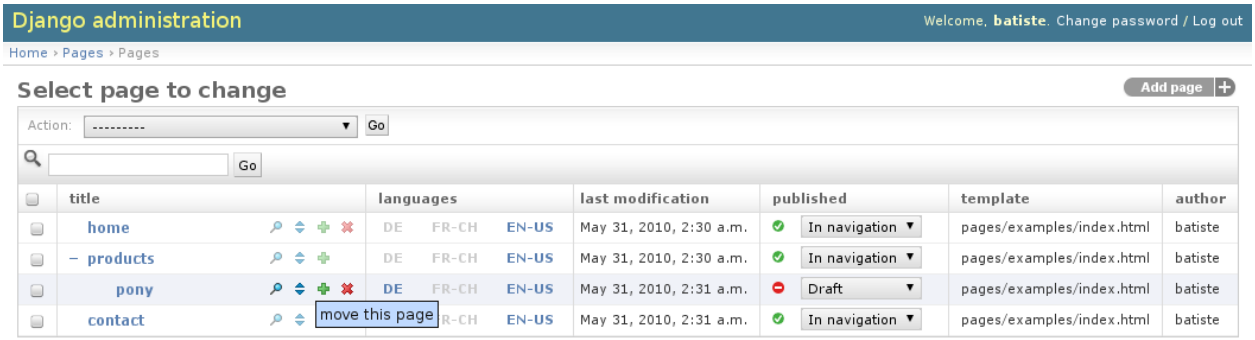
1.1 Introduction

Django Basic CMS enable you to create and administrate hierarchical pages in a simple and powerful way.

Django Basic CMS is based around a placeholders concept. A placeholder is special template tag that you use in your page templates. Every time you add a placeholder in your template a field dynamically appears in the page admin.

The project code repository is found at this address: http://github.com/YD-Technology/django-basic-cms

1.1.1 Screenshot



1.1.2 Demo

This admin interface is no up to date, but could give you an idea of what the software is doing:

- admin : http://pagesdemo.piquadrat.ch/admin/
- frontend : http://pagesdemo.piquadrat.ch/pages/

1.1.3 Key features

- *Automatic creation of localized placeholders* (content area) in admin by adding placeholders tags into page templates.
- Django admin application integration.
- Multilingual support.
- Search indexation with Django haystack.
- Fine grained rights management (publisher, hierarchy manager, language manager).
- *Rich Text Editors* are directly available.
- Page can be moved in the tree in a visual way.
- The tree can be expanded/collapsed. A cookie remember your preferences.
- Possibility to specify a different page URL for each language.
- The frontend example provide a basic "edit in place" feature.
- Directory-like page hierarchy (page can have the same name if they are not in the same directory).
- Every page can have multiple alias URLs. It's especially useful to migrate websites.
- *Possibility to integrate 3th party apps*.

1.1.4 Other features

Here is the list of features you can enable/disable:

- Revisions,
- Image placeholder,
- File browser with django-filebrowser,
- Support for future publication start/end date,
- Page redirection to another page,
- Page tagging,
- User input sanitizer (to avoid XSS),
- Sites framework

1.1.5 Dependencies & Compatibility

- Django 1.1.1, Django 1.0 with older release (1.0.5)
- Python 2.3.
- django-haystack if used
- django-authority for per object rights management.
- django-mptt-2
- django-taggit (if PAGE_TAGGING = True)
- html5lib (if PAGE_SANITIZE_USER_INPUT = True)
- django-tinymce (if PAGE_TINYMCE = True)

- Django Basic CMS is shipped with jQuery.
- Django Basic CMS works well with django-staticfiles
- Compatible with MySQL, PostgreSQL, SQLite3, some issues are known with Oracle.

Note: For install instruction go to the *Installation section*

1.1.6 How to contribute

I recommend to create a clone on github and make your modifications in your branch. There is a things that is nice to do:

- Follow the pep08, and the 79 characters rules.
- Add new features in the *doc/changelog.rst* file.
- Document how the user might use a new feature.
- It's better if a new feature is not activated by default but with a new setting.
- Be careful of performance regresssion.
- Write tests so the test coverage stay over 90%.
- Every new CMS setting should start with PAGE_<something>
- Every new template_tag should start with pages_<something>

1.1.7 Ask for help

Django Basic CMS Github

Test it

To test this CMS checkout the code with git:

```
$ git clone git://github.com/YD-Technology/django-basic-cms.git django-basic-cms
```

Install the dependencies:

```
$ sudo easy_install pip
$ wget -c http://github.com/YD-Technology/django-basic-cms/raw/master/requirements/external_apps.txt
$ sudo pip install -r external_apps.txt
```

And then, run the development server:

```
$ cd example/
$ python manage.py syncdb
$ python manage.py build_static
$ python manage.py manage.py runserver
```

YD-Technology CMS try to keep the code base stable. The test coverage is higher than 80% and we try to keep it this way. To run the test suite:

```
python setup.py test
```

Note: If you are not admin you have to create the appropriate permissions to modify pages. After that you will be able to create pages.

Handling images and files

YD-Technology include a image placeholder for basic needs. For a more advanced files browser you could use django-filebrowser:

- http://code.google.com/p/django-filebrowser/

Once the application installed try to register the *FileBrowseInput* widget to make it available to your placeholders.

Translations

This application is available in English, German, French, Spanish, Danish, Russian and Hebrew.

1.2 Placeholders template tag

Contents

The placeholder template tag is what make Django Basic CMS special. The workflow is that you design your template first according to the page design. Then you put placeholder tag where you want dynamic content.

For each placeholder you will have a corresponding field appearing automaticaly in the administration interface. You can make as many templates as you want, even use the template inheritance: this CMS administration will still behave as intended.

The syntax for a placeholder tag is the following:

```
{% placeholder <name> [on <page>] [with <widget>] [parsed] [inherited] [as <varname>] %}
```

1.2.1 Detailed explanations on placeholder options

the on option

If the **on** option is omitted the CMS will automatically take the current page (by using the *current_page* context variable) to get the content of the placeholder.

Template syntax example:

```
{% placeholder main_menu on root_page %}
```

the widget option

The **widget** option is used to change the placeholders widgets in the administration interface.

By default the CMS will use a simple *TextInput* widget. Otherwise the CMS will use the widget of your choice. Widgets need to be registered before you can use them in the CMS:

```
from basic_cms.widgets_registry import register_widget
from django.forms import TextInput

class NewWidget(TextInput):
    pass

register_widget(NewWidget)
```

Template syntax example:

```
{% placeholder body with NewWidget %}
```

Note: This CMS is shipped with *a list of useful widgets* .

The as option

If you use the option **as** the content of the placeholder content will not be displayed: a variable of your choice will be defined within the template's context.

Template syntax example:

```
{% placeholder image as image_src %}
<img src="{{ img_src }}" alt=""/>
```

The parsed keyword

If you add the keyword **parsed** the content of the placeholder will be evaluated as Django template, within the current context. Each placeholder with the **parsed** keyword will also have a note in the admin interface noting its ability to be evaluated as template.

Template syntax example:

```
{% placeholder special-content parsed %}
```

The inherited keyword

If you add the keyword **inherited** the placeholder's content displayed on the frontend will be retrieved from the closest parent. But only if there is no content for the current page.

Template syntax example:

```
{% placeholder right-column inherited %}
```

The untranslated keyword

If you add the keyword **untranslated** the placeholder's content will be the same whatever language your use. It's especialy useful for an image placeholder that should remain the same in every language.

Template syntax example:

```
{% placeholder logo untranslated %}
```

Examples of other valid syntaxes

This is a list of different possible syntaxes for this template tag:

```
{% placeholder title with TextIntput %}
{% placeholder logo untranslated on root_page %}
{% placeholder right-column inherited as right_column parsed %}

...
<div class="my_funky_column">{{ right_column|safe }}</div>
```

1.2.2 Image placeholder

There is a special placeholder for images:

```
{% imageplaceholder body-image as imgsrc %}
{% if imgsrc %}
    <img src="{{ MEDIA_URL }}{{ imgsrc }}" alt=""/>
{% endif %}
```

A file upload field will appears into the page admin interface.

