django-blog-zinnia Documentation

Release 0.19.dev0

Fantomas42

Aug 14, 2017

Contents

1 Django Blog Zinnia **3**
- 1.1 Features . . . 3
- 1.2 Examples . . . 4
- 1.3 Online resources . . . 4

2 Getting Started **7**
- 2.1 Installation . . . 7
- 2.2 Advanced Configuration . . . 10
- 2.3 Upgrading Zinnia . . . 12

3 Topics **15**
- 3.1 Channels . . . 15
- 3.2 Search Engines . . . 15
- 3.3 URL Shortener . . . 17
- 3.4 Spam Checker . . . 18
- 3.5 Permissions . . . 19
- 3.6 Ecosystem . . . 19

4 HOW-TOs **23**
- 4.1 Customize Zinnia's look and feel . . . 23
- 4.2 Extending Entry model . . . 28
- 4.3 Rewriting Entry's URL . . . 31

5 Development **35**
- 5.1 Contributing to Zinnia . . . 35
- 5.2 Buildout . . . 37
- 5.3 Testing and Coverage . . . 39

6 References **41**
- 6.1 List of settings . . . 41
- 6.2 Template Tags . . . 47
- 6.3 Zinnia API . . . 54

7 Notes **109**
- 7.1 Frequently Asked Questions . . . 109
- 7.2 Compatibility . . . 111
- 7.3 Thanks . . . 113

7.4 CHANGELOG 114

8 Related **123**

9 Indices and tables **125**

Python Module Index **127**

Welcome to the version 0.19 of the documentation.

You can also find the differents editions of the documentation online at readthedocs.org.

CHAPTER 1

Django Blog Zinnia

Simple yet powerful and really extendable application for managing a blog within your Django Web site.

Zinnia has been made for publishing Weblog entries and designed to do it well.

Basically any feature that can be provided by another reusable app has been left out. Why should we re-implement something that is already done and reviewed by others and tested?

Features

More than a long speech, here the list of the main features:

- Comments
- Sitemaps
- Archives views
- Related entries
- Private entries
- RSS or Atom Feeds
- Tags and categories views
- Advanced search engine
- Prepublication and expiration
- Custom templates for various contents
- Editing in Markdown, Textile or reStructuredText
- Widgets (Popular entries, Similar entries, ...)
- Spam protection with Akismet, TypePad or Mollom
- Admin dashboard

- MetaWeblog API
- Ping Directories
- Ping External links
- Bit.ly support
- Twitter support
- Gravatar support
- Django-CMS plugins
- Collaborative work
- Tags autocompletion
- Entry model extendable
- Pingback/Trackback support
- Blogger conversion utility
- WordPress conversion utility
- WYMeditor, TinyMCE , CKEditor and MarkItUp support
- Efficient database queries
- Ready to use and extendable templates
- Compass and Sass3 integration
- Windows Live Writer compatibility

Examples

Take a look at the online demo at: http://demo.django-blog-zinnia.com/ or you can visit these websites who use Zinnia.

- Fantomas' side
- Ubuntu's developers blog
- Vidzor Studio LLC
- Bookshadow
- Future Proof Games
- Programeria
- Tihomir Blajev Blog

If you are a proud user of Zinnia, send me the URL of your website and I will add it to the list.

Online resources

More information and help available at these URLs:

- Code repository
- Documentation
- Travis CI server

- Coverage report
- Discussions and help at Google Group
- For reporting a bug use GitHub Issues

CHAPTER 2

Getting Started

Installation

Dependencies

Make sure to install these packages prior to installation :

- Python >= 2.7
- Django >= 1.10,<1.11
- Pillow >= 2.0.0
- django-mptt >= 0.8.6
- django-tagging >= 0.4.5
- beautifulsoup4 >= 4.1.3
- mots-vides >= 2015.2.6
- regex >= 2016.3.2
- django-contrib-comments >= 1.7

The packages below are optionnal but needed for run the full test suite or migrate the database.

- pytz
- pyparsing >= 2.0.1
- django-xmlrpc >= 0.1.7

Note that all the needed dependencies will be resolved if you install Zinnia with **`pip`** or **`easy_install`**, excepting Django.

Getting the code

For the latest stable version of Zinnia use **easy_install**:

```
$ easy_install django-blog-zinnia
```

or use **pip**:

```
$ pip install django-blog-zinnia
```

You could also retrieve the last sources from https://github.com/Fantomas42/django-blog-zinnia. Clone the repository using **git** and run the installation script:

```
$ git clone git://github.com/Fantomas42/django-blog-zinnia.git
$ cd django-blog-zinnia
$ python setup.py install
```

or more easily via **pip**:

```
$ pip install -e git://github.com/Fantomas42/django-blog-zinnia.git#egg=django-blog-
↪zinnia
```

Applications

Assuming that you have an already existing Django project, register *zinnia*, and these following applications in the INSTALLED_APPS section of your project's settings.

```
INSTALLED_APPS = (
  'django.contrib.auth',
  'django.contrib.admin',
  'django.contrib.sites',
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'django.contrib.contenttypes',
  'django_comments',
  'mptt',
  'tagging',
  'zinnia',
)
```

Template Context Processors

Add these following `template context processors` if not already present.

```
TEMPLATES = [
  {
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'APP_DIRS': True,
    'OPTIONS': {
      'context_processors': [
        'django.contrib.auth.context_processors.auth',
        'django.template.context_processors.i18n',
        'django.template.context_processors.request',
        'django.contrib.messages.context_processors.messages',
```

```
        'zinnia.context_processors.version',  # Optional
      ]
    }
  }
]
```

URLs

Add at least these following lines to your project's urls.py in order to display the Weblog.

```
url(r'^weblog/', include('zinnia.urls')),
url(r'^comments/', include('django_comments.urls')),
```

Remember to enable the `admin` site in the urls.py of your project if you haven't done it yet for having the edition capabilities.

Note that the default Zinnia URLset *`zinnia.urls`* is calibrated for convenient usage, but you can customize your Weblog URLs as you want. Here's a custom implementation of the URLs provided by Zinnia:

```
blog_urls = ([
    url(r'^', include('zinnia.urls.capabilities')),
    url(r'^search/', include('zinnia.urls.search')),
    url(r'^sitemap/', include('zinnia.urls.sitemap')),
    url(r'^trackback/', include('zinnia.urls.trackback')),
    url(r'^blog/tags/', include('zinnia.urls.tags')),
    url(r'^blog/feeds/', include('zinnia.urls.feeds')),
    url(r'^blog/random/', include('zinnia.urls.random')),
    url(r'^blog/authors/', include('zinnia.urls.authors')),
    url(r'^blog/categories/', include('zinnia.urls.categories')),
    url(r'^blog/comments/', include('zinnia.urls.comments')),
    url(r'^blog/', include('zinnia.urls.entries')),
    url(r'^blog/', include('zinnia.urls.archives')),
    url(r'^blog/', include('zinnia.urls.shortlink')),
    url(r'^blog/', include('zinnia.urls.quick_entry'))
], 'zinnia')

url(r'^', include(blog_urls))
```

Sites

Define the value of `SITE_ID` if not already done.

```
SITE_ID = 1
```

Emails

Be sure that the sending of emails is correctly configured, otherwise the moderation system will not work. Please refer to https://docs.djangoproject.com/en/dev/topics/email/ for more information about sending emails.

Static Files

Since the version 1.3 of Django, Zinnia uses the `staticfiles` application to serve the static files needed. Please refer to https://docs.djangoproject.com/en/dev/howto/static-files/ for more information about serving static files.

Syncing the database

Now that you have everything set up, simply run the following in your project directory to sync the models with the database.

```
$ python manage.py migrate
```

Advanced Configuration

Sitemaps

One of the cool features of Django is the sitemap application, so if you want to fill your Web site's sitemap with the entries of your blog, follow these steps.

- Register `django.contrib.sitemaps` in the `INSTALLED_APPS` section.
- Edit your project's URLs and add this code:

```
from django.contrib.sitemaps.views import index
from django.contrib.sitemaps.views import sitemap

from zinnia.sitemaps import AuthorSitemap
from zinnia.sitemaps import CategorySitemap
from zinnia.sitemaps import EntrySitemap
from zinnia.sitemaps import TagSitemap

sitemaps = {
    'tags': TagSitemap,
    'blog': EntrySitemap,
    'authors': AuthorSitemap,
    'categories': CategorySitemap
}

urlpatterns += [
    url(r'^sitemap.xml$',
        index,
        {'sitemaps': sitemaps}),
    url(r'^sitemap-(?P<section>.+)\.xml$',
        sitemap,
        {'sitemaps': sitemaps},
        name='django.contrib.sitemaps.views.sitemap'),
]
```

Templates for entries

In your Weblog you will always publish entries, but sometimes you want to have a different look and feel for special entries.

You may want to publish an entry with a short content like a quote, in which case it would be better not to provide a *continue reading* link when rendering this entry.

To solve this problem, Zinnia allows the user to select a template to render the entry's content and the entry's detail page.

In order to use a template without the *continue reading* link, we need to register it under this setting in the project's configuration:

```
ZINNIA_ENTRY_CONTENT_TEMPLATES = [
  ('zinnia/_short_entry_detail.html', 'Short entry template'),
]
```

Now we will create the `zinnia/_short_entry_detail.html` template with this sample of code:

```
{% extends "zinnia/_entry_detail.html" %}

{% block continue-reading %}{% endblock %}
```

A new template is now available in the admin interface to display the entry without the *continue reading* link when displayed in a list.

Then if you want to have custom rendering of the detail page of the entry, by displaying the entry fullwidth without the sidebar for example, the same process applies. We will add this setting in the project's configuration:

```
ZINNIA_ENTRY_DETAIL_TEMPLATES = [
    ('zinnia/fullwidth_entry_detail.html', 'Fullwidth template'),
]
```

And now we finally create the `zinnia/fullwidth_entry_detail.html` template with this sample of code:

```
{% extends "zinnia/entry_detail.html" %}

{% block sidebar-class %}no-sidebar{% endblock %}

{% block sidebar %}{% endblock %}
```

Pinging

By default Zinnia is configured to ping the directories and the external urls embedded in your entries when a new entry is published.

If you want to completly remove these features, simply set these settings in your project's configuration:

```
ZINNIA_PING_EXTERNAL_URLS = False
ZINNIA_SAVE_PING_DIRECTORIES = False
```

You can also edit the list of the directories to be pinged by using this setting:

```
ZINNIA_PING_DIRECTORIES = ('http://ping.directory.com/',
                           'http://pong.directory.com/')
```

Markup languages

If you doesn't want to write your entries in HTML, because you are an über coder knowing more than 42 programming languages, you have the possibility to use a custom markup language for editing the entries.

Currently **MarkDown**, **Textile** and **reStructuredText** are supported, so if you want to use one of these languages, first set this setting as appropriate in your project's settings.

```
ZINNIA_MARKUP_LANGUAGE = 'restructuredtext'
```

Note that the name of the language must be in lowercase.

Then install the corresponding library to your needs:

- `textile` – requires Textile >= 2.1.5
- `markdown` – requires Markdown >= 2.3.1
- `restructuredtext` – requires Docutils >= 0.10

Cache

For performance considerations the Django's cache API is used when comparing the entries between them. To isolate these operations, the `CACHES` setting must contain a value named `'comparison'`, otherwise the `'default'` value will be used.

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
    },
    'comparison': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'comparison',
        'TIMEOUT': None,
   }
}
```

XML-RPC

Zinnia provides few Webservices via XML-RPC, but before using it, you need to install django-xmlrpc.

Then register `django_xmlrpc` in your `INSTALLED_APPS` section of your project's settings.

Finally we need to register the URL of the XML-RPC server. Insert something like this in your project's urls.py:

```
url(r'^xmlrpc/$', 'django_xmlrpc.views.handle_xmlrpc'),
```

Note: For the Pingback service check if your site is enabled for pingback detection. More information at http://hixie.ch/specs/pingback/pingback-1.0#TOC2

Upgrading Zinnia

If you want to upgrade your installation of Zinnia from a previous release, it's easy, but you need to be cautious. The whole process takes less than 15 minutes.

Dumping

The first thing to do is a to dump your data for safety reasons.

```
$ python manage.py dumpdata --indent=2 zinnia > dump_zinnia_before_migration.json
```

Update Zinnia's code

We are now ready to upgrade Zinnia. If you want to use the latest stable version use **easy_install** with this command:

```
$ easy_install -U django-blog-zinnia
```

or if you prefer to upgrade from the development release, use **pip** like that:

```
$ pip install -U -e git://github.com/Fantomas42/django-blog-zinnia.git#egg=django-
↪blog-zinnia
```

Update the database

The database should probably be updated to the latest database schema of Zinnia.

```
$ python manage.py migrate zinnia
```

The database is now up to date, and ready to use.

Check list

In order to finish the upgrade process, we must check if everything works fine by browsing the Web site.

By experience, problems mainly come from customized templates, because of changes in the URL reverse functions.

CHAPTER 3

Topics

Channels

Views by author, categories, tags is not enough :).

The idea is to create specific pages based on a query search. Imagine that we want to customize the homepage of the Weblog, because we write on a variety of subjects and we don't want to bore visitors who aren't interested in some really specific entries. Another usage of the channels is for SEO, for aggregating entries under a well-formatted URL.

For doing that Zinnia provides a view called *EntryChannel*.

If we take our first example, we will do like that for customizing the Weblog homepage in our project's urls.py.

```
from zinnia.views.channels import EntryChannel

url(r'^weblog/$', EntryChannel.as_view(
    query='category:python OR category:django')),
url(r'^weblog/', include('zinnia.urls', namespace='zinnia')),
```

The first URL will handle the homepage of the blog instead of the default URL provided by Zinnia.

As we can see, the only required argument for this view is `query`. This parameter represents a query search string. This string will be interpreted by the search engine activated in Zinnia and return a list of entries (See *Search Engines* for more informations).

So our homepage will only display entries filled under the categories **Python** or **Django**.

The others parameters handled by the channel view are the same that the generic view named `ListView`.

Search Engines

Zinnia like almost all blogging systems contains a `search engine` feature.

But in fact there are 2 search engines, a basic and an advanced, the advanced search engine is enabled by default, but if he fails the basic search engine will resume the job.

Basic Search Engine

The basic search engine is the original engine of Zinnia, and will be used if the advanced engine cannot be used.

It will always returns more results than the advanced engine, because each terms of the query will be searched in the entries and the results are added to a main result list. We can say that the results are inclusives.

Example of a query : `love paris`

This will returns all the entries containing the terms `love` or `paris`.

Advanced Search Engine

The advanced search engine has several possibilities for making more elaborated queries, with it's own grammar system.

The grammar of the search is close to the main search engines like Google or Yahoo.

The main difference with the basic engine is that the results are exclusives.

For enabling the advanced search engine, you simply need to install the `pyparsing` package. Otherelse the basic engine will be used.

Query examples

Here a list of examples and possibilities:

Example of a query with terms: `love paris`

This will returns all the entries containing the terms `love` and `paris`.

Example of a query with excluded terms: `paris -hate`

This will returns all the entries containing the term `paris` without the term `hate`.

Example of a query with expressions: `"Paris, I love you"`

This will returns all the entries containing the expression `Paris, I love you`.

Example of a query with category operator: `love category:paris`

This will returns all the entries containing the term `love` filled in the category named `paris`.

Example of a query with tag operator: `paris tag:love`

This will returns all the entries containing the term `paris` with the tag `love`.

Example of a query with author operator: `paris author:john`

This will returns all the entries containing the term `paris` writed by `john`.

Example of a query with boolean operator: `paris or berlin`

This will returns all the entries containing the term `paris` or `berlin`.

Example of e query with parenthesis: `(paris or berlin) love`

This will returns all the entries containing the terms `paris` or `berlin` with the term `love`.

Complex example: `((paris or berlin) and (tag:love or category:meet*)) girl -money`

This will returns all the entries containing the terms `paris` or `berlin` with the tag `love` or filled under the categories starting by `meet` also containing the term `girl` excluding entries with the term `money`.

Note that the query is stripped of common words known as stop words. These are words such as **on**, **the** or **which** that are generally not meaningful and cause irrelevant results.

URL Shortener

New in version 0.9.

The URL shortening has becoming a big deal of the Internet especially for transfering long URLs.

And so many URL shortening services exist, each with his own features.

Originally Zinnia provided a only way to generate short URLs for your entries, and you needed to install mod:*django-bitly*.

One way it's not bad, but it's not enough.

Now Zinnia provides his own backend by default for making the URLs of the entries shorter, example:

> http://mydomain.com/blog/2S/

This backend use the primary key of the entries, encoded in base 36 to save a few more characters.

Of course the URL is short (and can be shorter) but if you have a long domain, the URL can be not so short, example:

> http://mysuperverylongdomain.com/blog/15R/ (42 characters !)

But now you can easily change this behavior and use your favorite URL shortener service by writing a backend shortening your URLs.

Note: The default backend is limited. When reaching the primary key **46656**, the short URLs generated enter in conflict with the archives by year.

If you have reached this number of entries, it's effectively a good idea to change the default backend for a more scalable solution.

Writing your own URL shortener backend

Writing a backend for using your custom URL shortener is simple as possible, you only needs to follows 4 rules.

1. In a new Python file write a function named **backend** taking an *`Entry`* instance in parameters.
2. The **backend** function should returns an URL including the protocol and the domain.
3. If the **backend** requires initial configuration you must raise a `ImproperlyConfigured` exception if the configuration is not valid. The error will be displayed in the console.
4. Register your backend to be used in your project with this setting:

```
ZINNIA_URL_SHORTENER_BACKEND = 'path.to.your.url.shortener.module'
```

Here the source code of the default backend.