1.2.3 File placeholder

There is also a more general placeholder for files:

```
{% fileplaceholder uploaded_file as filesrc %}
{% if filesrc %}
    <a href="{{ MEDIA_URL }}{{ filesrc }}">Download file</a>
{% endif %}
```

A file upload field will appears into the page admin interface.

1.2.4 Contact placeholder

If you want to include a simple contact form in your page, there is a contact placeholder:

```
{% contactplaceholder contact %}
```

This placeholder use ´settings.ADMINS´ for recipients email. The template used to render the contact form is ´pages/contact.html´.

1.2.5 Create your own placeholder

If you want to create your own new type of placeholder, you can simply subclass the `PlaceholderNode`:

```
from basic_cms.placeholders import PlaceholderNode
from basic_cms.templatetags.page_tags import parse_placeholder
register = template.Library()

class ContactFormPlaceholderNode(PlaceholderNode):

    def __init__(self, name, *args, **kwargs):
        ...

    def get_widget(self, page, language, fallback=Textarea):
        """Redefine this to change the widget of the field."""
        ...

    def get_field(self, page, language, initial=None):
        """Redefine this to change the field displayed in the admin."""
        ...

    def save(self, page, language, data, change):
        """Redefine this to change the way to save the placeholder data."""
        ...

    def render(self, context):
        """Output the content of the node in the template."""
        ...

def do_contactplaceholder(parser, token):
    name, params = parse_placeholder(parser, token)
    return ContactFormPlaceholderNode(name, **params)
register.tag('contactplaceholder', do_contactplaceholder)
```

And use it in your templates as a normal placeholder:

```
{% contactplaceholder contact %}
```

1.2.6 Changing the widget of the common placeholder

If you want to just redefine the widget of the default `PlaceholderNode` without subclassing it, you can just you create a valid Django Widget that take an extra language paramater:

```
from django.forms import Textarea
from django.utils.safestring import mark_safe
from basic_cms.widgets_registry import register_widget

class CustomTextarea(Textarea):
    class Media:
        js = ['path to the widget extra javascript']
        css = {
            'all': ['path to the widget extra javascript']
        }

    def __init__(self, language=None, attrs=None, **kwargs):
        attrs = {'class': 'custom-textarea'}
        super(CustomTextarea, self).__init__(attrs)

    def render(self, name, value, attrs=None):
        rendered = super(CustomTextarea, self).render(name, value, attrs)
        return mark_safe("""Take a look at \
                example.widgets.CustomTextarea<br>""") \
                + rendered

register_widget(CustomTextarea)
```

Create a file named widgets (or whathever you want) somewhere in one of your project's application and then you can simply use the placeholder syntax:

```
{% placeholder custom_widget_example CustomTextarea parsed  %}
```

More examples of custom widgets are available in `pages.widgets.py`.

1.2.7 List of placeholder widgets shipped with the CMS

Placeholder could be rendered with different widgets

TextInput

A simple line input:

```
{% placeholder [name] with TextInput %}
```

Textarea

A multi line input:

```
{% placeholder [name] with Textarea %}
```

AdminTextInput

A simple line input with Django admin CSS styling (better for larger input fields):

```
{% placeholder [name] with AdminTextInput %}
```

AdminTextarea

A multi line input with Django admin CSS styling:

```
{% placeholder [name] with AdminTextarea %}
```

FileBrowseInput

A file browsing widget:

```
{% placeholder [name] with FileBrowseInput %}
```

Note: The following django application needs to be installed: http://code.google.com/p/django-filebrowser/

RichTextarea

A simple Rich Text Area Editor based on jQuery:

```
{% placeholder [name] with RichTextarea %}
```

WYMEditor

A complete jQuery Rich Text Editor called wymeditor:

```
{% placeholder [name] with WYMEditor %}
```

CKEditor

A complete JavaScript Rich Text Editor called CKEditor:

```
{% placeholder [name] with CKEditor %}
```

markItUpMarkdown

markdown editor based on markitup:

```
{% placeholder [name] with markItUpMarkdown %}
```

markItUpHTML

A HTML editor based on markitup:

```
{% placeholder [name] with markItUpHTML %}
```

TinyMCE

HTML editor based on TinyMCE

1. You should install the django-tinymce application first
2. Then in your settings you should activate the application:

   ```
   PAGE_TINYMCE = True
   ```

3. And add `tinymce` in your `INSTALLED_APPS` list.

The basic javascript files required to run TinyMCE are distributed with this CMS.

However if you want to use plugins you need to fully install TinyMCE. To do that follow carefully those install instructions

Usage:

```
{% placeholder [name] with TinyMCE %}
```

EditArea

Allows to edit raw html code with syntax highlight based on this project: http://www.cdolivet.com/index.php?page=editArea

Basic code (Javascript, CSS) for editarea is included into the codebase. If you want the full version you can get it there:

```
pages/media/pages/edit_area -r29 https://editarea.svn.sourceforge.net/svnroot/editarea/trunk/edit_are
```

Usage:

```
{% placeholder [name] with EditArea %}
```

1.3 Installation

This document explain how to install Django Basic CMS into an existing Django project. This document assume that you already know how to setup a Django project.

If you have any problem installing this CMS, take a look at the example application that stands in the example directory. This application works out of the box and will certainly help you to get started.

- Evaluate quickly the application
- Install dependencies by using pip
- Install dependencies by using easy_install
- Urls
- Settings

1.3.1 Evaluate quickly the application

After you have installed all the dependencies you can simply checkout the code with git:

```
git clone git://github.com/YD-Technology/django-basic-cms.git django-basic-cms
```

And then, run the example project:

```
cd django-basic-cms/example/
python manage.py syncdb
python manage.py build_static pages
python manage.py runserver
```

Then visit http://127.0.0.1:8000/admin/ and create a few pages.

1.3.2 Install dependencies by using pip

The pip install is the easiest and the recommended installation method. Use:

```
$ sudo easy_install pip
$ wget -c http://github.com/YD-Technology/django-basic-cms/raw/master/requirements/external_apps.txt
$ sudo pip install -r external_apps.txt
```

Every package listed in the `external_app.txt` should be downloaded and installed.

If you are not using the source code version of the application then install it using:

```
$ sudo pip install django-basic-cms
```

1.3.3 Install dependencies by using easy_install

On debian linux you can do:

```
$ sudo easy_install html5lib BeautifulSoup django django-staticfiles django-authority
```

Optionnaly:

```
$ sudo easy_install django-haystack
```

If you are not using the source code version of the application then install it using:

```
$ sudo easy_install django-page-cms
```

Note: Django-Tagging and Django-mptt maybe required to be installed by hand or with subversion because the available packages are not compatible with django 1.0.

1.3.4 Urls

Take a look in the `example/urls.py` and copy desired URLs in your own `urls.py`. Basically you need to have something like this:

```
urlpatterns = patterns('',
    ...
    url(r'^cms/', include('basic_cms.urls')),
```

```
    (r'^admin/', include(admin.site.urls)),
)
```

When you will visit the site the first time (`/cms/`), you will get a 404 error because there is no published page. Go to the admin first and create and publish some pages.

You will certainly want to activate the static file serve view in your `urls.py` if you are in developement mode:

```
if settings.DEBUG:
    urlpatterns += patterns('',
        # Trick for Django to support static files (security hole: only for Dev environement! remove
        url(r'^media/(?P<path>.*)$', 'django.views.static.serve', {'document_root': settings.MEDIA_R
        url(r'^admin_media/(?P<path>.*)$', 'django.views.static.serve', {'document_root': settings.AI
    )
```

1.3.5 Settings

All the Django Basic CMS specific settings and options are listed and explained in the `pages/settings.py` file.

Django Basic CMS require several of these settings to be set. They are marked in this document with a bold "*must*".

Note: If you want a complete list of the available settings for this CMS visit *the list of all available settings*.

Default template

You *must* set `PAGE_DEFAULT_TEMPLATE` to the path of your default CMS template:

```
PAGE_DEFAULT_TEMPLATE = 'pages/index.html'
```

This template must exist somewhere in your project. If you want you can copy the example templates from the directory `pages/templates/pages/examples/` into the directory `page` of your root template directory.