```
from django.contrib.sites.models import Site
from django.core.urlresolvers import reverse
from zinnia.settings import PROTOCOL

def backend(entry):
```

```
    return '%s://%s%s' % (PROTOCOL, Site.objects.get_current().domain,
                          reverse('zinnia_entry_shortlink', args=[entry.pk]))
```

For a more examples take a look in this folder: `zinnia/url_shortener/backends/`.

Spam Checker

New in version 0.9.

Spam protection is mandatory when you want to let your users react on your entries.

Originally Zinnia provided a only one type of spam protection with the support of Akismet.

One it's not bad, but it's not enough, because depend of a third-party service may be a little bit risky.

Now Akismet has been moved in a dedicated module and the moderation system let you choose the spam checkers to use. With this new feature you can now write a custom spam checker corresponding to your needs and use it for moderating your comments, trackbacks and linkbacks.

We can imagine for example that you want to authorize comments from a white-list of IPs, it's possible by writing a backend.

Note: You can use multiple backends for checking the content, because they are chained, useful for a maximum protection.

Configuration example:

```
ZINNIA_SPAM_CHECKER_BACKENDS = (
    'path.to.your.spam.checker.module',
    'path.to.your.other.spam.checker.module',
)
```

See also:

ZINNIA_SPAM_CHECKER_BACKENDS

Changed in version 0.19.

The spam protection now also apply on incoming trackbacks and linkbacks.

Built-in spam checkers

- *zinnia.spam_checker.backends.all_is_spam*
- *zinnia.spam_checker.backends.long_enough*

Writing your own spam checker backend

Writing a backend for using a custom spam checker is simple as possible, you only needs to follows 4 rules.

1. In a new Python file write a function named **backend** taking in parameter : `content` the text to verify, `content_object` the object related to the text and `request` the current request.
2. The **backend** function should returns `True` if `content` is spam and `False` otherwhise.

3. If the **backend** requires initial configuration you must raise an `ImproperlyConfigured` exception if the configuration is not valid. The error will be displayed in the console.
4. Register your backend to be used in your project with this setting:

```
ZINNIA_SPAM_CHECKER_BACKENDS = ('path.to.your.spam.checker.module',)
```

For a more examples take a look in this folder : `zinnia/spam_checker/backends/`.

Permissions

In addition to the **add**, **change** and **delete** permissions automatically created, the default *`Entry`* model provides three extra permissions. These permissions will be used in the admin site to provide a collaborative work feature when creating and editing the entries. You can use these permissions in your custom views and templates and of course change the list of Entry's permissions by *Extending Entry model*.

See also:

`django.db.models.Options.permissions` for more information about the permissions on the Django models.

Now let's move on to the descriptions and implementations of these permissions.

Can view all entries

In the admin site, this permission is used to limit the entries displayed and editable by a staff member. If the user does not have this permission, only his own entries will be editable. It's particulary useful when you have multiple authors and you don't want them to be allowed to share the entries

Can change status

Thanks to this permission, a user can change the status of an entry. If the user is not granted with this permission, he will be able to create entries but they will remain in the `DRAFT` status until someone granted with this permission changes the status to `PUBLISH`.

Or you can let an user edit your entries without letting him change the publication status.

Can change authors

This permission allows a user to change the authors who can participate to the entries. When you create an entry, you will be its author by default, unless you set the authors field. If you are granted with this permission, you can assign any staff member to the authors' list. If you set an author who does not have the `can_view_all` permission, he will now be able to view the entry.

Ecosystem

As explained in the main goals part of Zinnia, any feature that can be provided by another reusable app has been left out. This principle must be applied from downstream to upstream.

This principle has already been applied to the downstream part, by strictly selecting the dependancies required by Zinnia, in order not to reinvent the wheel and to respect the DRY principles.

Now it's time to talk about the upstream part. Zinnia is made to be *Ready To Use* by providing all the core functionalities required by a Weblog application. But sometimes even a full Weblog is not enough, so Zinnia has also been made fully extendable, so it encourages and facilitates the creation of extensions.

Since Zinnia has become stable, documented and reviewed, some extensions have been built to enhance the core functionalities of the Weblog. These extensions act like an ecosystem: they add multiple layers of functionnalities around the core - which is Django and Zinnia - and they allow interaction between each layer independently.

Of course, your Zinnia's Weblog can run without theses extensions, but you might find some that suit your needs.

Note: If you have written or are aware of an extension that can enhance the Zinnia's ecosystem, please share your code or information by sending me a message. Your extension will then be listed here.

---

cmsplugin-zinnia

Cmsplugin-zinnia is a bridge between Django-CMS and Zinnia.

This package provides plugins, menus and apphook for integrating your Zinnia powered Weblog into your django-cms Website.

The code bundled in this application is a copy of the original `zinnia.plugins` module, made for forward compatibility with django-blog-zinnia > 0.11.

URL: https://github.com/django-blog-zinnia/cmsplugin-zinnia

admin-tools-zinnia

Admin-tools-zinnia is an extension based on django-admin-tools providing new dashboard modules for your admin interface, related to your Weblog.

Useful for having a sexier admin interface.

URL: https://github.com/django-blog-zinnia/admin-tools-zinnia

zinnia-threaded-comment

Zinnia-threaded-comments customizes the comment application bundled by Django to enable replies to comments within your Weblog.

URL: https://github.com/django-blog-zinnia/zinnia-threaded-comments

zinnia-theme-html5

Zinnia-theme-html5 is an extension theme for making your Zinnia's Weblog HTML5 ready.

URL: https://github.com/django-blog-zinnia/zinnia-theme-html5

Changed in version 0.15.2: This extension is no longer needed, because Zinnia is now HTML5 ready.

zinnia-theme-bootstrap

Zinnia-theme-bootstrap is an extension theme for your Weblog based on Bootstrap.

URL: https://github.com/django-blog-zinnia/zinnia-theme-bootstrap

zinnia-theme-foundation

Zinnia-theme-foundation is an extension theme for your Weblog based on Zurb Foundation 5.

URL: https://github.com/django-blog-zinnia/zinnia-theme-foundation

zinnia-wysiwyg-wymeditor

Zinnia-wysiwyg-wymeditor is an extension for editing your entries in the admin with WYMEditor.

URL: https://github.com/django-blog-zinnia/zinnia-wysiwyg-wymeditor

zinnia-wysiwyg-tinymce

Zinnia-wysiwyg-tinymce is an extension for editing your entries in the admin with TinyMCE.

URL: https://github.com/django-blog-zinnia/zinnia-wysiwyg-tinymce

zinnia-wysiwyg-ckeditor

Zinnia-wysiwyg-ckeditor is an extension for editing your entries in the admin with CKeditor.

URL: https://github.com/django-blog-zinnia/zinnia-wysiwyg-ckeditor

zinnia-wysiwyg-markitup

Zinnia-wysiwyg-markitup is an extension for editing your entries in the admin with MarkItUp.

URL: https://github.com/django-blog-zinnia/zinnia-wysiwyg-markitup

zinnia-url-shortener-hashids

Zinnia-url-shortener-bitly is an extension providing URL shortening for the entries via Hashids algorithm.

URL: https://github.com/django-blog-zinnia/zinnia-url-shortener-hashids

zinnia-url-shortener-bitly

Zinnia-url-shortener-bitly is an extension providing URL shortening for the entries via Bit.ly.

URL: https://github.com/django-blog-zinnia/zinnia-url-shortener-bitly

zinnia-spam-checker-akismet

Zinnia-spam-checker-akismet is an extension adding anti-spam protection via Akismet or Typepad.

URL: https://github.com/django-blog-zinnia/zinnia-spam-checker-akismet

zinnia-spam-checker-mollom

Zinnia-spam-checker-mollom is an extension adding anti-spam protection via Mollom.

URL: https://github.com/django-blog-zinnia/zinnia-spam-checker-mollom

zinnia-twitter

Zinnia-twitter is an admin extension allowing you to post your entries on Twitter.

URL: https://github.com/django-blog-zinnia/zinnia-twitter

wordpress2zinnia

Migrate your Wordpress blog into Zinnia.

URL: https://github.com/django-blog-zinnia/wordpress2zinnia

blogger2zinnia

Migrate your Blogger blog into Zinnia.

URL: https://github.com/django-blog-zinnia/blogger2zinnia

feed2zinnia

Import RSS or Atom feeds into Zinnia.

URL: https://github.com/django-blog-zinnia/feed2zinnia

byteflow2zinnia

Migrate your users, tags, command and posts from Byteflow to Zinnia by Richard Espelin.

URL: https://bitbucket.org/resplin/byteflow2zinnia

zinnia-drupal

Helper Django application for importing content from Drupal into Django Blog Zinnia by Branko Majic.

URL: https://github.com/azaghal/zinnia-drupal

CHAPTER 4

HOW-TOs

Customize Zinnia's look and feel

The templates provided for Zinnia are simple but complete and as generic as possible. You can easily change them by specifying a template directory. If you are not familiar with Django, part two of the excellent Django tutorial explains in details how to customize the look and feel of the `admin` app: it's actually the same thing in Zinnia.

A good starting point is to copy-paste the `zinnia/base.html` template, and edit the `extends` instruction in order to fit into your skin.

Note:

- The main content is displayed in a block named `content`.
- Additional data is displayed in a block named `sidebar`.

You can also create your own app containing some Zinnia's templates based on inheritance. For example you can find these applications which can be a good starting point to make your own at:

- Zinnia-theme-bootstrap.
- Zinnia-theme-foundation.
- Zinnia-theme-html5.
- Django Blog Quintet.

Warning: Changed in version 0.9.

Django Blog Quintet is no longer compatible with Zinnia, but is still a good example.

Now that we have seen the basic mechanisms to add and customize Zinnia's templates we will see in details the different possibilities in the customization process.

CSS customizations

Most of the time the customization process of Zinnia is about editing the cascading style sheet of the differents pages delivered by the Weblog.

First of all you have to note that each page of the Weblog has several classes applied on the `<body>` markup. For examples if the document has paginated entries, the `paginated` and `page-{id}` classes will be added. Many classes are used within the default templates so should take a look on it, maybe it will be useful for you.

Secondly all the documents served by Zinnia have the `zinnia` class name on the `<body>`. If you remove this class, all the default CSS provided by Zinnia will not be applied. And if you add it on templates provided by third-party applications, the Zinnia's style will be applied. Pretty useful, for enabling or disabling Zinnia's default style.

Of course adding or removing classes can easily be done in your own templates by overriding the block named `body-class`.

You also have to note that a real effort has be done for providing clean and valid HTML documents, without redundant and useless classes or IDs overweighting the document respecting the **presentation-free markup** rule.

Now that you have all of these information in mind, you can add new cascading style sheets into your templates, containing your customization rules and of course remove the default CSS files provided by Zinnia if needed.

Variations on the default theme

New in version 0.12.

Beside the `zinnia` class name in the `<body>` tag of the `zinnia/skeleton.html` template, three other class names are available:

```
<body class="zinnia default blue right-sidebar {% block body-class %}{% endblock %}">
```

The `default` class name represents the original default theme of Zinnia. You can remove this class, or replace with the classes `light` or `dark` to activate the variations with high readability and contrast, thanks to the Solarized project.

The `blue` class represents the main color used within the theme. Available color are: `yellow`, `orange`, `red`, `magenta`, `violet`, `blue`, `cyan`, `green`.

The `right-sidebar` class sets the sidebar at right and `left-sidebar` at left, by default if none of these classes are present, the sidebar is set at right. You can hide the sidebar by using the `no-sidebar` class.

With these 3 sets of classes available in the CSS, you now have 4*9*3=108 variations of the default theme available. Try them and choose your favorite!

Special templates

Since the beginning of Zinnia, the development has been influenced by the idea of **Power templates for easy rendering**. Customizing all the templates of the Weblog must be possible, easy and fast. So Zinnia has a unique feature for returning custom templates depending on the view's context.

Templates for filters

Zinnia as a complete Weblog application provides views for filtering the last entries by authors, categories and tags. In these views you have the possibility to use a dedicated template related to the filtering model. This feature is useful for highlighting a special category or for providing a template per author.

Each of these views will return a list of templates name to render the page but only the first template name matching to an existing template will be used to render.

Examples:

- For the URL `/blog/categories/events/` the *`CategoryDetail`* view will be called and return this list of template names:

```
['zinnia/category/event/entry_list.html',
 'zinnia/category/event_entry_list.html',
 'zinnia/category/entry_list.html',
 'zinnia/entry_list.html']
```

- For the URL `/blog/tags/featured/` the *`TagDetail`* view will be called and return this list of template names:

```
['zinnia/tag/featured/entry_list.html',
 'zinnia/tag/featured_entry_list.html',
 'zinnia/tag/entry_list.html',
 'zinnia/entry_list.html']
```

- For the URL `/blog/authors/keneda/` the *`AuthorDetail`* view will be called and return this list of template names:

```
['zinnia/author/keneda/entry_list.html',
 'zinnia/author/keneda_entry_list.html',
 'zinnia/author/entry_list.html',
 'zinnia/entry_list.html']
```

Templates for archives

Concerning the archive views the same feature is implemented, a list of template names will be returned depending of the date and the archive period. This feature take all his sense if want to use *Halloween* or *Christmas* templates for your Weblog. With this feature you can also program and re-use your themes on several periods.

Another side effect is if you write an Entry during the *Halloween* period with dedicated templates, even after the *Halloween* period the templates will still be used.

Examples:

- For the URL `/blog/2012/` the *`EntryYear`* view will be called and return this list of template names:

```
['zinnia/archives/2012/entry_archive_year.html',
 'zinnia/archives/entry_archive_year.html',
 'zinnia/entry_archive_year.html',
 'entry_archive_year.html']
```

- For the URL `/blog/2012/week/16/` the *`EntryWeek`* view will be called and return this list of template names:

```
['zinnia/archives/2012/week/16/entry_archive_week.html',
 'zinnia/archives/week/16/entry_archive_week.html',
 'zinnia/archives/2012/entry_archive_week.html',
 'zinnia/archives/entry_archive_week.html',
 'zinnia/entry_archive_week.html',
 'entry_archive_week.html']
```

- For the URL `/blog/2012/04/21/` the `EntryDay` view will be called and return this list of template names:

```
['zinnia/archives/2012/04/21/entry_archive_day.html',
 'zinnia/archives/month/04/day/21/entry_archive_day.html',
 'zinnia/archives/2012/day/21/entry_archive_day.html',
 'zinnia/archives/day/21/entry_archive_day.html',
 'zinnia/archives/2012/month/04/entry_archive_day.html',
 'zinnia/archives/month/04/entry_archive_day.html',
 'zinnia/archives/2012/entry_archive_day.html',
 'zinnia/archives/entry_archive_day.html',
 'zinnia/entry_archive_day.html',
 'entry_archive_day.html']
```

Templates for entry detail

Each entries of the Weblog has the possibility to have his own template to be rendered by using the *ZINNIA_ENTRY_DETAIL_TEMPLATES* settings, so with this option you can handle multiple presentation for your entries. And because *EntryDetail* is based on an archive view a custom list of templates is built uppon the publication date. The entry's slug is also used to build the template list for having maximal customization capabilities with ease.

For example if I use the `custom.html` template to render the entry located at the URL `/blog/2012/04/21/my-entry/` the list of template names will be:

```
['zinnia/archives/2012/04/21/my-entry_custom.html',
 'zinnia/archives/month/04/day/21/my-entry_custom.html',
 'zinnia/archives/2012/day/21/my-entry_custom.html',
 'zinnia/archives/day/21/my-entry_custom.html',
 'zinnia/archives/2012/04/21/my-entry.html',
 'zinnia/archives/month/04/day/21/my-entry.html',
 'zinnia/archives/2012/day/21/my-entry.html',
 'zinnia/archives/day/21/my-entry.html',
 'zinnia/archives/2012/04/21/custom.html',
 'zinnia/archives/month/04/day/21/custom.html',
 'zinnia/archives/2012/day/21/custom.html',
 'zinnia/archives/day/21/custom.html',
 'zinnia/archives/2012/month/04/my-entry_custom.html',
 'zinnia/archives/month/04/my-entry_custom.html',
 'zinnia/archives/2012/month/04/my-entry.html',
 'zinnia/archives/month/04/my-entry.html',
 'zinnia/archives/2012/month/04/custom.html',
 'zinnia/archives/month/04/custom.html',
 'zinnia/archives/2012/my-entry_custom.html',
 'zinnia/archives/2012/my-entry.html',
 'zinnia/archives/2012/custom.html',
 'zinnia/archives/my-entry_custom.html',
 'zinnia/my-entry_custom.html',
 'my-entry_custom.html',
 'zinnia/archives/my-entry.html',
 'zinnia/my-entry.html',
 'my-entry.html',
 'zinnia/archives/custom.html',
 'zinnia/custom.html',
 'custom.html']
```

Now you have the choice !

Templates for entries' content

Imagine that you have different kind of entries, some with photos, some with videos or even with tweets. You might not want to share the same presentation between these different entries.

An elegent solution to better highlight the content is to use different templates for each kind of content or presentation you want.

You can easily do this by using the *ZINNIA_ENTRY_CONTENT_TEMPLATES*. It allows you to specify a content template for each entries of your blog using the administration interface.

Templates within loops

When displaying a list of entries the templates are chosen according to *ZINNIA_ENTRY_CONTENT_TEMPLATES*.

But how can we specify a template for a given index position the list ? For example if we want to highlight the first entry.

Then we simply create a new template suffixed by a dash followed by the position.

Example: `zinnia/_entry_detail-1.html`

If we use an underscore instead of a dash the position will reset at every new page. Replacing the dash by an underscore in the previous example would highlight the first entry of every page.

You can bypass this behavior altogether and have more control over your templates by using the *ZINNIA_ENTRY_LOOP_TEMPLATES* setting.

Changing templates

Maybe CSS customizations and adding markup to the templates is not enough because you need to change a more important part of the templates or you simply don't want to use it.

Because all the front views bundled in Zinnia are customizable, changing the template used to render the view is pretty easy and can be a good solution for you if you are confortable with Django.

Example of changing the default template for the search view by another view:

```
from zinnia.views.search import EntrySearch

class CustomTemplateEntrySearch(EntrySearch):
    template_name = 'custom/template.html'
```

or directly in the urls:

```
from django.conf.urls import url
from django.conf.urls import patterns

from zinnia.views.search import EntrySearch

urlpatterns = patterns(
    '',
    url(r'^$', EntrySearch.as_view(
        template_name='custom/template.html'),
        name='entry_search'),
    )
```

Going further

As you can see that you can customize the look and feel of Zinnia by CSS, SASS, HTML and Python and even by adding custom views. So why don't you make a Python package containing a Django application of your complete theme ? The theme of your weblog will be sharable and easily installable.

Remember to take a look at Zinnia-theme-bootstrap for having a good starting point of a packaged theme.

Extending Entry model

New in version 0.8.

The *Entry* model bundled in Zinnia can now be extended and customized.

This feature is useful for who wants to add some fields in the model, or change its behavior. It also allows Zinnia to be a really generic and reusable application.

Why extending ?

Imagine that I find Zinnia really great for my project but some fields or features are missing to be the Weblog app that suits to my project. For example I need to add a custom field linking to an image gallery, two solutions:

- I search for another Django blogging app fitting my needs.
- I do a monkey patch, into the Zinnia code base.

These two solutions are really bad.

For the first solution maybe you will not find the desired application and also mean that Zinnia is not a reusable application following the Django's convention. For the second solution, I don't think that I need to provide more explanations about the evil side of monkey patching (evolution, reproduction...). That's why Zinnia provides a third generic solution.

- Customizing the *Entry* model noninvasively with the power of class inheritance !

The extension process is done in three main steps:

1. Write a model class containing your customizations.
2. Register your model class into Zinnia to be used.
3. Create and register a new `ModelAdmin` class for your new model class.

Writing model extension

In the suite of this document we will show how to add an image gallery into the *Entry* model to illustrate the concepts involved when extending. We assume that the pieces of codes written for this document belong in the `zinnia_gallery` module/application.

Changed in version 0.13.

The `zinnia.models.entry.EntryAbstractClass` has been moved and renamed to *zinnia.models_bases.entry.AbstractEntry*.

The first step to extend the *Entry* model is to define a new class inherited from the *AbstractEntry* and add your fields or/and override the inherited methods if needed. So in `zinnia_gallery` let's write our gallery models and the extension in the *Entry* model in `models.py`.

```
from django.db import models
from zinnia.models_bases.entry import AbstractEntry

class Picture(models.Model):
    title = models.CharField(max_length=50)
    image = models.ImageField(upload_to='gallery')

class Gallery(models.Model):
    title = models.CharField(max_length=50)
    pictures = models.ManyToManyField(Picture)

class EntryGallery(AbstractEntry):
    gallery = models.ForeignKey(Gallery)

    def __str__(self):
        return 'EntryGallery %s' % self.title

    class Meta(AbstractEntry.Meta):
        abstract = True
```

In this code sample, we simply add in our *Entry* model a new ForeignKey field named gallery pointing to a Gallery model and we override the Entry.__unicode__() method.

Note: You have to respect **2 important rules** to make extending working :

1. Do not import the *Entry* model in your file defining the extended model because it will cause a circular importation.
2. Don't forget to tell that your model is abstract. Otherwise a table will be created and the extending process will not work as expected.

See also:

Model inheritance for more information about the concepts behind the model inheritence in Django and the limitations.

Writing model customisation

Adding fields is pretty easy, but now that the *Entry* model has been extended, we want to change the image field wich is an ImageField by default to use our new Picture instead.

To customise this field, the same process as extending apply, but we can take advantage of all the abstracts classes provided to build the *AbstractEntry* to rebuild our own custom *Entry* model like this:

```
from django.db import models
from zinnia.models_bases import entry

class Picture(models.Model):
    title = models.CharField(max_length=50)
    image = models.ImageField(upload_to='gallery')

class Gallery(models.Model):
    title = models.CharField(max_length=50)
    pictures = models.ManyToManyField(Picture)

class EntryGallery(
          entry.CoreEntry,
```

```
          entry.ContentEntry,
          entry.DiscussionsEntry,
          entry.RelatedEntry,
          entry.ExcerptEntry,
          entry.FeaturedEntry,
          entry.AuthorsEntry,
          entry.CategoriesEntry,
          entry.TagsEntry,
          entry.LoginRequiredEntry,
          entry.PasswordRequiredEntry,
          entry.ContentTemplateEntry,
          entry.DetailTemplateEntry):

    image = models.ForeignKey(Picture)
    gallery = models.ForeignKey(Gallery)

    def __str__(self):
        return 'EntryGallery %s' % self.title

    class Meta(entry.CoreEntry.Meta):
        abstract = True
```

Now we have an *Entry* model extended with a gallery of pictures and customised with a `Picture` model relation as the `image` field.

Note that the same process apply if you want to delete some built-in fields.

Considerations about the database

If you do the extension of the *Entry* model, you have to alter the Zinnia's database tables for reflecting your changes made on the model class.

Fortunately since Django 1.7 you just have to write a new migration for reflecting your changes, but the migration script will be written in the `zinnia.migrations` module, which is not recommended because the result is not replicable for multiple installations and breaks the migration system with future releases of Zinnia.

Fortunatly Django provides a solution with the `MIGRATION_MODULES` setting. Once this setting is done for the `'zinnia'` key, can now start to write new migrations. Note that it is recommended to use a different package then the default one for your app to avoid conflict (e.g. `MIGRATION_MODULES = {'zinnia': 'zinnia_gallery.migrations_zinnia'}`).

It's recommended that the new **initial** migration represents the default *Entry* schema provided by Zinnia, because after that, you just have to write a new migration for reflecting your changes, and you just alter your database schema with the `migrate` command.

Registering the extension

Once your extension class is defined you simply have to register it, with the *ZINNIA_ENTRY_BASE_MODEL* setting in your Django settings. The expected value is a string representing the full Python path to the extented model's class name. This is the easiest part of the process.

Following our example we must add this line in the project's settings.

```
ZINNIA_ENTRY_BASE_MODEL = 'zinnia_gallery.models.EntryGallery'
```

If an error occurs when your new class is imported a warning will be raised and the `EntryAbstractClass` will be used.

Updating the admin interface

Now we should create a new *`EntryAdmin`* admin class to reflect our changes and use the new fields.

To do that we will write a new admin class inherited from *`EntryAdmin`* and register it within the admin site.

In the file `zinnia_gallery/admin.py` we can write these code lines for adding the gallery field:

```
from django.contrib import admin
from django.utils.translation import ugettext_lazy as _

from zinnia.models.entry import Entry
from zinnia.admin.entry import EntryAdmin

class EntryGalleryAdmin(EntryAdmin):
  # In our case we put the gallery field
  # into the 'Content' fieldset
  fieldsets = (
    (_('Content'), {
      'fields': (('title', 'status'), 'lead', 'content',)}),
    (_('Illustration'), {
      'fields': ('image', 'gallery'),
      'classes': ('collapse', 'collapse-closed')}),) + \
    EntryAdmin.fieldsets[2:]

admin.site.register(Entry, EntryGalleryAdmin)
```

Templating

Now we can easily *customize the templates* provided by Zinnia to display the gallery field into the Weblog's pages.

For more information you can see another implementation example in the cmsplugin-zinnia package.

Rewriting Entry's URL

By default the *`Entry`* model implements a default `get_absolute_url()` method to retrieve the canonical URL for an instance into the Weblog.

See also:

`get_absolute_url()` for more information about the usage of this method if your are not familiar with this concept.

The result of this method is a string composed of the entry's `creation date` and the `slug`. For example this URL: `/blog/2011/07/17/how-to-change-url/` refers to an entry created on the 17th July 2011 under the slug `how-to-change-url`.

This URL pattern is common for most of the Weblog engines and have these following advantages.

- SEO Friendly.
- Human readable.
- You can remove parts of the URL and find archives.

- The slug is unique with the creation date, so you can reuse it.

But if you want to change it into a different form, you have to know that it's possible, but not easy.

You have to note that the changes required on the Zinnia's code base to simplify this customization step in a generic way, are evil, dirty and unsecured. You will see throughout this document why this customization is not directly implemented, why it cannot be handled genericaly and which are the pitfalls to avoid.

Warning: Before further reading, you have to note that the methods explained below are reserved for confirmed Django developers, knowing what they are doing. No warranties and no support will be provided for the problems encountered if you customize this part of Zinnia.

Choosing your new URL pattern

We can imagine many different forms of new URL for your entries:

- `/blog/<id>/`
- `/blog/<slug>/`
- `/blog/<year>/<slug>/`
- `/blog/<creation-date>-<slug>/`
- `/blog/<slug>/<tag-1>/<tag-n>/`
- `/blog/<category-1>/<category-n>/<slug>/`

As you can see we can imagine a lot of new patterns to handle the canonical URL of an entry. But you must keep in mind that you must have a unique URL per entry.

Like we said above, the slug is unique with the creation date, so only using the entry' slug to retrieve the matching *`Entry`* instance is not safe, because the view will fail if you have 2 entries with the same slug.

If you want to decorate the entry's slug with the categories' slugs of the entry, or with some additionnal datas (like in the latest examples), make sure that you can write an efficient regular expression for capturing text in the URL. The complexity of the URL's regexp will depend on the pattern choosen for the new URL.

For the rest of this document we will show how to change the entry's URL with the `/blog/<id>/` pattern. This is just to illustrate the facts presented in this document, because this pattern is already handled by the default *URL Shortener* backend, but have the advantage to be perfect for this tutorial.

We assume that the code involved in this document belong in the `zinnia_customized` package/application. This package will contain all the pieces of code to customize the default behaviour of Zinnia.

The `Entry.get_absolute_url()` method

Accordingly to your new URL pattern you have to override the `Entry.get_absolute_url()` method to pass the desired parameters to build the canonical URL of an entry.

To do this override, simply use the method explained in the *Extending Entry model* document to create a new class based on *`AbstractEntry`* with the new `get_absolute_url` method.

```
class EntryWithNewUrl(AbstractEntry):
    """Entry with '/blog/<id>/' URL"""

    @models.permalink
    def get_absolute_url(self):
```

```
        return ('zinnia:entry_detail', (),
                {'pk': self.id})

    class Meta(AbstractEntry.Meta):
        abstract = True
```

Due to the intensive use of this method into the templates, make sure that your re-implemention is not too slow. For example hitting the database to recontruct this URL is not a really good idea. That's why an URL pattern based on the categories like `/blog/<category-1>/<category-n>/<slug>/` is really bad.

Adding your view

Now we must write a custom view to handle the detailed view of an *Entry* instance from the text parameters passed in the URL. So in a module called `zinnia_customized.views` we can write this view for handling our new URL.

```
from django.views.generic.detail import DetailView

from zinnia.models.entry import Entry
from zinnia.views.mixins.entry_preview import EntryPreviewMixin
from zinnia.views.mixins.entry_protection import EntryProtectionMixin

class EntryDetail(EntryPreviewMixin,
                  EntryProtectionMixin,
                  DetailView):
    queryset = Entry.published.on_site()
    template_name_field = 'template'
```

Pretty easy isn't it ? For more information, check the documentation about the `DetailView` view. Note that the `EntryProtectionMixin` is used for enabling password and login protections if needed on the entry.

Configuring URLs

The final step to rewrite the entry's URL, is to change the URLconf for the Weblog application. Instead of using the default implementation provided by *zinnia.urls* in your project's URLconf, you have to re-implement all the URLsets provided by Zinnia as described in the *URLs* section of the installation process.

But instead of including *zinnia.urls.entries* you will include your own URLconf containing the new URL code for the canonical URL of your entries. Doing a copy of the original module in your own project can save you a lot time.

```
...
url(r'^weblog/', include('zinnia_customized.urls', namespace='zinnia')),
...
```

Now in `zinnia_customized.urls` rewrite the `url()` named `'zinnia_entry_detail'` with your new regular expression handling the canonical URL of your entries and the text parameters. Don't forget to also change the path to your view retrieving the *Entry* instance from the text parameters.

```
from zinnia_customized.views import EntryDetail

url(r'^(?P<pk>\d+)/$',
    EntryDetail.as_view(),
    name='entry_detail')
```

Warning: If you use the pingback XML-RPC service, you will also need change to *pingback_ping()* function for retrieving the *Entry* instance, accordingly to the new text parameters captured in the URL.

Actually you should consider Zinnia like a ready to use Weblog application and also like a framework to make customized Weblog engines.

CHAPTER 5

Development

Contributing to Zinnia

Zinnia is an open-source project, so yours contributions are welcomed and needed.

Writing code

So you have a great idea to program, found a bug or a way to optimize the code ? You are welcome.

Process

1. Fork and clone the repository on Github.
2. Create a branch based on `develop`.
3. Write tests.
4. Develop your code.
5. Update the documentation if needed.
6. Push your branch and open a pull-request.

Once the pull-request is open, the continuous integration server will build your pull-request. If the build is passing, your contribution has great chances to be integrated quickly.

Conventions

Code conventions are important in a way where they ensure the lisibility of the code in the time, that's why the code try to respect at most the **PEP 8**.

If you have already *run the buildout* script you can execute this Makefile rule to check your code.

```
$ make kwalitee
```

With a clear and uniform code, the development is better and faster.

Tests

The submited code should be covered with one or more unittests to ensure the new behavior and will make easier future developments. Without that, your code will not be reliable and may not be integrated.

See *Testing and Coverage* for more informations.

Writing documentation

Sometimes considered like "annoying" by hard-core coders, documentation is more important than the code itself! 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 a semi-decent English. People need to understand you. We don't care about style or correctness.

The documentation should :

- Be written in English.
- Follow the 80 column rule.
- Use **.rst** as file extension.
- Use **Sphinx** and **restructuredText**.
- Be accessible. You should assume the reader to be moderately familiar with Python and Django, but not anything else.

Keep it mind that documenting is most useful than coding, so your contribution will be greatly appreciated.

Contributing changes

Contribute changes to the documentation in the same fashion as committing to source code. Essentially, you will fork the project on github, make your changes to the documentation, commit them, and submit a pull request.

See *code process* for more details.

Writing CSS

You want to contribute to the default stylesheets provided by Zinnia ?

If you take a look at `zinnia/static/zinnia/theme/css/screen.css` you will probably notice that the CSS is not edited manually. It has been generated from Sass files and so it is good pratice not to edit this file directly.

Aside of `zinnia/static/zinnia/theme/css` directory, you can see another directory named `sass` which is organized like this:

```
sass/
|-- config/
|-- mixins/
|-- partials/
`-- screen.scss
```

The `partials` folder contains all the **partials** used to build the CSS, the `mixins` folder contains **reusable mixins** like the tag-cloud and finally the `config` folder contains all the **configurable variables**. For example the `screen.scss` file will include at the end all the files who belong in these directories into a single compiled CSS document, named `screen.css`.

Actually the Sass files are compiled with the libsass implementation using a Gulp script.

To install and use Gulp, you need to have a recent version of Node.js and install the dependencies like this in the root directory of Zinnia:

```
$ npm install .
```

Then you just have to run the `gulp` command and start to edit the Sass files to customize the stylesheets provided by Zinnia.

Once you are done, open a new pull-request on Github with your commited changes.

Translations

If you want to contribute by updating a translation or adding a translation in your language, it's simple: create a account on Transifex.net and you will be able to edit the translations at this URL :

https://www.transifex.net/projects/p/django-blog-zinnia/resource/djangopo/

The translations hosted on Transifex.net will be pulled periodically in the repository, but if you are in a hurry, send me a message.

If you've found that a particular piece of text cannot be translated in your language, because it lacks a plural form, or requires to be split in two separate sentences to deal with a different gender, you can click the open issue button to mark your comment as an issue. A developer can then resolve the issue.

Buildout

To increase the speed of the development process a buildout script is provided to properly initialize the project for anybody who wants to contribute to the project.

Buildout is a developer oriented tool designed for workings with Python eggs, so can be used for installing egg-based scripts for personal use.

One of the major force of buildout is that is **repeatable**, it should be possible to check-in a buildout specification and reproduce the same software later by checking out the specification and rebuilding.

Actually buildout is actively used for development and deployment.

VirtualEnv

First of all, please use virtualenv to protect your system, it's not mandatory but handy.

What problem does virtualenv solve? If you like Python as I do, chances are you want to use it for other projects besides django-blog-zinnia. But the more projects you have, the more likely it is that you will be working with different versions of Python itself, or at least different versions of Python libraries. Let's face it; quite often libraries break backwards compatibility, and it's unlikely that any serious application will have zero dependencies.

So what do you do if two or more of your projects have conflicting dependencies? Virtualenv basically enables multiple side-by-side installations of Python, one for each project. It doesn't actually install separate copies of Python, but it does provide a clever way to keep different project environments isolated.

So if you doesn't already have virtualenv I suggest to you to type one of the following two commands:

```
$ sudo easy_install virtualenv
```

or even better:

```
$ sudo pip install virtualenv
```

Running the buildout

Before running the buildout script we will clone the main development repository of django-blog-zinnia, create a virtual Python environment to isolate the installation of the required librairies, then bootstrap the buildout script to finally execute it.

Follow these few command to start the development:

```
$ git clone git://github.com/Fantomas42/django-blog-zinnia.git
$ cd django-blog-zinnia
$ virtualenv .
$ source ./bin/activate
$ pip install zc.buildout
$ ./bin/buildout
```

The buildout script will resolve all the dependencies needed to develop the application and install some usefull scripts.

Once the buildout has run, you are ready to hack the Zinnia project.

Development scripts

Use this command to launch the test suite:

```
$ ./bin/test
```

To view the code coverage run this command:

```
$ ./bin/cover
```

Execute these commands to check the code conventions:

```
$ ./bin/flake8 --count -r --exclude=tests.py,migrations zinnia
```

For building the HTML documentation run this simple command:

```
$ ./bin/docs
```

Demo project

A demo project using Zinnia, is available once the buildout script has run. The demo project is usefull when you want to do functionnal testing.

To launch the demo site, execute these commands:

```
$ ./bin/demo migrate
$ ./bin/demo runserver
```

To directly have entries in your demo, run this command:

```
$ ./bin/demo loaddata helloworld
```

Pretty easy no ?

Testing and Coverage

> ***"An application without tests, is a dead-born application."*** Someone very serious

Writing tests is important, maybe more important than coding.

And this for a lot of reasons, but I'm not here to convince you about the benefits of software testing, some prophets will do it better than me.

- http://en.wikipedia.org/wiki/Software_testing
- https://docs.djangoproject.com/en/dev/topics/testing/

Of course Zinnia is tested using the unittest approach. All the tests belong in the directory `zinnia/tests/`.

Launching the test suite

If you have *run the buildout script* bundled in Zinnia, the tests are run under nose by launching this command:

```
$ ./bin/test
```

But the tests can also be launched within a Django project with the default test runner:

```
$ django-admin.py test zinnia --settings=zinnia.testsettings
```

Using the `./bin/test` script is usefull when you develop because the tests are calibrated to run fast, but testing Zinnia within a Django project even if it's slow, can prevent some integration issues.

Coverage

Despite my best efforts, some functionnalities are not yet tested, that's why I need your help !

As I write these lines the **282** tests in Zinnia cover **100%** of the code bundled in Zinnia. A real effort has been made to obtain this percentage, for ensuring the quality of the code.

I know that a coverage percent does not represent the quality of the tests, but maintaining or increasing this percentage ensures the quality of Zinnia and his future evolutions. For information, you can check the actual coverage percent on Python 2.7 online.

I hope that you will write some tests and find some bugs. :)

CHAPTER 6

References

List of settings

Zinnia has a lot of parameters to configure the application accordingly to your needs. Knowing this list of settings can save you a lot of time.

Here's a full list of all available settings, and their default values.

All settings described here can be found in `zinnia/settings.py`.

- *Entry*
- *Edition*
- *Preview*
- *Views*
- *Feeds*
- *URLs*
- *Comments*
- *Linkbacks*
- *Pinging*
- *Miscellaneous*

Entry

ZINNIA_ENTRY_BASE_MODEL

Default value: `'zinnia.models_bases.entry.AbstractEntry'` (Empty string)

String defining the base model path for the Entry model. See *Extending Entry model* for more informations.

ZINNIA_ENTRY_DETAIL_TEMPLATES

Default value: `[]` (Empty list)

List of tuple for extending the list of templates availables for rendering the entry detail view. By using this setting, you can change the look and feel of an entry page directly in the admin interface. Example:

```
ZINNIA_ENTRY_DETAIL_TEMPLATES = [('entry_detail_alternate.html',
                                  gettext('Alternative template')),]
```

ZINNIA_ENTRY_CONTENT_TEMPLATES

Default value: `[]` (Empty list)

List of tuple for extending the list of templates availables for rendering the content of an entry. By using this setting, you can change the look and feel of an entry directly in the admin interface. Example:

```
ZINNIA_ENTRY_CONTENT_TEMPLATES = [('zinnia/_entry_detail_alternate.html',
                                   gettext('Alternative template')),]
```

ZINNIA_ENTRY_LOOP_TEMPLATES

Default value: `{'default': {}}`

Dictionary of dictionaries of indexes for by-passing the template used when rendering an entry within a loop of filtered entries. By using this setting, you can change with Python code, the look and feel of an entry within a specific loop. Example:

```
ZINNIA_ENTRY_LOOP_TEMPLATES = {
    'default': {1: 'zinnia/_entry_detail_first.html'},
    'author-admin': dict([(i, 'zinnia/_entry_detail_admin.html')
                          for i in range(1000) if not i % 5])
}
```

ZINNIA_UPLOAD_TO

Default value: `'uploads/zinnia'`

String setting that tells Zinnia where to upload entries' images.

Changed in version 0.10.

Previously the default value was `'uploads'`.

Edition

ZINNIA_MARKUP_LANGUAGE

Default value: `'html'`

String determining the markup language used for writing the entries. You can use one of these values:

```
['html', 'markdown', 'restructuredtext', 'textile']
```

ZINNIA_MARKDOWN_EXTENSIONS

Default value: `[]` (Empty list)

List of either markdown.Extension instances or extension paths, used for rendering the entries in MarkDown. Example:

```
ZINNIA_MARKDOWN_EXTENSIONS =  ['markdown.extensions.nl2br',
                               MyExtension(mysetting="foo")]
```

ZINNIA_RESTRUCTUREDTEXT_SETTINGS

Default value: `{}` (Empty dict)

A dictionary containing settings for the RestructuredText markup processing. See the Docutils restructuredtext writer settings docs for details.

Preview

ZINNIA_PREVIEW_SPLITTERS

Default value: `['<!-- more -->', '<!--more-->'])`

List of split markers used to make a preview of the entry's content if present in the HTML. All the content before the marker will be used to build the preview of the entry.

ZINNIA_PREVIEW_MAX_WORDS

Default value: `55`

Number of words used to build the entry's preview if no split markers are found.

ZINNIA_PREVIEW_MORE_STRING

Default value: `' ...'`

The string to be appended to the content when a truncation for the preview is done.

Views

ZINNIA_PAGINATION

Default value: `10`

Integer used to paginate the entries. So by default you will have 10 entries displayed per page on the Weblog.

ZINNIA_ALLOW_EMPTY

Default value: `True`

Used for archives views, raise a 404 error if no entries are present at a specified date.

ZINNIA_ALLOW_FUTURE

Default value: `True`

Used for allowing archives views in the future.

Feeds

ZINNIA_FEEDS_FORMAT

Default value: `'rss'`

String determining the format of the syndication feeds. You can use `'atom'` if your prefer Atom feeds.

ZINNIA_FEEDS_MAX_ITEMS

Default value: `15`

Integer used to define the maximum items provided in the syndication feeds. So by default you will have 15 entries displayed on the feeds.

URLs

ZINNIA_TRANSLATED_URLS

New in version 0.12.2.

Default value: `False`

Boolean used to activate the internationalization of the URLs provided by Zinnia if the translation is avaialable in your language.

ZINNIA_URL_SHORTENER_BACKEND

Default value: `'zinnia.url_shortener.backends.default'`

String representing the module path to the URL shortener backend.

ZINNIA_PROTOCOL

Default value: `'http'`

String representing the protocol of the site. If your Web site uses HTTPS, set this setting to `https`.

Comments

ZINNIA_AUTO_MODERATE_COMMENTS

Default value: `False`

Determine if a new comment should be marked non-public and await approval. Leave as `False` to allow comments to show up immediately.

ZINNIA_AUTO_CLOSE_COMMENTS_AFTER

Default value: `None` (forever)

Determine the number of days where comments are open. If you set this setting to `10` the comments will be closed automaticaly 10 days after the publication date of your entries.

`0` means disabling comments completely.

ZINNIA_MAIL_COMMENT_REPLY

Default value: `False`

Boolean used for sending an email to comment's authors when a new comment is posted.

ZINNIA_MAIL_COMMENT_AUTHORS

Default value: `True`

Boolean used for sending an email to entry authors when a new comment is posted.

ZINNIA_MAIL_COMMENT_NOTIFICATION_RECIPIENTS

Default value:

```
[manager_tuple[1] for manager_tuple in settings.MANAGERS]
```

List of emails used for sending a notification when a new public comment has been posted.

ZINNIA_SPAM_CHECKER_BACKENDS

Default value: `[]` (Empty list)

List of strings representing the module path to a spam checker backend. See *Spam Checker* for more informations about this setting.

ZINNIA_COMMENT_MIN_WORDS

Default value: `4`

Minimal number of words required to post a comment if *`zinnia.spam_checker.backends.long_enough.backend()`* is enabled in *`ZINNIA_SPAM_CHECKER_BACKENDS`*.

ZINNIA_COMMENT_FLAG_USER_ID

Default value: `1`

The ID of the User to be used when flagging the comments as spam, pingback or trackback.

Linkbacks

ZINNIA_AUTO_CLOSE_PINGBACKS_AFTER

Default value: `None` (forever)

Determine the number of days where pingbacks are open. If you set this setting to `10` the pingbacks will be closed automaticaly 10 days after the publication date of your entries.

`0` means disabling pingbacks completely.

ZINNIA_AUTO_CLOSE_TRACKBACKS_AFTER

Default value: `None` (forever)

Determine the number of days where trackbacks are open. If you set this setting to `10` the trackbacks will be closed automaticaly 10 days after the publication date of your entries.

`0` means disabling trackbacks completely.

Pinging

ZINNIA_PING_DIRECTORIES

Default value: `('http://django-blog-zinnia.com/xmlrpc/',)`

List of the directories you want to ping.

ZINNIA_PING_EXTERNAL_URLS

Default value: `True`

Boolean setting for telling if you want to ping external URLs when saving an entry.

ZINNIA_SAVE_PING_DIRECTORIES

Default value: `bool(ZINNIA_PING_DIRECTORIES)`

Boolean setting for telling if you want to ping directories when saving an entry.

ZINNIA_PINGBACK_CONTENT_LENGTH

Default value: `300`

Size of the excerpt generated on pingback.

Miscellaneous

ZINNIA_COPYRIGHT

Default value: `'Zinnia'`

String used for copyrighting your entries, used in the syndication feeds and in the opensearch document.

ZINNIA_COMPARISON_FIELDS

Default value: `['title', 'lead', 'content', 'excerpt', 'image_caption', 'tags']`

List of text fields used to find similarity between entries.

ZINNIA_SEARCH_FIELDS

Default value: `['title', 'lead', 'content', 'excerpt', 'image_caption', 'tags']`

List of text fields used to search within entries.

Template Tags

Zinnia provides several template tags based on inclusion_tag system to create some **widgets** in your Web site's templates.

Note: The presence of the `template` argument in many template tags allow you to reuse and customize the rendering of a template tag in a generic way. Like that you can display the same template tag many times in your pages but with a different appearance.

To start using any of the following template tags you need to load them first at the top of your template:

```
{% load zinnia %}
```

get_recent_entries

Display the latest entries.

`zinnia.templatetags.zinnia.`**`get_recent_entries`**(*number=5*, *template='zinnia/tags/entries_recent.html'*)

Return the most recent entries.

Usage examples:

```
{% get_recent_entries %}
{% get_recent_entries 3 %}
{% get_recent_entries 3 "custom_template.html" %}
{% get_recent_entries template="custom_template.html" %}
```

get_featured_entries

Display the featured entries.

zinnia.templatetags.zinnia.**get_featured_entries**(*number=5*, *template='zinnia/tags/entries_featured.html'*)

Return the featured entries.

Usage examples:

```
{% get_featured_entries %}
{% get_featured_entries 3 %}
{% get_featured_entries 3 "custom_template.html" %}
{% get_featured_entries template="custom_template.html" %}
```

get_draft_entries

Display the latest entries marked as draft.

zinnia.templatetags.zinnia.**get_draft_entries**(*number=5*, *template='zinnia/tags/entries_draft.html'*)

Return the last draft entries.

Usage examples:

```
{% get_draft_entries %}
{% get_draft_entries 3 %}
{% get_draft_entries 3 "custom_template.html" %}
{% get_draft_entries template="custom_template.html" %}
```

get_random_entries

Display random entries.

zinnia.templatetags.zinnia.**get_random_entries**(*number=5*, *template='zinnia/tags/entries_random.html'*)

Return random entries.

Usage examples:

```
{% get_random_entries %}
{% get_random_entries 3 %}
{% get_random_entries 3 "custom_template.html" %}
{% get_random_entries template="custom_template.html" %}
```

get_popular_entries

Display popular entries.

zinnia.templatetags.zinnia.**get_popular_entries**(*number=5*, *template='zinnia/tags/entries_popular.html'*)

Return popular entries.

Usage examples:

```
{% get_popular_entries %}
{% get_popular_entries 3 %}
{% get_popular_entries 3 "custom_template.html" %}
{% get_popular_entries template="custom_template.html" %}
```

get_similar_entries

Display entries similar to an existing entry.

zinnia.templatetags.zinnia.**get_similar_entries**(*context*, *number=5*, *template='zinnia/tags/entries_similar.html'*)

Return similar entries.

Usage examples:

```
{% get_similar_entries %}
{% get_similar_entries 3 %}
{% get_similar_entries 3 "custom_template.html" %}
{% get_similar_entries template="custom_template.html" %}
```

get_calendar_entries

Display an HTML calendar with date of publications.

If you don't set the *year* or the *month* parameter, the calendar will look in the context of the template if one of these variables is set in this order : `(month, day, object.creation_date)`.

If no one of these variables is found, the current month will be displayed.

zinnia.templatetags.zinnia.**get_calendar_entries**(*context*, *year=None*, *month=None*, *template='zinnia/tags/entries_calendar.html'*)

Return an HTML calendar of entries.

Usage examples:

```
{% get_calendar_entries %}
{% get_calendar_entries 2011 4 %}
{% get_calendar_entries 2011 4 "custom_template.html" %}
{% get_calendar_entries template="custom_template.html" %}
{% get_calendar_entries year=object.creation_date|date:"Y" month=12 %}
```

get_archives_entries

Display the archives by month.

zinnia.templatetags.zinnia.**get_archives_entries**(*template='zinnia/tags/entries_archives.html'*)

Return archives entries.

Usage examples:

```
{% get_archives_entries %}
{% get_archives_entries "custom_template.html" %}
```

get_archives_entries_tree

Display all the archives as a tree.

zinnia.templatetags.zinnia.**get_archives_entries_tree**(*template='zinnia/tags/entries_archives_tree.html'*)
Return archives entries as a tree.

Usage examples:

```
{% get_archives_entries_tree %}
{% get_archives_entries_tree "custom_template.html" %}
```

get_authors

Display all the published authors.

zinnia.templatetags.zinnia.**get_authors**(*context*, *template='zinnia/tags/authors.html'*)
Return the published authors.

Usage examples:

```
{% get_authors %}
{% get_authors "custom_template.html" %}
```

get_categories

Display all the published categories.

zinnia.templatetags.zinnia.**get_categories**(*context*, *template='zinnia/tags/categories.html'*)
Return the published categories.

Usage examples:

```
{% get_categories %}
{% get_categories "custom_template.html" %}
```

get_categories_tree

Display a hierarchical tree of all the categories available.

zinnia.templatetags.zinnia.**get_categories_tree**(*context*, *template='zinnia/tags/categories_tree.html'*)
Return the categories as a tree.

Usage examples:

```
{% get_categories_tree %}
{% get_categories "custom_template.html" %}
```

get_tags

Store in a context variable a queryset of all the published tags.

zinnia.templatetags.zinnia.**get_tags**()
Return the published tags.

Usage example:

```
{% get_tags as entry_tags %}
```

get_tag_cloud

Display a cloud of published tags.

zinnia.templatetags.zinnia.**get_tag_cloud**(*context*, *steps=6*, *min_count=None*, *template='zinnia/tags/tag_cloud.html'*)

Return a cloud of published tags.

Usage examples:

```
{% get_tag_cloud %}
{% get_tag_cloud 9 %}
{% get_tag_cloud 9 3 %}
{% get_tag_cloud 9 3 "custom_template.html" %}
{% get_tag_cloud template="custom_template.html" %}
```

get_recent_comments

Display the latest comments.

zinnia.templatetags.zinnia.**get_recent_comments**(*number=5*, *template='zinnia/tags/comments_recent.html'*)

Return the most recent comments.

Usage examples:

```
{% get_recent_comments %}
{% get_recent_comments 3 %}
{% get_recent_comments 3 "custom_template.html" %}
{% get_recent_comments template="custom_template.html" %}
```

get_recent_linkbacks

Display the latest linkbacks.

zinnia.templatetags.zinnia.**get_recent_linkbacks**(*number=5*, *template='zinnia/tags/linkbacks_recent.html'*)

Return the most recent linkbacks.

Usage examples:

```
{% get_recent_linkbacks %}
{% get_recent_linkbacks 3 %}
{% get_recent_linkbacks 3 "custom_template.html" %}
{% get_recent_linkbacks template="custom_template.html" %}
```

zinnia_pagination

Display a Digg-like pagination for long list of pages.

zinnia.templatetags.zinnia.**zinnia_pagination**(*context*, *page*, *begin_pages=1*, *end_pages=1*, *before_pages=2*, *after_pages=2*, *template='zinnia/tags/pagination.html'*)

Return a Digg-like pagination, by splitting long list of page into 3 blocks of pages.

Usage examples:

```
{% zinnia_pagination page_obj %}
{% zinnia_pagination page_obj 2 2 %}
{% zinnia_pagination page_obj 2 2 3 3 %}
{% zinnia_pagination page_obj 2 2 3 3 "custom_template.html" %}
{% zinnia_pagination page_obj begin_pages=2 template="custom_template.html" %}
```

zinnia_breadcrumbs

Display the breadcrumbs for the pages handled by Zinnia.

zinnia.templatetags.zinnia.**zinnia_breadcrumbs**(*context*, *root_name=''*, *template='zinnia/tags/breadcrumbs.html'*)

Return a breadcrumb for the application.

Usage examples:

```
{% zinnia_breadcrumbs %}
{% zinnia_breadcrumbs "News" %}
{% zinnia_breadcrumbs "News" "custom_template.html" %}
{% zinnia_breadcrumbs template="custom_template.html" %}
```

zinnia_loop_template

Store in a context variable a `Template` choosen from his position whithin a loop of entries.

zinnia.templatetags.zinnia.**zinnia_loop_template**(*context*, *default_template*)

Return a selected template from his position within a loop and the filtering context.

Usage example:

```
{% for object in object_list %}
  {% zinnia_loop_template "my-template.html" as template %}
  {% include template %}
{% endfor %}
```

zinnia_statistics

Display the statistics about the contents handled in Zinnia.

zinnia.templatetags.zinnia.**zinnia_statistics**(*template='zinnia/tags/statistics.html'*)

Return statistics on the content of Zinnia.

Usage examples:

```
{% zinnia_statistics %}
{% zinnia_statistics "custom_template.html" %}
```

get_gravatar

Display the Gravatar image associated to an email, useful for comments.

zinnia.templatetags.zinnia.**get_gravatar**(*email*, *size=80*, *rating='g'*, *default=None*, *protocol='http'*)

Return url for a Gravatar.

Usage examples:

```
{% get_gravatar user.email %}
{% get_gravatar user.email 50 %}
{% get_gravatar user.email 50 "PG" %}
{% get_gravatar user.email 50 "PG" "identicon" "https" %}
{% get_gravatar user.email rating="PG" protocol="https" %}
```

widont

Insert a non-breaking space between the last two words of your sentence.

zinnia.templatetags.zinnia.**widont**(**args*, ***kwargs*)

Add an HTML non-breaking space between the final two words of the string to avoid "widowed" words.

Usage example:

```
{{ variable|widont }}
```

week_number

Return the Python week number of a date.

zinnia.templatetags.zinnia.**week_number**(*date*)

Return the Python week number of a date. The django |date:"W" returns incompatible value with the view implementation.

Usage example:

```
{{ date_variable|week_number }}
```

comment_admin_urlname

Return an admin URL for managing the comments, whatever the the application used.

zinnia.templatetags.zinnia.**comment_admin_urlname**(*action*)

Return the admin URLs for the comment app used.

Usage example:

```
{% url 'changelist'|comment_admin_urlname %}
```

user_admin_urlname

Return an admin URL for managing the users, whatever the the application used.

zinnia.templatetags.zinnia.**user_admin_urlname**(*action*)
Return the admin URLs for the user app used.

Usage example:

```
{% url 'changelist'|user_admin_urlname %}
```

Zinnia API

Contents

- *Zinnia API*
  - zinnia *Package*
  - *apps Module*
  - *breadcrumbs Module*
  - *calendar Module*
  - *comparison Module*
  - *context_processors Module*
  - *context Module*
  - *feeds Module*
  - *flags Module*
  - *managers Module*
  - *markups Module*
  - *moderator Module*
  - *ping Module*
  - *preview Module*
  - *search Module*
  - *signals Module*
  - *sitemaps Module*
  - *templates Module*
  - *Subpackages*

zinnia Package

Zinnia

apps Module

Apps for Zinnia

class zinnia.apps.**ZinniaConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

Config for Zinnia application.

label = 'zinnia'

name = 'zinnia'

ready()

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

breadcrumbs Module

Breadcrumb module for Zinnia

class zinnia.breadcrumbs.**Crumb**(*name*, *url=None*)
Bases: object

Part of the breadcrumbs.

zinnia.breadcrumbs.**day_crumb**(*date*)
Crumb for a day.

zinnia.breadcrumbs.**entry_breadcrumbs**(*entry*)
Breadcrumbs for an Entry.

zinnia.breadcrumbs.**handle_page_crumb**(*func*)
Decorator for handling the current page in the breadcrumbs.

zinnia.breadcrumbs.**month_crumb**(*date*)
Crumb for a month.

zinnia.breadcrumbs.**retrieve_breadcrumbs**(*path*, *model*, *page*, *root_name*)
Build a semi-hardcoded breadcrumbs based of the model's url handled by Zinnia.

zinnia.breadcrumbs.**year_crumb**(*date*)
Crumb for a year.

calendar Module

Calendar module for Zinnia

class zinnia.calendar.**Calendar**
Bases: calendar.HTMLCalendar

Extension of the HTMLCalendar.

formatday(*day*, *weekday*)
Return a day as a table cell with a link if entries are published this day.

formatfooter(*previous_month*, *next_month*)
Return a footer for a previous and next month.

formatmonth(*theyear*, *themonth*, *withyear=True*, *previous_month=None*, *next_month=None*)
Return a formatted month as a table with new attributes computed for formatting a day, and thead/tfooter.

formatmonthname(*theyear*, *themonth*, *withyear=True*)
Return a month name translated as a table row.

formatweekday(*day*)
Return a weekday name translated as a table header.

formatweekheader()
Return a header for a week as a table row.

comparison Module

Comparison tools for Zinnia

class zinnia.comparison.**CachedModelVectorBuilder**(***kwargs*)
Bases: *zinnia.comparison.ModelVectorBuilder*

Cached version of VectorBuilder.

cache
Get the cache from cache.

cache_backend
Try to access to comparison cache value, if fail use the default cache backend config.

cache_flush()
Flush the cache for this instance.

cache_key
Key for the cache.

columns_dataset
Implement high level cache system for columns and dataset.

get_cache()
Get the cache from cache.

get_related(*instance*, *number*)
Implement high level cache system for get_related.

set_cache(*value*)
Assign the cache in cache.

class zinnia.comparison.**EntryPublishedVectorBuilder**(***kwargs*)
Bases: *zinnia.comparison.CachedModelVectorBuilder*

Vector builder for published entries.

cache_key
Key for the cache handling current site.

fields = ['title', 'lead', 'content', 'excerpt', 'image_caption', 'tags']

limit = 100

queryset = <zinnia.managers.EntryPublishedManager object>

class zinnia.comparison.**ModelVectorBuilder**(***kwargs*)
Bases: object

Build a list of vectors based on a Queryset.

columns
Access to columns.

columns_dataset
Generate the columns and the whole dataset.

compute_related(*object_id*, *score=<function pearson_score>*)
Compute the most related pks to an object's pk.

dataset
Access to dataset.

fields = None

get_related(*instance*, *number*)
Return a list of the most related objects to instance.

limit = None

queryset = None

raw_clean(*datas*)
Apply a cleaning on raw datas.

raw_dataset
Generate a raw dataset based on the queryset and the specified fields.

zinnia.comparison.**pearson_score**(*list1*, *list2*)
Compute the Pearson' score between 2 lists of vectors.

zinnia.context_processors Module

Context Processors for Zinnia

zinnia.context_processors.**version**(*request*)
Add version of Zinnia to the context.

zinnia.context Module

Context module for Zinnia

zinnia.context.**get_context_first_matching_object**(*context*, *context_lookups*)
Return the first object found in the context, from a list of keys, with the matching key.

zinnia.context.**get_context_first_object**(*context*, *context_lookups*)
Return the first object found in the context, from a list of keys.

zinnia.context.**get_context_loop_positions**(*context*)
Return the paginated current position within a loop, and the non-paginated position.

zinnia.feeds Module

Feeds for Zinnia

class zinnia.feeds.**AuthorEntries**
Bases: *zinnia.feeds.EntryFeed*

Feed filtered by an author.

description(*obj*)
Description of the feed.

get_object(*request*, *username*)
Retrieve the author by his username.

get_title(*obj*)
Title of the feed.

items(*obj*)
Items are the published entries of the author.

link(*obj*)
URL of the author.

class zinnia.feeds.**CategoryEntries**
Bases: *zinnia.feeds.EntryFeed*

Feed filtered by a category.

description(*obj*)
Description of the feed.

get_object(*request*, *path*)
Retrieve the category by his path.

get_title(*obj*)
Title of the feed.

items(*obj*)
Items are the published entries of the category.

link(*obj*)
URL of the category.

class zinnia.feeds.**DiscussionFeed**
Bases: *zinnia.feeds.ZinniaFeed*

Base class for discussion Feed.

description_template = 'feeds/discussion_description.html'

item_author_email(*item*)
Author's email of the discussion item.

item_author_link(*item*)
Author's URL of the discussion.

item_author_name(*item*)
Author of the discussion item.

item_link(*item*)
URL of the discussion item.

item_pubdate(*item*)
Publication date of a discussion.

title_template = 'feeds/discussion_title.html'

class zinnia.feeds.**EntryComments**
Bases: *zinnia.feeds.EntryDiscussions*

Feed for comments on an entry.

description(*obj*)
Description of the feed.

description_template = 'feeds/comment_description.html'

get_title(*obj*)
Title of the feed.

item_enclosure_length(*item*)
Hardcoded enclosure length.

item_enclosure_mime_type(*item*)
Hardcoded enclosure mimetype.

item_enclosure_url(*item*)
Return a gravatar image for enclosure.

item_link(*item*)
URL of the comment.

items(*obj*)
Items are the comments on the entry.

title_template = 'feeds/comment_title.html'

class zinnia.feeds.**EntryDiscussions**
Bases: *zinnia.feeds.DiscussionFeed*

Feed for discussions on an entry.

description(*obj*)
Description of the feed.

get_object(*request*, *year*, *month*, *day*, *slug*)
Retrieve the discussions by entry's slug.

get_title(*obj*)
Title of the feed.

items(*obj*)
Items are the discussions on the entry.

link(*obj*)
URL of the entry.

class zinnia.feeds.**EntryFeed**
Bases: *zinnia.feeds.ZinniaFeed*

Base Entry Feed.

description_template = 'feeds/entry_description.html'

item_author_email(*item*)
Return the first author's email. Should not be called if self.item_author_name has returned None.

item_author_link(*item*)
Return the author's URL. Should not be called if self.item_author_name has returned None.

item_author_name(*item*)
Return the first author of an entry.

item_categories(*item*)
Entry's categories.

item_enclosure_length(*item*)
Try to obtain the size of the enclosure if it's present on the FS, otherwise returns an hardcoded value. Note: this method is only called if item_enclosure_url has returned something.

item_enclosure_mime_type(*item*)
Guess the enclosure's mimetype. Note: this method is only called if item_enclosure_url has returned something.

item_enclosure_url(*item*)
Return an image for enclosure.

item_pubdate(*item*)
Publication date of an entry.

item_updateddate(*item*)
Update date of an entry.

title_template = 'feeds/entry_title.html'

class zinnia.feeds.**EntryPingbacks**
Bases: *zinnia.feeds.EntryDiscussions*

Feed for pingbacks on an entry.

description(*obj*)
Description of the feed.

description_template = 'feeds/pingback_description.html'

get_title(*obj*)
Title of the feed.

item_link(*item*)
URL of the pingback.

items(*obj*)
Items are the pingbacks on the entry.

title_template = 'feeds/pingback_title.html'

class zinnia.feeds.**EntryTrackbacks**
Bases: *zinnia.feeds.EntryDiscussions*

Feed for trackbacks on an entry.

description(*obj*)
Description of the feed.

description_template = 'feeds/trackback_description.html'

get_title(*obj*)
Title of the feed.

item_link(*item*)
URL of the trackback.

items(*obj*)
Items are the trackbacks on the entry.

title_template = 'feeds/trackback_title.html'

class zinnia.feeds.**LastDiscussions**
Bases: *zinnia.feeds.DiscussionFeed*

Feed for the last discussions.

description()
Description of the feed.

get_title(*obj*)
Title of the feed.

items()
Items are the discussions on the entries.

link()
URL of last discussions.

class zinnia.feeds.**LastEntries**
Bases: *zinnia.feeds.EntryFeed*

Feed for the last entries.

description()
Description of the feed.

get_title(*obj*)
Title of the feed

items()
Items are published entries.

link()
URL of last entries.

class zinnia.feeds.**SearchEntries**
Bases: *zinnia.feeds.EntryFeed*

Feed filtered by a search pattern.

description(*obj*)
Description of the feed.

get_object(*request*)
The GET parameter 'pattern' is the object.

get_title(*obj*)
Title of the feed.

items(*obj*)
Items are the published entries founds.

link(*obj*)
URL of the search request.

class zinnia.feeds.**TagEntries**
Bases: *zinnia.feeds.EntryFeed*

Feed filtered by a tag.

description(*obj*)
Description of the feed.

get_object(*request*, *tag*)
Retrieve the tag by his name.

get_title(*obj*)
Title of the feed.

items(*obj*)
Items are the published entries of the tag.

link(*obj*)
URL of the tag.

class zinnia.feeds.**ZinniaFeed**
Bases: django.contrib.syndication.views.Feed

Base Feed class for the Zinnia application, enriched for a more convenient usage.

feed_copyright = 'Zinnia'

feed_format = 'rss'

get_title(*obj*)

limit = 15

protocol = 'http'

site
Acquire the current site used.

site_url
Return the URL of the current site.

title(*obj=None*)
Title of the feed prefixed with the site name.

flags Module

Comment flags for Zinnia

zinnia.flags.**get_user_flagger**(**args*, ***kwds*)
Return an User instance used by the system when flagging a comment as trackback or pingback.

managers Module

Managers of Zinnia

class zinnia.managers.**EntryPublishedManager**
Bases: django.db.models.manager.Manager

Manager to retrieve published entries.

advanced_search(*pattern*)
Advanced search on entries.

basic_search(*pattern*)
Basic search on entries.

get_queryset()
Return published entries.

on_site()
Return entries published on current site.

search(*pattern*)
Top level search method on entries.

class zinnia.managers.**EntryRelatedPublishedManager**
Bases: django.db.models.manager.Manager

Manager to retrieve objects associated with published entries.

get_queryset()
Return a queryset containing published entries.

zinnia.managers.**entries_published**(*queryset*)
Return only the entries published.

zinnia.managers.**tags_published**()
Return the published tags.

markups Module

Set of" markup" function to transform plain text into HTML for Zinnia. Code originally provided by django.contrib.markups

zinnia.markups.**html_format**(*value*)
Returns the value formatted in HTML, depends on MARKUP_LANGUAGE setting.

zinnia.markups.**markdown**(*value*, *extensions=[]*)
Markdown processing with optionally using various extensions that python-markdown supports. *extensions* is an iterable of either markdown.Extension instances or extension paths.

zinnia.markups.**restructuredtext**(*value*, *settings={}*)
RestructuredText processing with optionnally custom settings.

zinnia.markups.**textile**(*value*)
Textile processing.

moderator Module

Moderator of Zinnia comments

class zinnia.moderator.**EntryCommentModerator**(*model*)
Bases: django_comments.moderation.CommentModerator

Moderate the comments on entries.

auto_close_field = 'start_publication'

auto_moderate_comments = False

close_after = None

do_email_authors(*comment*, *entry*, *site*)
Send email notification of a new comment to the authors of the entry.

do_email_notification(*comment*, *entry*, *site*)
Send email notification of a new comment to site staff.

do_email_reply(*comment*, *entry*, *site*)
Send email notification of a new comment to the authors of the previous comments.

email(*comment*, *entry*, *request*)
Send email notifications needed.

email_authors = True

email_reply = False

enable_field = 'comment_enabled'

mail_comment_notification_recipients = []

moderate(*comment*, *entry*, *request*)
Determine if a new comment should be marked as non-public and await approval. Return `True` to put the comment into the moderator queue, or `False` to allow it to be showed up immediately.

spam_checker_backends = []

ping Module

Pings utilities for Zinnia

class zinnia.ping.**DirectoryPinger**(*server_name*, *entries*, *timeout=10*)
Bases: threading.Thread

Threaded web directory pinger.

ping_entry(*entry*)
Ping an entry to a directory.

run()
Ping entries to a directory in a thread.

class zinnia.ping.**ExternalUrlsPinger**(*entry*, *timeout=10*)
Bases: threading.Thread

Threaded external URLs pinger.

find_external_urls(*entry*)
Find external URLs in an entry.

find_pingback_href(*content*)
Try to find LINK markups to pingback URL.

find_pingback_urls(*urls*)
Find the pingback URL for each URLs.

is_external_url(*url*, *site_url*)
Check if the URL is an external URL.

pingback_url(*server_name*, *target_url*)
Do a pingback call for the target URL.

run()
Ping external URLs in a Thread.

class zinnia.ping.**URLRessources**
Bases: object

Object defining the ressources of the Website.

preview Module

Preview for Zinnia

class zinnia.preview.**HTMLPreview**(*content, lead='', splitters=['<!– more –>', '<!–more–>'], max_words=55, more_string='...'*)
Bases: object

Build an HTML preview of an HTML content.

build_preview()
Build the preview by:

- Returning the lead attribut if not empty.
- Checking if a split marker is present in the content Then split the content with the marker to build the preview.
- Splitting the content to a fixed number of words.

displayed_percent
Return the percentage of the content displayed in the preview.

displayed_words
Return the number of words displayed in the preview.

has_more
Boolean telling if the preview has hidden content.

preview
The preview is a cached property.

remaining_percent
Return the percentage of the content remaining after the preview.

remaining_words
Return the number of words remaining after the preview.

split(*splitter*)
Split the HTML content with a marker without breaking closing markups.

total_words
Return the total of words contained in the content and in the lead.

truncate()
Truncate the content with the Truncator object.

search Module

Search module with complex query parsing for Zinnia

zinnia.search.**advanced_search**(*pattern*)
Parse the grammar of a pattern and build a queryset with it.

zinnia.search.**create_q**(*token*)
Creates the Q() object.

zinnia.search.**union_q**(*token*)
Appends all the Q() objects.

signals Module

Signal handlers of Zinnia

zinnia.signals.**connect_discussion_signals**()
Connect all the signals on the Comment model to maintains a valid discussion count on each entries when an action is done with the comments.

zinnia.signals.**connect_entry_signals**()
Connect all the signals on Entry model.

zinnia.signals.**count_comments_handler**(*sender*, ***kwargs*)
Update Entry.comment_count when a public comment was posted.

zinnia.signals.**count_discussions_handler**(*sender*, ***kwargs*)
Update the count of each type of discussion on an entry.

zinnia.signals.**count_pingbacks_handler**(*sender*, ***kwargs*)
Update Entry.pingback_count when a pingback was posted.

zinnia.signals.**count_trackbacks_handler**(*sender*, ***kwargs*)
Update Entry.trackback_count when a trackback was posted.

zinnia.signals.**disable_for_loaddata**(*signal_handler*)
Decorator for disabling signals sent by 'post_save' on loaddata command. http://code.djangoproject.com/ticket/8399

zinnia.signals.**disconnect_discussion_signals**()
Disconnect all the signals on Comment model provided by Zinnia.

zinnia.signals.**disconnect_entry_signals**()
Disconnect all the signals on Entry model.

zinnia.signals.**flush_similar_cache_handler**(**args*, ***kwargs*)
Flush the cache of similar entries when an entry is saved.

zinnia.signals.**ping_directories_handler**(**args*, ***kwargs*)
Ping directories when an entry is saved.

zinnia.signals.**ping_external_urls_handler**(**args*, ***kwargs*)
Ping externals URLS when an entry is saved.

sitemaps Module

Sitemaps for Zinnia

class zinnia.sitemaps.**AuthorSitemap**
Bases: *zinnia.sitemaps.EntryRelatedSitemap*

Sitemap for authors.

model
alias of Author

class zinnia.sitemaps.**CategorySitemap**
Bases: *zinnia.sitemaps.EntryRelatedSitemap*

Sitemap for categories.

model
alias of Category

class zinnia.sitemaps.**EntryRelatedSitemap**
Bases: *zinnia.sitemaps.ZinniaSitemap*

Sitemap for models related to Entries.

cache_infos(*queryset*)
Cache infos like the number of entries published and the last modification date for standardized access later.

changefreq = 'monthly'

get_queryset()
Build a queryset of items with published entries and annotated with the number of entries and the latest modification date.

items()
Get a queryset, cache infos for standardized access to them later then compute the maximum of entries to define the priority of each items.

lastmod(*item*)
The last modification date is defined by the latest entry last update in the cache.

model = None

priority(*item*)
The priority of the item depends of the number of entries published in the cache divided by the maximum of entries.

set_max_entries()
Define the maximum of entries for computing the priority of each items later.

class zinnia.sitemaps.**EntrySitemap**
Bases: *zinnia.sitemaps.ZinniaSitemap*

Sitemap for entries.

changefreq = 'weekly'

items()
Return published entries.

lastmod(*obj*)
Return last modification of an entry.

priority = 0.5

class zinnia.sitemaps.**TagSitemap**
Bases: *zinnia.sitemaps.EntryRelatedSitemap*

Sitemap for tags.

cache_infos(*queryset*)
Cache the number of entries published and the last modification date under each tag.

get_queryset()
Return the published Tags with option counts.

location(*item*)
Return URL of the tag.

class zinnia.sitemaps.**ZinniaSitemap**
Bases: django.contrib.sitemaps.Sitemap

Base Sitemap class for Zinnia.

protocol = 'http'

templates Module

Templates module for Zinnia

zinnia.templates.**append_position**(*path*, *position*, *separator=''*)
Concatenate a path and a position, between the filename and the extension.

zinnia.templates.**loop_template_list**(*loop_positions*, *instance*, *instance_type*, *default_template*, *registry*)
Build a list of templates from a position within a loop and a registry of templates.

Subpackages

admin Package

admin Package

Admin of Zinnia

category Module

CategoryAdmin for Zinnia

class zinnia.admin.category.**CategoryAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

Admin for Category model.

fields = ('title', 'parent', 'description', 'slug')

form
alias of CategoryAdminForm

get_tree_path(*category*)
Return the category's tree path in HTML.

list_display = ('title', 'slug', 'get_tree_path', 'description')

list_filter = ('parent',)

media

prepopulated_fields = {'slug': ('title',)}

search_fields = ('title', 'description')

entry Module

EntryAdmin for Zinnia

class zinnia.admin.entry.**EntryAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

Admin for Entry model.

actions = [u'make_mine', u'make_published', u'make_hidden', u'close_comments', u'close_pingbacks', u'close_trackb

actions_on_bottom = True

actions_on_top = True

close_comments(*request*, *queryset*)
Close the comments for selected entries.

close_pingbacks(*request*, *queryset*)
Close the pingbacks for selected entries.

close_trackbacks(*request*, *queryset*)
Close the trackbacks for selected entries.

date_hierarchy = u'publication_date'

fieldsets = ((<django.utils.functional.__proxy__ object>, {u'fields': ((u'title', u'status'), u'lead', u'content')}), (<djang

filter_horizontal = (u'categories', u'authors', u'related')

form
alias of EntryAdminForm

formfield_for_manytomany(*db_field*, *request*, ***kwargs*)
Filter the disposable authors.

get_actions(*request*)
Define actions by user's permissions.

get_authors(*entry*)
Return the authors in HTML.

get_categories(*entry*)
Return the categories linked in HTML.

get_changeform_initial_data(*request*)
Provide initial datas when creating an entry.

get_is_visible(*entry*)
Admin wrapper for entry.is_visible.

get_queryset(*request*)
Make special filtering by user's permissions.

get_readonly_fields(*request*, *obj=None*)
Return readonly fields by user's permissions.

get_short_url(*entry*)
Return the short url in HTML.

get_sites(*entry*)
Return the sites linked in HTML.

get_tags(*entry*)
Return the tags linked in HTML.

get_title(*entry*)
Return the title with word count and number of comments.

list_display = (u'get_title', u'get_authors', u'get_categories', u'get_tags', u'get_sites', u'get_is_visible', u'featured',

list_filter = (<class 'zinnia.admin.filters.CategoryListFilter'>, <class 'zinnia.admin.filters.AuthorListFilter'>, u'pub

make_hidden(*request*, *queryset*)
Set entries selected as hidden.

make_mine(*request*, *queryset*)
Set the entries to the current user.