Additional templates

Optionally you can set `PAGE_TEMPLATES` if you want additional templates choices. In the the example application you have actually this:

```
PAGE_TEMPLATES = (
    ('pages/nice.html', 'nice one'),
    ('pages/cool.html', 'cool one'),
)
```

Media directory

The django CMS come with some javascript and CSS files. These files are standing in the `pages/media/pages` directory.

To make these files accessible to your project you can simply copy them or make a symbolic link into your media directory. That's necessary to have a fully functioning administration interface.

You can also look at how the example project is working to make a local setup. It use the very good django-staticfiles application that can gather the media files for you. After installation in your project just run:

```
$ python manage.py build_static pages
```

And the cms media files will be copied in your project's media directory.

Languages

Please first read how django handle languages

- http://docs.djangoproject.com/en/dev/ref/settings/#languages
- http://docs.djangoproject.com/en/dev/ref/settings/#language-code

This CMS use the `PAGE_LANGUAGES` setting in order to present which language are supported by the CMS.

Django itself use the `LANGUAGES` setting to set the `request.LANGUAGE_CODE` value that is used by this CMS. So if the language you want to support is not present in the `LANGUAGES` setting the `request.LANGUAGE_CODE` will not be set correctly.

A possible solution is to redefine `settings.LANGUAGES`. For example you can do:

```
# Default language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = 'en-us'

# This is defined here as a do-nothing function because we can't import
# django.utils.translation -- that module depends on the settings.
gettext_noop = lambda s: s

# here is all the languages supported by the CMS
PAGE_LANGUAGES = (
    ('de', gettext_noop('German')),
    ('fr-ch', gettext_noop('Swiss french')),
    ('en-us', gettext_noop('US English')),
)

# copy PAGE_LANGUAGES
languages = list(PAGE_LANGUAGES)

# redefine the LANGUAGES setting in order to be sure to have the correct request.LANGUAGE_CODE
LANGUAGES = languages
```

Template context processors and Middlewares

You *must* have these context processors into your `TEMPLATE_CONTEXT_PROCESSORS` setting:

```
TEMPLATE_CONTEXT_PROCESSORS = (
    'django.core.context_processors.auth',
    'django.core.context_processors.i18n',
    'django.core.context_processors.debug',
    'django.core.context_processors.media',
    'django.core.context_processors.request',
    'basic_cms.context_processors.media',
    ...
)
```

You *must* have these middleware into your `MIDDLEWARE_CLASSES` setting:

```
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.middleware.doc.XViewMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    ...
)
```

Caching

Django Basic CMS use the caching framework quite intensively. You should definitely setting-up a cache-backend to have decent performance.

You can easily setup a local memory cache this way:

```
CACHE_BACKEND = "locmem:///?max_entries=5000"
```

The sites framework

If you want to use the Django sites framework with Django Basic CMS, you *must* define the `SITE_ID` and `PAGE_USE_SITE_ID` settings and create the appropriate Site object into the admin interface:

```
PAGE_USE_SITE_ID = True
SITE_ID = 1
```

The Site object should have the domain that match your actual domain (ie: 127.0.0.1:8000)

Tagging

Tagging is optional and disabled by default.

If you want to use it set `PAGE_TAGGING` at `True` into your setting file and add it to your installed apps:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.admin',
    'django.contrib.sites',
    'mptt',
    'tagging',
    'basic_cms',
    ...
)
```

1.4 Display page's content in templates

Django Basic CMS provide several template tags to extract data from the CMS. To use these tags in your templates you must load them first:

```
{% load pages_tags %}
```

1.4.1 get_content

Store a content type from a page into a context variable that you can reuse after:

```
{% get_content current_page "title" as content %}
```

You can also use the slug of a page:

```
{% get_content "my-page-slug" "title" as content %}
```

You can also use the id of a page:

```
{% get_content 10 "title" as content %}
```

Note: You can use either the page object, the slug, or the id of the page.

1.4.2 show_content

Output the content of a page directly within the template:

```
{% show_content current_page "title" %}
```

Note: You can use either the page object, the slug, or the id of the page.

1.4.3 get_page

Retrieve a Page object and store it into a context variable that you can reuse after. Here is an example of the use of this template tag to display a list of news:

```
<h2>Latest news</h2>
{% get_page "news" as news_page %}
<ul>
{% for new in news_page.get_children %}
<li>
    <h3>{{ new.title }}</h3>
    {{ new.publication_date }}
    {% show_content new body %}
</li>
{% endfor %}
</ul>
```

Note: You can use either the slug, or the id of the page.

1.4.4 show_absolute_url

This tag show the absolute url of a page. The difference with the *Page.get_url_path* method is that the template knows which language is used within the context and display the URL accordingly:

```
{% show_absolute_url current_page %}
```

Note: You can use either the page object, the slug, or the id of the page.

1.4.5 Display content from other applications

There is several ways to change the way the default view provided by the CMS render the pages. This list try explain the most common.

Using the PAGE_EXTRA_CONTEXT setting

Considering you have a simple news model:

```
class News(models.Model):
    title = models.CharField(max_length=200)
    postdate = models.DateTimeField(default=datetime.now)
    body = models.CharField(max_length=200)
```

And that you would like to display a list of news into some of your page's templates:

```
<ul>
{% for new in news %}
    <li>
        <h2>{{ news.title }}</p>
        <p>{{ news.body }}</p>
    </li>
{% endfor %}
</ul>
```

Then you might want to use the *PAGE_EXTRA_CONTEXT* setting. You should set this setting to be a function. This function should return a Python dictionary. This dictionary will be merged with the context of every page of your website.

Example in the case of the news:

```
def extra_context():
    from news.models import News
    lastest_news = News.object.all()
    return {'news': lastest_news}

PAGE_EXTRA_CONTEXT = extra_context
```

Delegate the page rendering to another application

You can set another application to render certain pages of your website.

Subclass the default view

New in 1.3.0: The default view is now a real class. That will help if you want to override some default behavior:

```
from basic_cms.views import Details
from news.models import News

class NewsView(Details):

    def extra_context(self, request, context):
        lastest_news = News.object.all()
        return {'news': lastest_news}

details = NewsView()
```

And don't forget to redefine the urls to point to your new view with something similar to this code:

```
from django.conf.urls.defaults import url, include, patterns
from YOUR_APP.views import details
from basic_cms import page_settings

if page_settings.PAGE_USE_LANGUAGE_PREFIX:
    urlpatterns = patterns('',
        url(r'^(?P<lang>[-\w]+)/(?P<path>.*)$', details,
            name='pages-details-by-path')
    )
else:
    urlpatterns = patterns('',
        url(r'^(?P<path>.*)$', details, name='pages-details-by-path')
    )
```

1.5 How to use the various navigation template tags

Presenting a navigational structure to the user is an common task on every website. Django Basic CMS offers various template tags which can be used to create a site navigation menu.

- pages_menu
- pages_dynamic_tree_menu
- pages_sub_menu
- pages_siblings_menu
- pages_breadcrumb
- load_pages

1.5.1 pages_menu

The pages_menu tag displays the whole navigation tree, including all subpages. This is useful for smaller sites which do not have a large number of pages.

Use the following snippet in your template:

```
<ul>
{% for page in pages_navigation %}
    {% pages_menu page %}
{% endfor %}
</ul>
```

The pages_menu tag uses the *pages/menu.html* template to render the navigation menu. By default, the menu is rendered as a nested list:

```
<ul>
    <li><a href="/page/1">page1</a></li>
    ...
</ul>
```

You can of course change *pages/menu.html* with the Django override mechanism to render things differently.

1.5.2 pages_dynamic_tree_menu

The pages_dynamic_tree_menu tag works similar to the pages_menu tag. but instead of displaying the whole navigation structure, only the following pages are displayed:

- all "root" pages (pages which have no parent)
- all parents of the current page
- all direct children of the current page

This type of navigation is recommended if your site has a large number of pages and/or a deep hierarchy, which is too complex or large to be presented to the user at once.

Use the following snippet in your template:

```
<ul>
{% for page in pages_navigation %}
    {% pages_dynamic_tree_menu page %}
{% endfor %}
</ul>
```

The pages_dynamic_tree_menu tag uses the *pages/dynamic_tree_menu.html* template to render the navigation menu. By default, the menu is rendered as a nested list similar to the pages_menu tag.

1.5.3 pages_sub_menu

The pages_sub_menu tag shows all the children of the root of the current page (the highest in the hierarchy). This is typically used for a secondary navigation menu.

Use the following snippet to display a list of all the children of the current root:

```
<ul>
{% pages_sub_menu current_page %}
</ul>
```

Again, the default template *pages/sub_menu.html* will render the items as a nested, unordered list (see above).

1.5.4 pages_siblings_menu

The pages_siblings_menu tag shows all the children of the immediate parent of the current page. This can be used for example as a secondary menu.

Use the following snippet to display a list of all the children of the immediate parent of the current page:

```
<ul>
{% pages_siblings_menu current_page %}
</ul>
```

Again, the default template *pages/sub_menu.html* will render the items as a nested, unordered list (see above).

1.5.5 pages_breadcrumb

With the pages_breadcrumb tag, it is possible to use the "breadcrumb"/"you are here" navigational pattern, consisting of a list of all parents of the current page:

```
{% pages_breadcrumb current_page %}
```

The output of the pages_breadcrumb tag is defined by the template *pages/breadcrumb.html*.

1.5.6 load_pages

The load_pages Tag can be used to load the navigational structure in views which are *not* rendered through page's own details() view. It will check the current template context and adds the pages and current_page variable to the context, if they are not present.

This is useful if you are using a common base template for your whole site, and want the pages menu to be always present, even if the actual content is not a page.

The load_pages does not take any parameters and must be placed before one of the menu-rendering tags:

```
{% load_pages %}
```

1.6 Integrate with other applications

1.6.1 Delegate the rendering of a page to an application

By delegating the rendering of a page to another application, you will be able to use customized views and still get all the CMS variables to render a proper navigation.

First you need a *urls.py* file that you can register to the CMS. It might look like this:

```
from django.conf.urls.defaults import *
from basic_cms.testproj.documents.views import document_view
from basic_cms.http import pages_view

urlpatterns = patterns('',
    url(r'^doc-(?P<document_id>[0-9]+)$', pages_view(document_view), name='document_details'),
    url(r'^$', pages_view(document_view), name='document_root'),
)
```

Note: The decorator *pages_view* call the CMS if the context variables *current_page* and *pages_navigation* are not present in the arguments.

It's not necessary to decorate your views if you only call them via the CMS or you don't need those variables.

Then you need to register the urlconf module of your application to use it within the admin interface. Put this code in you urls.py *before* admin.autodiscover(). Here is an example for a document application.:

```
from basic_cms.urlconf_registry import register_urlconf

register_urlconf('Documents', 'pages.testproj.documents.urls',
    label='Display documents')

# this need to be executed after the registry happened.
admin.autodiscover()
```

As soon as you have registered your *urls.py*, a new field will appear in the page administration. Choose the *Display documents*. The view used to render this page on the frontend is now choosen by *pages.testproj.documents.urls*.

Note: The path passed to your urlconf module is the remaining path available after the page slug. Eg: */pages/page1/doc-1* will become *doc-1*.

Note: If you want to have the reverse URLs with your delegated application, you will need to include your URLs into your main urls.py, eg:

```
(r'^pages/', include('pages.urls')),
...
(r'^pages/(?P<path>.*)', include('pages.testproj.documents.urls')),
```

Here is an example of a valid view from the documents application:

```
from django.shortcuts import render_to_response
from django.template import RequestContext
from basic_cms.testproj.documents.models import Document

def document_view(request, *args, **kwargs):
    context = RequestContext(request, kwargs)
    if kwargs.get('current_page', False):
        documents = Document.objects.filter(page=kwargs['current_page'])
        context['documents'] = documents
    if kwargs.has_key('document_id'):
        document = Document.objects.get(pk=int(kwargs['document_id']))
        context['document'] = document
    context['in_document_view'] = True
    return render_to_response('pages/examples/index.html', context)
```

The *document_view* will receive a bunch of extra parameters related to the CMS:

- *current_page* the page object,
- *path* the path used to reach the page,
- *lang* the current language,
- *pages_navigation* the list of pages used to render navigation.

Note: If the field doesn't appear within the admin interface make sure that your registry code is executed properly.

Note: Look at the testproj in the repository for an example on how to integrate an external application.

1.6.2 Integrate application models and forms into the page admin

If you don't want to sublass the PageAdmin class Basic CMS provides an alternative way to integrate external application into the page's administration interface.

For this you need an object with foreign key pointing to a page:

```
class Document(models.Model):
    "A dummy model used to illustrate the use of linked models in Basic CMS"

    title = models.CharField(_('title'), max_length=100, blank=False)
    text = models.TextField(_('text'), blank=True)

    # You need a foreign key to the page object, and it must be named page
    page = models.ForeignKey(Page)

class DocumentForm(ModelForm):
    class Meta:
        model = Document
```

After that you need to set up the PAGE_CONNECTED_MODELS into your settings similar to this one:

```
PAGE_CONNECTED_MODELS = [{
    'model':'documents.models.Document',
    'form':'documents.models.DocumentForm',
    'options':{
            'extra': 3,
            'max_num': 10,
        },
},]
```

When you edit a page, you should see a form to create/update/delete a Document object linked to this page.

1.6.3 Sitemaps

Django Basic CMS provide 2 sitemaps classes to use with Django sitemap framework:

```
from basic_cms.views import PageSitemap, MultiLanguagePageSitemap

(r'^sitemap\.xml$', 'django.contrib.sitemaps.views.sitemap',
    {'sitemaps': {'pages':PageSitemap}}),

# or for multi language:

(r'^sitemap\.xml$', 'django.contrib.sitemaps.views.sitemap',
    {'sitemaps': {'pages':MultiLanguagePageSitemap}})
```

The *PageSitemap* class provide a sitemap for every published page in the default language. The *MultiLanguagePageSitemap* is gonna create an extra entry for every other language.

1.7 Commands

Contents

1.7.1 Export pages content into translatable PO files

The pages CMS provide a command for those that would prefer to use PO files instead of the admin inteface to translate the pages content.

To export all the content from the published page into PO files you can execute this Django command:

```
$ python manage.py pages_export_po <path>
```

The files a created in the *poexport* directory if no path is provided.

After the translation is done, you can import back the changes with another command:

```
$ python manage.py pages_import_po <path>
```

1.8 List of all available settings

1.8.1 PAGE_TEMPLATES

PAGE_TEMPLATES is a list of tuples that specifies the which templates are available in the `pages` admin. Templates should be assigned in the following format:

```
PAGE_TEMPLATES = (
    ('pages/nice.html', 'nice one'),
    ('pages/cool.html', 'cool one'),
)
```

One can also assign a callable (which should return the tuple) to this setting to achieve dynamic template list e.g.:

```
def _get_templates():
    # to avoid any import issues
    from app.models import PageTemplate
    return PageTemplate.get_page_templates()

PAGE_TEMPLATES = _get_templates
```

Where the model might look like this:

```
class PageTemplate(OrderedModel):
    name = models.CharField(unique=True, max_length=100)
    template = models.CharField(unique=True, max_length=260)

    @staticmethod
    def get_page_templates():
        return PageTemplate.objects.values_list('template', 'name')
```

```
    class Meta:
        ordering = ["order"]

    def __unicode__(self):
        return self.name
```

1.8.2 PAGE_DEFAULT_TEMPLATE

You *must* set `PAGE_DEFAULT_TEMPLATE` to the path of your default template:

```
PAGE_DEFAULT_TEMPLATE = 'pages/index.html'
```

1.8.3 PAGE_LANGUAGES

A list tuples that defines the languages that pages can be translated into:

```
gettext_noop = lambda s: s

PAGE_LANGUAGES = (
    ('zh-cn', gettext_noop('Chinese Simplified')),
    ('fr-ch', gettext_noop('Swiss french')),
    ('en-us', gettext_noop('US English')),
)
```

1.8.4 PAGE_DEFAULT_LANGUAGE

Defines which language should be used by default. If `PAGE_DEFAULT_LANGUAGE` not specified, then project's `settings.LANGUAGE_CODE` is used:

```
LANGUAGE_CODE = 'en-us'
```

1.8.5 PAGE_LANGUAGE_MAPPING

PAGE_LANGUAGE_MAPPING must be a function that takes the language code of the incoming browser as an argument.

This function can change the incoming client language code to another language code, presumably one for which you are managing trough the CMS.

This is useful if your project only has one set of translation strings for a language like Chinese, which has several variants like `zh-cn`, `zh-tw`, `zh-hk`, but you don't have a translation for every variant.

PAGE_LANGUAGE_MAPPING help you to server the same Chinese translation to all those Chinese variants, not just those with the exact `zh-cn` locale.

Enable that behavior here by assigning the following function to the `PAGE_LANGUAGE_MAPPING` variable:

```
# here is all the languages supported by the CMS
PAGE_LANGUAGES = (
    ('de', gettext_noop('German')),
    ('fr-fr', gettext_noop('Swiss french')),
    ('en-us', gettext_noop('US English')),
)
```

```
# copy PAGE_LANGUAGES
languages = list(PAGE_LANGUAGES)

# Other languages accepted as a valid client language
languages.append(('fr-fr', gettext_noop('French')))
languages.append(('fr-be', gettext_noop('Belgium french')))

# redefine the LANGUAGES setting in order to be sure to have the correct request.LANGUAGE_CODE
LANGUAGES = languages

# Map every french based language to fr-fr
def language_mapping(lang):
    if lang.startswith('fr'):
        return 'fr-fr'
    return lang
PAGE_LANGUAGE_MAPPING = language_mapping
```

1.8.6 PAGES_MEDIA_URL

URL that handles pages media. If not set the default value is:

`<STATIC_URL|MEDIA_URL>pages/`

1.8.7 PAGE_UNIQUE_SLUG_REQUIRED

Set `PAGE_UNIQUE_SLUG_REQUIRED` to `True` to enforce unique slug names for all pages.

1.8.8 PAGE_CONTENT_REVISION

Set `PAGE_CONTENT_REVISION` to `False` to disable the recording of pages revision information in the database

1.8.9 SITE_ID

Set SITE_ID to the id of the default `Site` instance to be used on installations where content from a single installation is served on multiple domains via the `django.contrib.sites` framework.

1.8.10 PAGE_USE_SITE_ID

Set PAGE_USE_SITE_ID to `True` to make use of the `django.contrib.sites` framework

1.8.11 PAGE_USE_LANGUAGE_PREFIX

Set PAGE_USE_LANGUAGE_PREFIX to `True` to make the `get_absolute_url` method to prefix the URLs with the language code

1.8.12 PAGE_CONTENT_REVISION_EXCLUDE_LIST

Assign a list of placeholders to PAGE_CONTENT_REVISION_EXCLUDE_LIST to exclude them from the revision process.

1.8.13 PAGE_SANITIZE_USER_INPUT

Set `PAGE_SANITIZE_USER_INPUT` to `True` to sanitize the user input with `html5lib`.

1.8.14 PAGE_HIDE_ROOT_SLUG

Hide the slug's of the first root page ie: `/home/` becomes `/`

1.8.15 PAGE_SHOW_START_DATE

Show the publication start date field in the admin. Allows for future dating Changing the `PAGE_SHOW_START_DATE` from `True` to `False` after adding data could cause some weirdness. If you must do this, you should update your database to correct any future dated pages.

1.8.16 PAGE_SHOW_END_DATE

Show the publication end date field in the admin, allows for page expiration Changing `PAGE_SHOW_END_DATE` from `True` to `False` after adding data could cause some weirdness. If you must do this, you should update your database and null any pages with `publication_end_date` set.

1.8.17 PAGE_CONNECTED_MODELS

`PAGE_CONNECTED_MODELS` allows you to specify a model and form for this model into your settings to get an automatic form to create and directly link a new instance of this model with your page in the admin:

```
PAGE_CONNECTED_MODELS = [
    {'model':'documents.models.Document',
        'form':'documents.models.DocumentForm'},
]
```

Note: *Complete documentation on how to use this setting*

1.8.18 PAGE_LINK_FILTER

The page link filter enable a output filter on you content links. The goal is to transform special page classes into real links at the last moment. This ensure that even if you move a page within the CMS, the URLs pointing on it will remain correct.

1.8.19 PAGE_TINYMCE

Set this to `True` if you wish to use the `django-tinymce` application.

1.8.20 PAGE_EXTRA_CONTEXT

This setting is a function that can be defined if you need to pass extra context data to the pages templates.

1.9 Changelog

This file describe new features and incompatibilites between released version of the CMS.

1.9.1 Release 0.1.5

- Fix js

1.9.2 Release 0.1.4

- Fix setup

1.9.3 Release 0.1.3

- Fix migrations for custom user model

1.9.4 Release 0.1.2

- Set django-taggit as required
- Reset migrations

1.9.5 Release 0.1.1

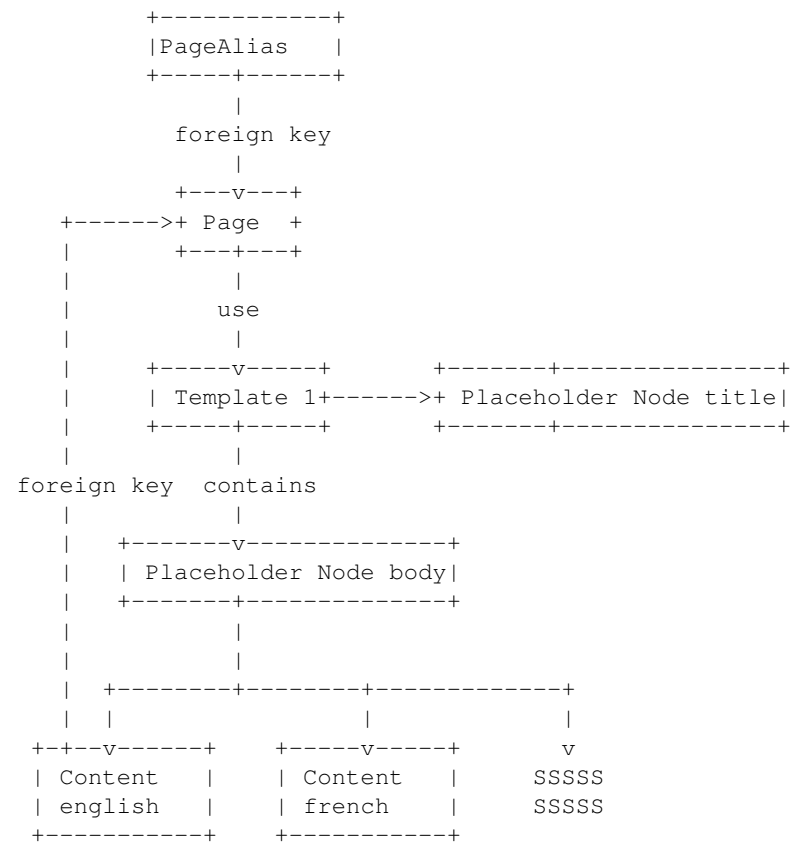
- Renamed to Django Basic CMS (basic_cms)
- Fixed tests (added travis + coveralls)
- Released on PyPI
- Released on readthedocs.org

1.10 Django Basic CMS reference API

1.10.1 The application model

Django Basic CMS declare rather simple models: `Page` `Content` and `PageAlias`.

Those Django models have the following relations:



1.10.2 Placeholders

Placeholder module, that's where the smart things happen.

class basic_cms.placeholders.**ContactForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=u':'*, *empty_permitted=False*)

base_fields = {'email': <django.forms.fields.EmailField object at 0x360d7d0>, 'subject': <django.forms.fields.CharFi

media

class basic_cms.placeholders.**ContactPlaceholderNode**(*name*, *page=None*, *widget=None*, *parsed=False*, *as_varname=None*, *inherited=False*, *untranslated=False*)

A contact *PlaceholderNode* example.

render(*context*)

class basic_cms.placeholders.**FilePlaceholderNode**(*name*, *page=None*, *widget=None*, *parsed=False*, *as_varname=None*, *inherited=False*, *untranslated=False*)

A *PlaceholderNode* that saves one file on disk.

PAGE_UPLOAD_ROOT setting define where to save the file.

get_field(*page*, *language*, *initial=None*)

save(*page*, *language*, *data*, *change*, *extra_data=None*)

class basic_cms.placeholders.**ImagePlaceholderNode**(*name*, *page=None*, *widget=None*, *parsed=False*, *as_varname=None*, *inherited=False*, *untranslated=False*)

A *PlaceholderNode* that saves one image on disk.

PAGE_UPLOAD_ROOT setting define where to save the image.

get_field(*page*, *language*, *initial=None*)

save(*page*, *language*, *data*, *change*, *extra_data=None*)

class basic_cms.placeholders.**PlaceholderNode**(*name*, *page=None*, *widget=None*, *parsed=False*, *as_varname=None*, *inherited=False*, *untranslated=False*)

This template node is used to output and save page content and dynamically generate input fields in the admin.

Parameters

- **name** – the name of the placeholder you want to show/create
- **page** – the optional page object
- **widget** – the widget you want to use in the admin interface. Take a look into basic_cms.widgets to see which widgets are available.
- **parsed** – if the parsed word is given, the content of the placeholder is evaluated as template code, within the current context.
- **as_varname** – if as_varname is defined, no value will be returned. A variable will be created in the context with the defined name.
- **inherited** – inherit content from parent's pages.
- **untranslated** – the placeholder's content is the same for every language.

field

alias of CharField

get_content(*page_obj*, *lang*, *lang_fallback=True*)

get_content_from_context(*context*)

get_extra_data(*data*)

Get eventual extra data for this placeholder from the admin form. This method is called when the Page is saved in the admin and passed to the placeholder save method.

get_field(*page*, *language*, *initial=None*)

The field that will be shown within the admin.

get_widget(*page*, *language*, *fallback=<class 'django.forms.widgets.Textarea'>*)

Given the name of a placeholder return a *Widget* subclass like Textarea or TextInput.

render(*context*)

Output the content of the *PlaceholdeNode* in the template.

save(*page*, *language*, *data*, *change*, *extra_data=None*)
Actually save the placeholder data into the Content object.

widget
alias of `TextInput`

class basic_cms.placeholders.**VideoPlaceholderNode**(*name*, *page=None*, *widget=None*, *parsed=False*, *as_varname=None*, *inherited=False*, *untranslated=False*)
A youtube *PlaceholderNode*, just here as an example.

render(*context*)

widget
alias of `VideoWidget`

basic_cms.placeholders.**parse_placeholder**(*parser*, *token*)
Parse the *PlaceholderNode* parameters.

Return a tuple with the name and parameters.

1.10.3 Template tags

Page CMS page_tags template tags

class basic_cms.templatetags.pages_tags.**FakeCSRFNode**
Fake CSRF node for django 1.1.1

class basic_cms.templatetags.pages_tags.**GetContentNode**(*page*, *content_type*, *varname*, *lang*, *lang_filter*)
Get content node

class basic_cms.templatetags.pages_tags.**GetPageNode**(*page_filter*, *varname*)
get_page Node

class basic_cms.templatetags.pages_tags.**LoadPagesNode**
Load page node.

basic_cms.templatetags.pages_tags.**do_contactplaceholder**(*parser*, *token*)
Method that parse the contactplaceholder template tag.

basic_cms.templatetags.pages_tags.**do_fileplaceholder**(*parser*, *token*)
Method that parse the fileplaceholder template tag.

basic_cms.templatetags.pages_tags.**do_get_content**(*parser*, *token*)
Retrieve a Content object and insert it into the template's context.

Example:

```
{% get_content page_object "title" as content %}
```

You can also use the slug of a page:

```
{% get_content "my-page-slug" "title" as content %}
```

Syntax:

```
{% get_content page type [lang] as name %}
```

Parameters

- **page** – the page object, slug or id

- **type** – content_type used by a placeholder
- **name** – name of the context variable to store the content in
- **lang** – the wanted language

basic_cms.templatetags.pages_tags.**do_get_page**(*parser*, *token*)
Retrieve a page and insert into the template's context.

Example:

```
{% get_page "news" as news_page %}
```

Parameters

- **page** – the page object, slug or id
- **name** – name of the context variable to store the page in

basic_cms.templatetags.pages_tags.**do_imageplaceholder**(*parser*, *token*)
Method that parse the imageplaceholder template tag.

basic_cms.templatetags.pages_tags.**do_load_pages**(*parser*, *token*)
Load the navigation pages, lang, and current_page variables into the current context.

Example:

```
<ul>
    {% load_pages %}
    {% for page in pages_navigation %}
        {% pages_menu page %}
    {% endfor %}
</ul>
```

basic_cms.templatetags.pages_tags.**do_placeholder**(*parser*, *token*)
Method that parse the placeholder template tag.

Syntax:

```
{% placeholder <name> [on <page>] [with <widget>] [parsed] [as <varname>] %}
```

Example usage:

```
{% placeholder about %}
{% placeholder body with TextArea as body_text %}
{% placeholder welcome with TextArea parsed as welcome_text %}
{% placeholder teaser on next_page with TextArea parsed %}
```

basic_cms.templatetags.pages_tags.**do_videoplaceholder**(*parser*, *token*)
Method that parse the imageplaceholder template tag.

basic_cms.templatetags.pages_tags.**get_page_from_string_or_id**(*page_string*, *lang=None*)
Return a Page object from a slug or an id.

basic_cms.templatetags.pages_tags.**has_content_in**(*page*, *language*)
Fitler that return `True` if the page has any content in a particular language.

Parameters

- **page** – the current page
- **language** – the language you want to look at

basic_cms.templatetags.pages_tags.**language_content_up_to_date**(*page*, *language*)
Tell if all the page content has been updated since the last change of the official version (settings.LANGUAGE_CODE)

This is approximated by comparing the last modified date of any content in the page, not comparing each content block to its corresponding official language version. That allows users to easily make "do nothing" changes to any content block when no change is required for a language.

basic_cms.templatetags.pages_tags.**pages_admin_menu**(*context*, *page*)
Render the admin table of pages.

basic_cms.templatetags.pages_tags.**pages_breadcrumb**(*context*, *page*, *url='/'*)
Tags

basic_cms.templatetags.pages_tags.**pages_dynamic_tree_menu**(*context*, *page*, *url='/'*)
Render a "dynamic" tree menu, with all nodes expanded which are either ancestors or the current page itself.

Override `pages/dynamic_tree_menu.html` if you want to change the design.

Parameters

- **page** – the current page
- **url** – not used anymore

basic_cms.templatetags.pages_tags.**pages_menu**(*context*, *page*, *url='/'*)
Render a nested list of all the descendents of the given page, including this page.

Parameters

- **page** – the page where to start the menu from.
- **url** – not used anymore.

basic_cms.templatetags.pages_tags.**pages_siblings_menu**(*context*, *page*, *url='/'*)
Get the parent page of the given page and render a nested list of its child pages. Good for rendering a secondary menu.

Parameters

- **page** – the page where to start the menu from.
- **url** – not used anymore.

basic_cms.templatetags.pages_tags.**pages_sub_menu**(*context*, *page*, *url='/'*)
Get the root page of the given page and render a nested list of all root's children pages. Good for rendering a secondary menu.

Parameters

- **page** – the page where to start the menu from.
- **url** – not used anymore.

basic_cms.templatetags.pages_tags.**show_absolute_url**(*context*, *page*, *lang=None*)
Show the url of a page in the right language

Example

```
{% show_absolute_url page_object %}
```

You can also use the slug of a page:

```
{% show_absolute_url "my-page-slug" %}
```

Keyword arguments: :param page: the page object, slug or id :param lang: the wanted language (defaults to *settings.PAGE_DEFAULT_LANGUAGE*)

basic_cms.templatetags.pages_tags.**show_content**(*context*, *page*, *content_type*, *lang=None*, *fallback=True*)

Display a content type from a page.

Example:

```
{% show_content page_object "title" %}
```

You can also use the slug of a page:

```
{% show_content "my-page-slug" "title" %}
```

Or even the id of a page:

```
{% show_content 10 "title" %}
```

Parameters

- **page** – the page object, slug or id
- **content_type** – content_type used by a placeholder
- **lang** – the wanted language (default None, use the request object to know)
- **fallback** – use fallback content from other language

basic_cms.templatetags.pages_tags.**show_revisions**(*context*, *page*, *content_type*, *lang=None*)

Render the last 10 revisions of a page content with a list using the `pages/revisions.html` template

basic_cms.templatetags.pages_tags.**show_slug_with_level**(*context*, *page*, *lang=None*, *fallback=True*)

Display slug with level by language.

1.10.4 Widgets

Django CMS come with a set of ready to use widgets that you can enable in the admin via a placeholder tag in your template.

class basic_cms.widgets.**CKEditor**(*language=None*, *attrs=None*, ***kwargs*)

CKEditor widget.

class Media

js = ['Nonepages/ckeditor/ckeditor.js', 'filebrowser/js/FB_CKEditor.js']

CKEditor.**media**

CKEditor.**render**(*name*, *value*, *attrs=None*, ***kwargs*)

class basic_cms.widgets.**EditArea**(*language=None*, *attrs=None*, ***kwargs*)

EditArea is a html syntax coloured widget.

class Media

js = ['Nonepages/edit_area/edit_area_full.js']

path = 'edit_area/edit_area_full.js'

EditArea.**media**

EditArea.**render**(*name*, *value*, *attrs=None*, ***kwargs*)

class basic_cms.widgets.**FileInput**(*page=None*, *language=None*, *attrs=None*, ***kwargs*)

media

render(*name*, *value*, *attrs=None*, ***kwargs*)

class basic_cms.widgets.**ImageInput**(*page=None*, *language=None*, *attrs=None*, ***kwargs*)

media

render(*name*, *value*, *attrs=None*, ***kwargs*)

class basic_cms.widgets.**LanguageChoiceWidget**(*language=None*, *attrs=None*, ***kwargs*)

media

render(*name*, *value*, *attrs=None*, ***kwargs*)

class basic_cms.widgets.**RichTextarea**(*language=None*, *attrs=None*, ***kwargs*)
A RichTextarea widget.

class Media

css = {'all': ['Nonepages/css/rte.css']}

js = ['Nonepages/javascript/jquery.js']

path = 'css/rte.css'

RichTextarea.**media**

RichTextarea.**render**(*name*, *value*, *attrs=None*, ***kwargs*)

class basic_cms.widgets.**VideoWidget**(*attrs=None*, *page=None*, *language=None*, *video_url=None*, *weight=None*, *height=None*)
A youtube *Widget* for the admin.

decompress(*value*)

format_output(*rendered_widgets*)
Given a list of rendered widgets (as strings), it inserts an HTML linebreak between them.

Returns a Unicode string representing the HTML for the whole lot.

media

value_from_datadict(*data*, *files*, *name*)

class basic_cms.widgets.**WYMEditor**(*language=None*, *attrs=None*, ***kwargs*)
WYMEditor widget.

class Media

js = ['Nonepages/javascript/jquery.js', 'Nonepages/javascript/jquery.ui.js', 'Nonepages/javascript/jquery.ui.resizabl

path = 'wymeditor/plugins/resizable/jquery.wymeditor.resizable.js'

WYMEditor.**media**

WYMEditor.**render**(*name*, *value*, *attrs=None*, ***kwargs*)

class basic_cms.widgets.**markItUpHTML**(*attrs=None*)
markItUpHTML widget.

class Media

css = {'all': ['Nonepages/markitup/skins/simple/style.css', 'Nonepages/markitup/sets/default/style.css']}

js = ['Nonepages/javascript/jquery.js', 'Nonepages/markitup/jquery.markitup.js', 'Nonepages/markitup/sets/defaul

path = 'markitup/sets/default/style.css'

markItUpHTML.**media**

markItUpHTML.**render**(*name*, *value*, *attrs=None*, ***kwargs*)

class basic_cms.widgets.**markItUpMarkdown**(*attrs=None*)
markItUpMarkdown widget.

class Media

css = {'all': ['Nonepages/markitup/skins/simple/style.css', 'Nonepages/markitup/sets/markdown/style.css']}

js = ['Nonepages/javascript/jquery.js', 'Nonepages/markitup/jquery.markitup.js', 'Nonepages/markitup/sets/markd

path = 'markitup/sets/markdown/style.css'

markItUpMarkdown.**media**

markItUpMarkdown.**render**(*name*, *value*, *attrs=None*, ***kwargs*)

class basic_cms.widgets.**markItUpRest**(*attrs=None*)
markItUpRest widget.

class Media

css = {'all': ['Nonepages/markitup/skins/simple/style.css', 'Nonepages/markitup/sets/rst/style.css']}

js = ['Nonepages/javascript/jquery.js', 'Nonepages/markitup/jquery.markitup.js', 'Nonepages/markitup/sets/rst/set.

path = 'markitup/sets/rst/style.css'

markItUpRest.**media**

markItUpRest.**render**(*name*, *value*, *attrs=None*, ***kwargs*)

1.10.5 Page Model

class basic_cms.models.**Page**(**args*, ***kwargs*)
This model contain the status, dates, author, template. The real content of the page can be found in the Content model.

creation_date
When the page has been created.

publication_date
When the page should be visible.

publication_end_date
When the publication of this page end.

last_modification_date
Last time this page has been modified.

status
The current status of the page. Could be DRAFT, PUBLISHED, EXPIRED or HIDDEN. You should the property :attr:`calculated_status` if you want that the dates are taken in account.

template
A string containing the name of the template file for this page.

calculated_status
Get the calculated status of the page based on `Page.publication_date`, `Page.publication_end_date`, and `Page.status`.

content_by_language(*language*)
Return a list of latest published `Content` for a particluar language.

Parameters **language** – wanted language,

dump_json_data()
Return a python dict representation of this page for use as part of a JSON export.

expose_content()
Return all the current content of this page into a *string*.

This is used by the haystack framework to build the search index.

get_absolute_url(**moreargs*, ***morekwargs*)
Alias for *get_url_path*.

This method is only there for backward compatibility and will be removed in a near futur.

Parameters **language** – the wanted url language.

get_children_for_frontend()
Return a `QuerySet` of published children page

get_complete_slug(*language=None*, *hideroot=True*)
Return the complete slug of this page by concatenating all parent's slugs.

Parameters **language** – the wanted slug language.

get_content(*language*, *ctype*, *language_fallback=False*)
Shortcut method for retrieving a piece of page content

Parameters

- **language** – wanted language, if not defined default is used.
- **ctype** – the type of content.
- **fallback** – if `True`, the content will also be searched in other languages.

get_date_ordered_children_for_frontend()
Return a `QuerySet` of published children page ordered by publication date.

get_languages()
Return a list of all used languages for this page.

get_template()
Get the `template` of this page if defined or the closer parent's one if defined or `pages.settings.PAGE_DEFAULT_TEMPLATE` otherwise.

get_template_name()
Get the template name of this page if defined or if a closer parent has a defined template or `pages.settings.PAGE_DEFAULT_TEMPLATE` otherwise.

get_url(*language=None*)
Alias for *get_complete_slug*.

This method is only there for backward compatibility and will be removed in a near futur.

Parameters **language** – the wanted url language.

get_url_path(*language=None*)
Return the URL's path component. Add the language prefix if `PAGE_USE_LANGUAGE_PREFIX` setting is set to `True`.

Parameters **language** – the wanted url language.

has_broken_link()
Return `True` if the page have broken links to other pages into the content.

invalidate()
Invalidate cached data for this page.

is_first_root()
Return `True` if this page is the first root pages.

margin_level()
Used in the admin menu to create the left margin.

save(**args*, ***kwargs*)
Override the default `save` method.

slug(*language=None*, *fallback=True*)
Return the slug of the page depending on the given language.

Parameters

- **language** – wanted language, if not defined default is used.
- **fallback** – if `True`, the slug will also be searched in other languages.

slug_with_level(*language=None*)
Display the slug of the page prepended with insecable spaces equal to simluate the level of page in the hierarchy.

title(*language=None*, *fallback=True*)
Return the title of the page depending on the given language.

Parameters

- **language** – wanted language, if not defined default is used.
- **fallback** – if `True`, the slug will also be searched in other languages.

update_redirect_to_from_json(*redirect_to_complete_slugs*)
The second pass of PageManager.create_and_update_from_json_data used to update the redirect_to field.

Returns a messages list to be appended to the messages from the first pass.

valid_targets()
Return a `QuerySet` of valid targets for moving a page into the tree.

Parameters **perms** – the level of permission of the concerned user.

visible
Return True if the page is visible on the frontend.

1.10.6 Page Manager

class basic_cms.managers.**PageManager**

Page manager provide several filters to obtain pages QuerySet that respect the page attributes and project settings.

create_and_update_from_json_data(*d*, *user*)

Create or update page based on python dict d loaded from JSON data. This applies all data except for redirect_to, which is done in a second pass after all pages have been imported,

user is the User instance that will be used if the author can't be found in the DB.

returns (page object, created, messages).

created is True if this was a new page or False if an existing page was updated.

messages is a list of strings warnings/messages about this import

drafts()

Creates a QuerySet of drafts using the page's Page.publication_date.

expired()

Creates a QuerySet of expired using the page's Page.publication_end_date.

filter_published(*queryset*)

Filter the given pages QuerySet to obtain only published page.

from_path(*complete_path*, *lang*, *exclude_drafts=True*)

Return a Page according to the page's path.

hidden()

Creates a QuerySet of the hidden pages.

navigation()

Creates a QuerySet of the published root pages.

on_site(*site_id=None*)

Return a QuerySet of pages that are published on the site defined by the SITE_ID setting.

Parameters **site_id** – specify the id of the site object to filter with.

populate_pages(*parent=None*, *child=5*, *depth=5*)

Create a population of Page for testing purpose.

published()

Creates a QuerySet of published Page.

root()

Return a QuerySet of pages without parent.

1.10.7 Page view

class basic_cms.views.**Details**

This class based view get the root pages for navigation and the current page to display if there is any.

All is rendered with the current page's template.

choose_language(*lang*, *request*)

Deal with the multiple corner case of choosing the language.

extra_context(*request*, *context*)

Call the PAGE_EXTRA_CONTEXT function if there is one.

get_navigation(*request*, *path*, *lang*)
Get the pages that are at the root level.

get_template(*request*, *context*)
Just there in case you have special business logic.

is_user_staff(*request*)
Return True if the user is staff.

resolve_page(*request*, *context*, *is_staff*)
Return the appropriate page according to the path.

resolve_redirection(*request*, *context*)
Check for redirections.

1.10.8 Content Model

class basic_cms.models.**Content**(**args*, ***kwargs*)
A block of content, tied to a Page, for a particular language

exception DoesNotExist

exception Content.**MultipleObjectsReturned**

Content.**get_next_by_creation_date**(**moreargs*, ***morekwargs*)

Content.**get_previous_by_creation_date**(**moreargs*, ***morekwargs*)

Content.**objects = <basic_cms.managers.ContentManager object at 0x3606e90>**

Content.**page**

1.10.9 Content Manager

class basic_cms.managers.**ContentManager**
Content manager methods

PAGE_CONTENT_DICT_KEY = 'page_content_dict_%d_%s_%d'

create_content_if_changed(*page*, *language*, *ctype*, *body*)
Create a Content for a particular page and language only if the content has changed from the last time.

Parameters

- **page** – the concerned page object.
- **language** – the wanted language.
- **ctype** – the content type.
- **body** – the content of the Content object.

get_content(*page*, *language*, *ctype*, *language_fallback=False*)
Gets the latest content string for a particular page, language and placeholder.

Parameters

- **page** – the concerned page object.
- **language** – the wanted language.
- **ctype** – the content type.
- **language_fallback** – fallback to another language if True.

get_content_object(*page*, *language*, *ctype*)
Gets the latest published `Content` for a particular page, language and placeholder type.

get_content_slug_by_slug(*slug*)
Returns the latest `Content` slug object that match the given slug for the current site domain.

Parameters **slug** – the wanted slug.

get_page_ids_by_slug(*slug*)
Return all page's id matching the given slug. This function also returns pages that have an old slug that match.

Parameters **slug** – the wanted slug.

sanitize(*content*)
Sanitize a string in order to avoid possible XSS using `html5lib`.

set_or_create_content(*page*, *language*, *ctype*, *body*)
Set or create a `Content` for a particular page and language.

Parameters

- **page** – the concerned page object.
- **language** – the wanted language.
- **ctype** – the content type.
- **body** – the content of the Content object.

1.10.10 PageAlias Model

class basic_cms.models.**PageAlias**(**args*, ***kwargs*)
URL alias for a `Page`

exception DoesNotExist

exception PageAlias.**MultipleObjectsReturned**

PageAlias.**objects = <basic_cms.managers.PageAliasManager object at 0x360d4d0>**

PageAlias.**page**

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

1.10.11 PageAlias Manager

class basic_cms.managers.**PageAliasManager**
`PageAlias` manager.

from_path(*request*, *path*, *lang*)
Resolve a request to an alias. returns a `PageAlias` if the url matches no page at all. The aliasing system supports plain aliases (`/foo/bar`) as well as aliases containing GET parameters (like `index.php?page=foo`).

Parameters

- **request** – the request object
- **path** – the complete path to the page
- **lang** – not used

1.10.12 Utils

A collection of functions for Page CMS

basic_cms.utils.**export_po_files**(*path='poexport'*, *stdout=None*)
Export all the content from the published pages into po files. The files will be automatically updated with the new content if you run the command again.

basic_cms.utils.**filter_link**(*content*, *page*, *language*, *content_type*)
Transform the HTML link href to point to the targeted page absolute URL.

```
>>> filter_link('<a class="page_1">hello</a>', page, 'en-us', body)
'<a href="/pages/page-1" class="page_1">hello</a>'
```

basic_cms.utils.**get_placeholders**(*template_name*)
Return a list of PlaceholderNode found in the given template.

Parameters **template_name** – the name of the template file

basic_cms.utils.**import_po_files**(*path='poexport'*, *stdout=None*)
Import all the content updates from the po files into the pages.

basic_cms.utils.**normalize_url**(*url*)
Return a normalized url with trailing and without leading slash.

```
>>> normalize_url(None)
'/'
>>> normalize_url('/')
'/'
>>> normalize_url('/foo/bar')
'/foo/bar'
>>> normalize_url('foo/bar')
'/foo/bar'
>>> normalize_url('/foo/bar/')
'/foo/bar'
```

basic_cms.utils.**now_utc**()

1.10.13 Http

Page CMS functions related to the `request` object.

exception basic_cms.http.**AutoRenderHttpError**
Cannot return context dictionary because a view returned an `HttpResponse` when a (template_name, context) tuple was expected.

basic_cms.http.**auto_render**(*func*)
This view decorator automatically calls the `render_to_response` shortcut. A view that use this decorator should return a tuple of this form : (template name, context) instead of a `HttpRequest` object.

basic_cms.http.**get_language_from_request**(*request*)
Return the most obvious language according the request.

basic_cms.http.**get_request_mock**()
Build a `request` mock up that is used in to render the templates in the most fidel environement as possible.

This fonction is used in the get_placeholders method to render the input template and search for the placeholder within.

basic_cms.http.**get_slug**(*path*)
Return the page's slug

```
>>> get_slug('/test/function/')
function
```

basic_cms.http.**get_template_from_request**(*request*, *page=None*)
Gets a valid template from different sources or falls back to the default template.

basic_cms.http.**pages_view**(*view*)
Make sure the decorated view gets the essential pages variables.

basic_cms.http.**remove_slug**(*path*)
Return the remainin part of the path

```
>>> remove_slug('/test/some/function/')
test/some
```

1.10.14 Admin views

Pages admin views

basic_cms.admin.views.**change_status**(*request*, **args*, ***kwargs*)
Switch the status of a page.

basic_cms.admin.views.**delete_content**(*request*, **args*, ***kwargs*)

basic_cms.admin.views.**modify_content**(*request*, **args*, ***kwargs*)
Modify the content of a page.

basic_cms.admin.views.**move_page**(**args*, ***kwargs*)
Move the page to the requested target, at the given position.

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

b