make_published(*request*, *queryset*)
Set entries selected as published.

mark_featured(*request*, *queryset*)
Mark selected as featured post.

media

ping_directories(*request*, *queryset*, *messages=True*)
Ping web directories for selected entries.

prepopulated_fields = {u'slug': (u'title',)}

put_on_top(*request*, *queryset*)
Put the selected entries on top at the current date.

radio_fields = {u'content_template': 2, u'detail_template': 2}

search_fields = (u'title', u'excerpt', u'content', u'tags')

unmark_featured(*request*, *queryset*)
Un-Mark selected featured posts.

fields Module

Fields for Zinnia admin

class zinnia.admin.fields.**MPTTModelChoiceIterator**(*field*)
Bases: django.forms.models.ModelChoiceIterator

MPTT version of ModelChoiceIterator.

choice(*obj*)
Overloads the choice method to add the position of the object in the tree for future sorting.

class zinnia.admin.fields.**MPTTModelMultipleChoiceField**(*level_indicator='|–'*, **args*, ***kwargs*)
Bases: django.forms.models.ModelMultipleChoiceField

MPTT version of ModelMultipleChoiceField.

choices
Override the _get_choices method to use MPTTModelChoiceIterator.

label_from_instance(*obj*)
Create labels which represent the tree level of each node when generating option labels.

filters Module

Filters for Zinnia admin

class zinnia.admin.filters.**AuthorListFilter**(*request*, *params*, *model*, *model_admin*)
Bases: *zinnia.admin.filters.RelatedPublishedFilter*

List filter for EntryAdmin with published authors only.

lookup_key = 'authors__id'

model
alias of Author

parameter_name = 'author'

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

class zinnia.admin.filters.**CategoryListFilter**(*request*, *params*, *model*, *model_admin*)
Bases: *zinnia.admin.filters.RelatedPublishedFilter*

List filter for EntryAdmin about categories with published entries.

lookup_key = 'categories__id'

model
alias of Category

parameter_name = 'category'

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

class zinnia.admin.filters.**RelatedPublishedFilter**(*request*, *params*, *model*, *model_admin*)
Bases: django.contrib.admin.filters.SimpleListFilter

Base filter for related objects to published entries.

lookup_key = None

lookups(*request*, *model_admin*)
Return published objects with the number of entries.

model = None

queryset(*request*, *queryset*)
Return the object's entries if a value is set.

forms Module

Forms for Zinnia admin

class zinnia.admin.forms.**CategoryAdminForm**(**args*, ***kwargs*)
Bases: django.forms.models.ModelForm

Form for Category's Admin.

class Meta
CategoryAdminForm's Meta.

fields = u'__all__'

model
alias of Category

CategoryAdminForm.**admin_site = <django.contrib.admin.sites.AdminSite object>**

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

CategoryAdminForm.**clean_parent**()
Check if category parent is not selfish.

CategoryAdminForm.**declared_fields = OrderedDict([('parent', <mptt.forms.TreeNodeChoiceField object>)])**

CategoryAdminForm.**media**

class zinnia.admin.forms.**EntryAdminForm**(**args*, ***kwargs*)
Bases: django.forms.models.ModelForm

Form for Entry's Admin.

class Meta
EntryAdminForm's Meta.

fields = u'__all__'

model
alias of Entry

widgets = {'excerpt': <class 'zinnia.admin.widgets.MiniTextarea'>, 'image_caption': <class 'zinnia.admin.widgets

EntryAdminForm.**admin_site = <django.contrib.admin.sites.AdminSite object>**

EntryAdminForm.**base_fields = OrderedDict([('title', <django.forms.fields.CharField object>), ('slug', <django.for**

EntryAdminForm.**declared_fields = OrderedDict([('categories', <zinnia.admin.fields.MPTTModelMultipleChoic**

EntryAdminForm.**media**

widgets Module

Widgets for Zinnia admin

class zinnia.admin.widgets.**MPTTFilteredSelectMultiple**(*verbose_name*, *is_stacked=False*, *attrs=None*, *choices=()*)
Bases: django.contrib.admin.widgets.FilteredSelectMultiple

MPTT version of FilteredSelectMultiple.

media
MPTTFilteredSelectMultiple's Media.

render_option(*selected_choices*, *option_value*, *option_label*, *sort_fields*)
Overrides the render_option method to handle the sort_fields argument.

render_options(*selected_choices*)
This is copy'n'pasted from django.forms.widgets Select(Widget) change to the for loop and render_option so they will unpack and use our extra tuple of mptt sort fields (if you pass in some default choices for this field, make sure they have the extra tuple too!).

class zinnia.admin.widgets.**MiniTextarea**(*attrs=None*)
Bases: django.contrib.admin.widgets.AdminTextareaWidget

Vertically shorter version of the admin textarea widget.

media

rows = 2

class zinnia.admin.widgets.**TagAutoComplete**(*attrs=None*)
Bases: django.contrib.admin.widgets.AdminTextInputWidget

Tag widget with autocompletion based on select2.

get_tags()
Returns the list of tags to auto-complete.

media
TagAutoComplete's Media.

render(*name*, *value*, *attrs=None*)
Render the default widget and initialize select2.

models Package

models Package

Models for Zinnia

class zinnia.models.**Entry**(**args*, ***kwargs*)
Bases: *zinnia.models_bases.entry.AbstractEntry*

The final Entry model based on inheritence.

Parameters

- **id** (*AutoField*) – Id
- **title** (*CharField*) – Title
- **slug** (*SlugField*) – Used to build the entry's URL.

- **status** (*IntegerField*) – Status
- **publication_date** (*DateTimeField*) – Used to build the entry's URL.
- **start_publication** (*DateTimeField*) – Start date of publication.
- **end_publication** (*DateTimeField*) – End date of publication.
- **creation_date** (*DateTimeField*) – Creation date
- **last_update** (*DateTimeField*) – Last update
- **content** (*TextField*) – Content
- **comment_enabled** (*BooleanField*) – Allows comments if checked.
- **pingback_enabled** (*BooleanField*) – Allows pingbacks if checked.
- **trackback_enabled** (*BooleanField*) – Allows trackbacks if checked.
- **comment_count** (*IntegerField*) – Comment count
- **pingback_count** (*IntegerField*) – Pingback count
- **trackback_count** (*IntegerField*) – Trackback count
- **lead** (*TextField*) – Lead paragraph
- **excerpt** (*TextField*) – Used for SEO purposes.
- **image** (*ImageField*) – Used for illustration.
- **image_caption** (*TextField*) – Image's caption.
- **featured** (*BooleanField*) – Featured
- **tags** (*TagField*) – Tags
- **login_required** (*BooleanField*) – Only authenticated users can view the entry.
- **password** (*CharField*) – Protects the entry with a password.
- **content_template** (*CharField*) – Template used to display the entry's content.
- **detail_template** (*CharField*) – Template used to display the entry's detail page.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

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

Entry.**authors**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

Entry.**categories**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Entry.`**`get_content_template_display`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_detail_template_display`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_next_by_creation_date`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_next_by_last_update`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_next_by_publication_date`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_previous_by_creation_date`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_previous_by_last_update`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_previous_by_publication_date`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_status_display`**(**moreargs*, ***morekwargs*)

`Entry.`**`id`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`Entry.`**`related`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Entry.`**`sites`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class `zinnia.models.`**`Author`**(**args*, ***kwargs*)
Bases: `django.contrib.auth.models.User`, *`zinnia.models.author.AuthorPublishedManager`*

Proxy model around `django.contrib.auth.models.get_user_model`.

Parameters

- **`id`** (*`AutoField`*) – Id

- **password** (*CharField*) – Password
- **last_login** (*DateTimeField*) – Last login
- **is_superuser** (*BooleanField*) – Designates that this user has all permissions without explicitly assigning them.
- **username** (*CharField*) – Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
- **first_name** (*CharField*) – First name
- **last_name** (*CharField*) – Last name
- **email** (*EmailField*) – Email address
- **is_staff** (*BooleanField*) – Designates whether the user can log into this admin site.
- **is_active** (*BooleanField*) – Designates whether this user should be treated as active. Unselect this instead of deleting accounts.
- **date_joined** (*DateTimeField*) – Date joined

exception DoesNotExist
Bases: `django.contrib.auth.models.DoesNotExist`

exception `Author.`**MultipleObjectsReturned**
Bases: `django.contrib.auth.models.MultipleObjectsReturned`

`Author.`**entries**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Author.`**entries_published**`()`
Returns author's published entries.

`Author.`**get_absolute_url**`(`**args*, ***kwargs*`)`
Builds and returns the author's URL based on his username.

class `zinnia.models.`**Category**`(`**args*, ***kwargs*`)`
Bases: `mptt.models.MPTTModel`

Simple model for categorizing entries.

Parameters

- **id** (*AutoField*) – Id
- **title** (*CharField*) – Title
- **slug** (*SlugField*) – Used to build the category's URL.
- **description** (*TextField*) – Description
- **parent_id** (*TreeForeignKey*) – Parent category
- **lft** (*PositiveIntegerField*) – Lft

- **rght** (*PositiveIntegerField*) – Rght
- **tree_id** (*PositiveIntegerField*) – Tree id
- **level** (*PositiveIntegerField*) – Level

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

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

Category.**children**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

Category.**description**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**entries**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

Category.**entries_published**()
Returns category's published entries.

Category.**get_absolute_url**(**args*, ***kwargs*)
Builds and returns the category's URL based on his tree path.

Category.**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**level**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**lft**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**objects = <mptt.managers.TreeManager object>**

Category.**parent**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

Category.**parent_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**published = <zinnia.managers.EntryRelatedPublishedManager object>**

Category.**rght**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**slug**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**title**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**tree_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**tree_path**
Returns category's tree path by concatening the slug of his ancestors.

author Module

Author model for Zinnia

class zinnia.models.author.**Author**(**args*, ***kwargs*)
Bases: django.contrib.auth.models.User, *zinnia.models.author.AuthorPublishedManager*

Proxy model around django.contrib.auth.models.get_user_model.

Parameters

- **id** (*AutoField*) – Id
- **password** (*CharField*) – Password
- **last_login** (*DateTimeField*) – Last login
- **is_superuser** (*BooleanField*) – Designates that this user has all permissions without explicitly assigning them.
- **username** (*CharField*) – Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
- **first_name** (*CharField*) – First name
- **last_name** (*CharField*) – Last name

- **email** (*EmailField*) – Email address
- **is_staff** (*BooleanField*) – Designates whether the user can log into this admin site.
- **is_active** (*BooleanField*) – Designates whether this user should be treated as active. Unselect this instead of deleting accounts.
- **date_joined** (*DateTimeField*) – Date joined

exception DoesNotExist
Bases: `django.contrib.auth.models.DoesNotExist`

exception `Author.`**MultipleObjectsReturned**
Bases: `django.contrib.auth.models.MultipleObjectsReturned`

`Author.`**entries**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Author.`**entries_published**`()`
Returns author's published entries.

`Author.`**get_absolute_url**`(`**args*, ***kwargs*`)`
Builds and returns the author's URL based on his username.

class `zinnia.models.author.`**AuthorPublishedManager**`(`**args*, ***kwargs*`)`
Bases: `django.db.models.base.Model`

Proxy model manager to avoid overriding of the default User's manager and issue #307.

class Meta

abstract = False

`AuthorPublishedManager.`**published**

`zinnia.models.author.`**safe_get_user_model**`()`
Safe loading of the User model, customized or not.

`category` Module

Category model for Zinnia

class `zinnia.models.category.`**Category**`(`**args*, ***kwargs*`)`
Bases: `mptt.models.MPTTModel`

Simple model for categorizing entries.

Parameters

- **id** (*AutoField*) – Id
- **title** (*CharField*) – Title

- **slug** (*SlugField*) – Used to build the category's URL.
- **description** (*TextField*) – Description
- **parent_id** (*TreeForeignKey*) – Parent category
- **lft** (*PositiveIntegerField*) – Lft
- **rght** (*PositiveIntegerField*) – Rght
- **tree_id** (*PositiveIntegerField*) – Tree id
- **level** (*PositiveIntegerField*) – Level

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `Category.`**MultipleObjectsReturned**
Bases: `django.core.exceptions.MultipleObjectsReturned`

`Category.`**children**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Category.`**description**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`Category.`**entries**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Category.`**entries_published**`()`
Returns category's published entries.

`Category.`**get_absolute_url**`(`*args*, ***kwargs*`)`
Builds and returns the category's URL based on his tree path.

`Category.`**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`Category.`**level**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**lft**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**objects = <mptt.managers.TreeManager object>**

Category.**parent**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

Category.**parent_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**published = <zinnia.managers.EntryRelatedPublishedManager object>**

Category.**rght**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**slug**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**title**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**tree_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Category.**tree_path**
Returns category's tree path by concatening the slug of his ancestors.

`entry` Module

Entry model for Zinnia

class zinnia.models.entry.**Entry**(**args*, ***kwargs*)
Bases: *zinnia.models_bases.entry.AbstractEntry*

The final Entry model based on inheritence.

Parameters

- **id** (*AutoField*) – Id
- **title** (*CharField*) – Title
- **slug** (*SlugField*) – Used to build the entry's URL.
- **status** (*IntegerField*) – Status
- **publication_date** (*DateTimeField*) – Used to build the entry's URL.
- **start_publication** (*DateTimeField*) – Start date of publication.

- **end_publication** (*DateTimeField*) – End date of publication.
- **creation_date** (*DateTimeField*) – Creation date
- **last_update** (*DateTimeField*) – Last update
- **content** (*TextField*) – Content
- **comment_enabled** (*BooleanField*) – Allows comments if checked.
- **pingback_enabled** (*BooleanField*) – Allows pingbacks if checked.
- **trackback_enabled** (*BooleanField*) – Allows trackbacks if checked.
- **comment_count** (*IntegerField*) – Comment count
- **pingback_count** (*IntegerField*) – Pingback count
- **trackback_count** (*IntegerField*) – Trackback count
- **lead** (*TextField*) – Lead paragraph
- **excerpt** (*TextField*) – Used for SEO purposes.
- **image** (*ImageField*) – Used for illustration.
- **image_caption** (*TextField*) – Image's caption.
- **featured** (*BooleanField*) – Featured
- **tags** (*TagField*) – Tags
- **login_required** (*BooleanField*) – Only authenticated users can view the entry.
- **password** (*CharField*) – Protects the entry with a password.
- **content_template** (*CharField*) – Template used to display the entry's content.
- **detail_template** (*CharField*) – Template used to display the entry's detail page.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `Entry.`**MultipleObjectsReturned**
Bases: `django.core.exceptions.MultipleObjectsReturned`

`Entry.`**authors**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Entry.`**categories**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Entry.`**`get_content_template_display`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_detail_template_display`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_next_by_creation_date`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_next_by_last_update`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_next_by_publication_date`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_previous_by_creation_date`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_previous_by_last_update`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_previous_by_publication_date`**(**moreargs*, ***morekwargs*)

`Entry.`**`get_status_display`**(**moreargs*, ***morekwargs*)

`Entry.`**`id`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`Entry.`**`related`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Entry.`**`sites`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

models_bases Package

`models_bases` Package

Base models for Zinnia

`zinnia.models_bases.`**`load_model_class`**(*model_path*)
Load by import a class by a string path like: 'module.models.MyModel'. This mechanism allows extension and customization of the Entry model class.

entry Module

Base entry models for Zinnia

class zinnia.models_bases.entry.**AbstractEntry**(**args*, ***kwargs*)

Bases: *zinnia.models_bases.entry.CoreEntry*, *zinnia.models_bases.entry.ContentEntry*, *zinnia.models_bases.entry.DiscussionsEntry*, *zinnia.models_bases.entry.RelatedEntry*, *zinnia.models_bases.entry.LeadEntry*, *zinnia.models_bases.entry.ExcerptEntry*, *zinnia.models_bases.entry.ImageEntry*, *zinnia.models_bases.entry.FeaturedEntry*, *zinnia.models_bases.entry.AuthorsEntry*, *zinnia.models_bases.entry.CategoriesEntry*, *zinnia.models_bases.entry.TagsEntry*, *zinnia.models_bases.entry.LoginRequiredEntry*, *zinnia.models_bases.entry.PasswordRequiredEntry*, *zinnia.models_bases.entry.ContentTemplateEntry*, *zinnia.models_bases.entry.DetailTemplateEntry*

Final abstract entry model class assembling all the abstract entry model classes into a single one.

In this manner we can override some fields without reimplemting all the AbstractEntry.

Parameters

- **title** (*CharField*) – Title
- **slug** (*SlugField*) – Used to build the entry's URL.
- **status** (*IntegerField*) – Status
- **publication_date** (*DateTimeField*) – Used to build the entry's URL.
- **start_publication** (*DateTimeField*) – Start date of publication.
- **end_publication** (*DateTimeField*) – End date of publication.
- **creation_date** (*DateTimeField*) – Creation date
- **last_update** (*DateTimeField*) – Last update
- **content** (*TextField*) – Content
- **comment_enabled** (*BooleanField*) – Allows comments if checked.
- **pingback_enabled** (*BooleanField*) – Allows pingbacks if checked.
- **trackback_enabled** (*BooleanField*) – Allows trackbacks if checked.
- **comment_count** (*IntegerField*) – Comment count
- **pingback_count** (*IntegerField*) – Pingback count
- **trackback_count** (*IntegerField*) – Trackback count
- **lead** (*TextField*) – Lead paragraph
- **excerpt** (*TextField*) – Used for SEO purposes.
- **image** (*ImageField*) – Used for illustration.
- **image_caption** (*TextField*) – Image's caption.
- **featured** (*BooleanField*) – Featured
- **tags** (*TagField*) – Tags
- **login_required** (*BooleanField*) – Only authenticated users can view the entry.
- **password** (*CharField*) – Protects the entry with a password.
- **content_template** (*CharField*) – Template used to display the entry's content.

- **detail_template** (*CharField*) – Template used to display the entry's detail page.

class Meta

Bases: zinnia.models_bases.entry.Meta

abstract = False

AbstractEntry.**authors**

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

AbstractEntry.**categories**

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

AbstractEntry.**get_content_template_display**(**moreargs*, ***morekwargs*)

AbstractEntry.**get_detail_template_display**(**moreargs*, ***morekwargs*)

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

AbstractEntry.**get_next_by_last_update**(**moreargs*, ***morekwargs*)

AbstractEntry.**get_next_by_publication_date**(**moreargs*, ***morekwargs*)

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

AbstractEntry.**get_previous_by_last_update**(**moreargs*, ***morekwargs*)

AbstractEntry.**get_previous_by_publication_date**(**moreargs*, ***morekwargs*)

AbstractEntry.**get_status_display**(**moreargs*, ***morekwargs*)

AbstractEntry.**related**

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

AbstractEntry.**sites**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

class zinnia.models_bases.entry.**AuthorsEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class to add relationship between the entries and their authors.

class Meta

abstract = False

AuthorsEntry.**authors**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

class zinnia.models_bases.entry.**CategoriesEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class to categorize the entries.

class Meta

abstract = False

CategoriesEntry.**categories**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

class zinnia.models_bases.entry.**ContentEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract content model class providing field and methods to write content inside an entry.

Parameters **content** (*TextField*) – Content

class Meta

abstract = False

ContentEntry.**content**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

ContentEntry.**html_content**
Returns the "content" field formatted in HTML.

ContentEntry.**html_preview**
Returns a preview of the "content" field or the "lead" field if defined, formatted in HTML.

ContentEntry.**word_count**
Counts the number of words used in the content.

class zinnia.models_bases.entry.**ContentTemplateEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class to display entry's content with a custom template.

Parameters **content_template** (*CharField*) – Template used to display the entry's content.

class Meta

abstract = False

ContentTemplateEntry.**content_template**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

ContentTemplateEntry.**get_content_template_display**(**moreargs*, ***morekwargs*)

class zinnia.models_bases.entry.**CoreEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract core entry model class providing the fields and methods required for publishing content over time.

Parameters

- **title** (*CharField*) – Title
- **slug** (*SlugField*) – Used to build the entry's URL.
- **status** (*IntegerField*) – Status
- **publication_date** (*DateTimeField*) – Used to build the entry's URL.
- **start_publication** (*DateTimeField*) – Start date of publication.
- **end_publication** (*DateTimeField*) – End date of publication.
- **creation_date** (*DateTimeField*) – Creation date
- **last_update** (*DateTimeField*) – Last update

class Meta
CoreEntry's meta informations.

abstract = False

get_latest_by = 'publication_date'

index_together = [['slug', 'publication_date'], ['status', 'publication_date', 'start_publication', 'end_publication

ordering = ['-publication_date']

permissions = (('can_view_all', 'Can view all entries'), ('can_change_status', 'Can change status'), ('can_change_

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

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

CoreEntry.**STATUS_CHOICES = ((0, <django.utils.functional.__proxy__ object>), (1, <django.utils.functional.__proxy_**

CoreEntry.**creation_date**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

CoreEntry.**end_publication**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

CoreEntry.**get_absolute_url**(**args*, ***kwargs*)
Builds and returns the entry's URL based on the slug and the creation date.

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

CoreEntry.**get_next_by_last_update**(**moreargs*, ***morekwargs*)

CoreEntry.**get_next_by_publication_date**(**moreargs*, ***morekwargs*)

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

CoreEntry.**get_previous_by_last_update**(**moreargs*, ***morekwargs*)

CoreEntry.**get_previous_by_publication_date**(**moreargs*, ***morekwargs*)

CoreEntry.**get_status_display**(**moreargs*, ***morekwargs*)

CoreEntry.**is_actual**
Checks if an entry is within his publication period.

CoreEntry.**is_visible**
Checks if an entry is visible and published.

CoreEntry.**last_update**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

CoreEntry.**next_entry**
Returns the next published entry if exists.

CoreEntry.**objects**

CoreEntry.**previous_entry**
Returns the previous published entry if exists.

CoreEntry.**previous_next_entries**
Returns and caches a tuple containing the next and previous published entries. Only available if the entry instance is published.

CoreEntry.**publication_date**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

CoreEntry.**published**

CoreEntry.**save**(**args*, ***kwargs*)
Overrides the save method to update the the last_update field.

CoreEntry.**short_url**
Returns the entry's short url.

CoreEntry.**sites**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

CoreEntry.**slug**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

CoreEntry.**start_publication**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

CoreEntry.**status**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

CoreEntry.**title**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class zinnia.models_bases.entry.**DetailTemplateEntry**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Abstract model class to display entries with a custom template if needed on the detail page.

Parameters **detail_template** (*CharField*) – Template used to display the entry's detail page.

class Meta

abstract = False

DetailTemplateEntry.**detail_template**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

DetailTemplateEntry.**get_detail_template_display**(**moreargs*, ***morekwargs*)

class zinnia.models_bases.entry.**DiscussionsEntry**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Abstract discussion model class providing the fields and methods to manage the discussions (comments, pingbacks, trackbacks).

Parameters

- **comment_enabled** (*BooleanField*) – Allows comments if checked.
- **pingback_enabled** (*BooleanField*) – Allows pingbacks if checked.
- **trackback_enabled** (*BooleanField*) – Allows trackbacks if checked.

- **comment_count** (*IntegerField*) – Comment count
- **pingback_count** (*IntegerField*) – Pingback count
- **trackback_count** (*IntegerField*) – Trackback count

class Meta

abstract = False

DiscussionsEntry.**comment_count**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

DiscussionsEntry.**comment_enabled**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

DiscussionsEntry.**comments**
Returns a queryset of the published comments.

DiscussionsEntry.**comments_are_open**
Checks if the comments are open with the AUTO_CLOSE_COMMENTS_AFTER setting.

DiscussionsEntry.**discussion_is_still_open**(*discussion_type*, *auto_close_after*)
Checks if a type of discussion is still open are a certain number of days.

DiscussionsEntry.**discussions**
Returns a queryset of the published discussions.

DiscussionsEntry.**pingback_count**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

DiscussionsEntry.**pingback_enabled**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

DiscussionsEntry.**pingbacks**
Returns a queryset of the published pingbacks.

DiscussionsEntry.**pingbacks_are_open**
Checks if the pingbacks are open with the AUTO_CLOSE_PINGBACKS_AFTER setting.

DiscussionsEntry.**trackback_count**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

DiscussionsEntry.**trackback_enabled**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

DiscussionsEntry.**trackbacks**
Return a queryset of the published trackbacks.

DiscussionsEntry.**trackbacks_are_open**
Checks if the trackbacks are open with the AUTO_CLOSE_TRACKBACKS_AFTER setting.

class zinnia.models_bases.entry.**ExcerptEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class to add an excerpt to the entries.

Parameters **excerpt** (*TextField*) – Used for SEO purposes.

class Meta

abstract = False

ExcerptEntry.**excerpt**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

ExcerptEntry.**save**(**args*, ***kwargs*)
Overrides the save method to create an excerpt from the content field if void.

class zinnia.models_bases.entry.**FeaturedEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class to mark entries as featured.

Parameters **featured** (*BooleanField*) – Featured

class Meta

abstract = False

FeaturedEntry.**featured**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class zinnia.models_bases.entry.**ImageEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class to add an image for illustrating the entries.

Parameters

- **image** (*ImageField*) – Used for illustration.
- **image_caption** (*TextField*) – Image's caption.

class Meta

abstract = False

ImageEntry.**image_caption**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

ImageEntry.**image_upload_to**(*filename*)
Compute the upload path for the image field.

class zinnia.models_bases.entry.**LeadEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class providing a lead content to the entries.

Parameters **lead** (*TextField*) – Lead paragraph

class Meta

abstract = False

LeadEntry.**html_lead**
Returns the "lead" field formatted in HTML.

LeadEntry.**lead**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class zinnia.models_bases.entry.**LoginRequiredEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class to restrcit the display of the entry on authenticated users.

Parameters **login_required** (*BooleanField*) – Only authenticated users can view the entry.

class Meta

abstract = False

LoginRequiredEntry.**login_required**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class zinnia.models_bases.entry.**PasswordRequiredEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class to restrict the display of the entry to users knowing the password.

Parameters **password** (*CharField*) – Protects the entry with a password.

class Meta

abstract = False

PasswordRequiredEntry.**password**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class zinnia.models_bases.entry.**RelatedEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class for making manual relations between the differents entries.

class Meta

abstract = False

RelatedEntry.**related**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

RelatedEntry.**related_published**
Returns only related entries published.

class zinnia.models_bases.entry.**TagsEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract model class to add tags to the entries.

Parameters **tags** (*TagField*) – Tags

class Meta

abstract = False

TagsEntry.**tags_list**
Return iterable list of tags.

zinnia.models_bases.entry.**image_upload_to_dispatcher**(*entry*, *filename*)
Dispatch function to allow overriding of image_upload_to method.

Outside the model for fixing an issue with Django's migrations on Python 2.

spam_checker Package

spam_checker Package

Spam checker for Zinnia

zinnia.spam_checker.**check_is_spam**(*content*, *content_object*, *request*, *backends=None*)
Return True if the content is a spam, else False.

zinnia.spam_checker.**get_spam_checker**(*backend_path*)
Return the selected spam checker backend.

Subpackages

backends Package

backends Package

Spam checker backends for Zinnia

all_is_spam Module

All is spam, spam checker backend for Zinnia

zinnia.spam_checker.backends.all_is_spam.**backend**(**ka*)
Backend for setting all comments to spam.

long_enough Module

Long enough spam checker backend for Zinnia

zinnia.spam_checker.backends.long_enough.**backend**(*comment*, *content_object*, *request*)
Backend checking if the comment posted is long enough to be public. Generally a comments with few words is useless. The will avoid comments like this:

•First !

•I don't like.

•Check http://spam-ads.com/

url_shortener Package

url_shortener Package

URL shortener for Zinnia

zinnia.url_shortener.**get_url_shortener**()
Return the selected URL shortener backend.

Subpackages

backends Package

backends Package

URL shortener backends for Zinnia

default Module

Default URL shortener backend for Zinnia

zinnia.url_shortener.backends.default.**backend**(*entry*)
Default URL shortener backend for Zinnia.

zinnia.url_shortener.backends.default.**base36**(*value*)
Encode int to base 36.

urls Package

urls Package

Defaults urls for the Zinnia project

zinnia.urls.**i18n_url**(*url*, *translate=False*)
Translate or not an URL part.

authors Module

Urls for the Zinnia authors

archives Module

Urls for the Zinnia archives

`capabilities` Module

Urls for the zinnia capabilities

`categories` Module

Urls for the Zinnia categories

`comments` Module

Urls for the Zinnia comments

`entries` Module

Urls for the Zinnia entries

`feeds` Module

Urls for the Zinnia feeds

`quick_entry` Module

Url for the Zinnia quick entry view

`search` Module

Urls for the Zinnia search

`sitemap` Module

Urls for the Zinnia sitemap

`shortlink` Module

Urls for the Zinnia entries short link

`tags` Module

Urls for the Zinnia tags

`trackback` Module

Urls for the Zinnia trackback

views Package

views Package

Views for Zinnia

authors Module

Views for Zinnia authors

class zinnia.views.authors.**AuthorDetail**(***kwargs*)
Bases: *zinnia.views.mixins.templates.EntryQuerysetTemplateResponseMixin*, zinnia.views.mixins.prefetch_related.PrefetchCategoriesAuthorsMixin, *zinnia.views.authors.BaseAuthorDetail*, django.views.generic.list.BaseListView

Detailed view for an Author combinating these mixins:

- EntryQuerysetTemplateResponseMixin to provide custom templates for the author display page.
- PrefetchCategoriesAuthorsMixin to prefetch related Categories and Authors to belonging the entry list.
- BaseAuthorDetail to provide the behavior of the view.
- BaseListView to implement the ListView.

get_model_name()
The model name is the author's username.

model_type = 'author'

paginate_by = 10

class zinnia.views.authors.**AuthorList**(***kwargs*)
Bases: django.views.generic.list.ListView

View returning a list of all published authors.

get_queryset()
Return a queryset of published authors, with a count of their entries published.

class zinnia.views.authors.**BaseAuthorDetail**
Bases: object

Mixin providing the behavior of the author detail view, by returning in the context the current author and a queryset containing the entries written by author.

get_context_data(***kwargs*)
Add the current author in context.

get_queryset()
Retrieve the author by his username and build a queryset of his published entries.

archives Module

Views for Zinnia archives

class zinnia.views.archives.**EntryArchiveMixin**
Bases: *zinnia.views.mixins.archives.ArchiveMixin*, *zinnia.views.mixins.archives.PreviousNextPublishedMixin*, zinnia.views.mixins.prefetch_related.PrefetchCategoriesAuthorsMixin, *zinnia.views.mixins.callable_queryset.CallableQuerysetMixin*, *zinnia.views.mixins.templates.EntryQuerysetArchiveTemplateResponseMixin*

Mixin combinating:

- ArchiveMixin configuration centralizing conf for archive views.
- PrefetchCategoriesAuthorsMixin to prefetch related objects.
- PreviousNextPublishedMixin for returning published archives.
- CallableQueryMixin to force the update of the queryset.
- EntryQuerysetArchiveTemplateResponseMixin to provide a custom templates for archives.

queryset()

class zinnia.views.archives.**EntryDay**(***kwargs*)
Bases: *zinnia.views.archives.EntryArchiveMixin*, django.views.generic.dates.BaseDayArchiveView

View returning the archive for a day.

template_name_suffix = '_archive_day'

class zinnia.views.archives.**EntryIndex**(***kwargs*)
Bases: *zinnia.views.archives.EntryArchiveMixin*, *zinnia.views.mixins.templates.EntryQuerysetArchiveTodayTemplateResponseMixin*, django.views.generic.dates.BaseArchiveIndexView

View returning the archive index.

context_object_name = 'entry_list'

class zinnia.views.archives.**EntryMonth**(***kwargs*)
Bases: *zinnia.views.archives.EntryArchiveMixin*, django.views.generic.dates.BaseMonthArchiveView

View returning the archives for a month.

template_name_suffix = '_archive_month'

class zinnia.views.archives.**EntryToday**(***kwargs*)
Bases: *zinnia.views.archives.EntryArchiveMixin*, django.views.generic.dates.BaseTodayArchiveView

View returning the archive for the current day.

get_dated_items()
Return (date_list, items, extra_context) for this request. And defines self.year/month/day for EntryQuerysetArchiveTemplateResponseMixin.

template_name_suffix = '_archive_today'

class zinnia.views.archives.**EntryWeek**(***kwargs*)
Bases: *zinnia.views.archives.EntryArchiveMixin*, django.views.generic.dates.BaseWeekArchiveView

View returning the archive for a week.

get_dated_items()
Override get_dated_items to add a useful 'week_end_day' variable in the extra context of the view.

template_name_suffix = '_archive_week'

class zinnia.views.archives.**EntryYear**(**kwargs)
Bases: *zinnia.views.archives.EntryArchiveMixin*, django.views.generic.dates.BaseYearArchiveView

View returning the archives for a year.

make_object_list = True

template_name_suffix = '_archive_year'

categories Module

Views for Zinnia categories

class zinnia.views.categories.**BaseCategoryDetail**
Bases: object

Mixin providing the behavior of the category detail view, by returning in the context the current category and a queryset containing the entries published under it.

get_context_data(**kwargs)
Add the current category in context.

get_queryset()
Retrieve the category by his path and build a queryset of her published entries.

class zinnia.views.categories.**CategoryDetail**(**kwargs)
Bases: *zinnia.views.mixins.templates.EntryQuerysetTemplateResponseMixin*, zinnia.views.mixins.prefetch_related.PrefetchCategoriesAuthorsMixin, *zinnia.views.categories.BaseCategoryDetail*, django.views.generic.list.BaseListView

Detailed view for a Category combinating these mixins:

- EntryQuerysetTemplateResponseMixin to provide custom templates for the category display page.
- PrefetchCategoriesAuthorsMixin to prefetch related Categories and Authors to belonging the entry list.
- BaseCategoryDetail to provide the behavior of the view.
- BaseListView to implement the ListView.

get_model_name()
The model name is the category's slug.

model_type = 'category'

paginate_by = 10

class zinnia.views.categories.**CategoryList**(**kwargs)
Bases: django.views.generic.list.ListView

View returning a list of published categories.

get_queryset()
Return a queryset of published categories, with a count of their entries published.

zinnia.views.categories.**get_category_or_404**(*path*)
Retrieve a Category instance by a path.

capabilities Module

Views for Zinnia capabilities

class zinnia.views.capabilities.**CapabilityView**(**kwargs)
Bases: django.views.generic.base.TemplateView

Base view for the weblog capabilities.

get_context_data(**kwargs)
Populate the context of the template with technical informations for building urls.

class zinnia.views.capabilities.**HumansTxt**(**kwargs)
Bases: *zinnia.views.capabilities.CapabilityView*

http://humanstxt.org/

content_type = 'text/plain'

template_name = 'zinnia/humans.txt'

class zinnia.views.capabilities.**OpenSearchXml**(**kwargs)
Bases: *zinnia.views.capabilities.CapabilityView*

http://www.opensearch.org/

content_type = 'application/opensearchdescription+xml'

template_name = 'zinnia/opensearch.xml'

class zinnia.views.capabilities.**RsdXml**(**kwargs)
Bases: *zinnia.views.capabilities.CapabilityView*

http://en.wikipedia.org/wiki/Really_Simple_Discovery

content_type = 'application/rsd+xml'

template_name = 'zinnia/rsd.xml'

class zinnia.views.capabilities.**WLWManifestXml**(**kwargs)
Bases: *zinnia.views.capabilities.CapabilityView*

http://msdn.microsoft.com/en-us/library/bb463260.aspx

content_type = 'application/wlwmanifest+xml'

template_name = 'zinnia/wlwmanifest.xml'

channels Module

Views for Zinnia channels

class zinnia.views.channels.**BaseEntryChannel**
Bases: object

Mixin for displaying a custom selection of entries based on a search query, useful to build SEO/SMO pages aggregating entries on a thematic or for building a custom homepage.

get_context_data(**kwargs)
Add query in context.

get_queryset()
Override the get_queryset method to build the queryset with entry matching query.

query = ''

class zinnia.views.channels.**EntryChannel**(***kwargs*)
Bases: zinnia.views.mixins.prefetch_related.PrefetchCategoriesAuthorsMixin, *zinnia.views.channels.BaseEntryChannel*, django.views.generic.list.ListView

Channel view for entries combinating these mixins:

- PrefetchCategoriesAuthorsMixin to prefetch related Categories and Authors to belonging the entry list.
- BaseEntryChannel to provide the behavior of the view.
- ListView to implement the ListView and template name resolution.

paginate_by = 10

comments Module

Views for Zinnia comments

class zinnia.views.comments.**CommentSuccess**(***kwargs*)
Bases: django.views.generic.base.TemplateResponseMixin, django.views.generic.base.View

View for handing the publication of a Comment on an Entry. Do a redirection if the comment is visible, else render a confirmation template.

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

get_context_data(***kwargs*)

template_name = 'comments/zinnia/entry/posted.html'

entries Module

Views for Zinnia entries

class zinnia.views.entries.**EntryDateDetail**(***kwargs*)
Bases: *zinnia.views.mixins.archives.ArchiveMixin*, *zinnia.views.mixins.templates.EntryArchiveTemplateResponseMixin*, *zinnia.views.mixins.callable_queryset.CallableQuerysetMixin*, django.views.generic.dates.BaseDateDetailView

Mixin combinating:

- ArchiveMixin configuration centralizing conf for archive views
- EntryArchiveTemplateResponseMixin to provide a custom templates depending on the date
- BaseDateDetailView to retrieve the entry with date and slug
- CallableQueryMixin to defer the execution of the *queryset* property when imported

queryset()
Return entries published on current site.

class zinnia.views.entries.**EntryDetail**(***kwargs*)
Bases: *zinnia.views.mixins.entry_cache.EntryCacheMixin*, *zinnia.views.mixins.entry_preview.EntryPreviewMixin*, *zinnia.views.mixins.entry_protection.EntryProtectionMixin*, *zinnia.views.entries.EntryDateDetail*

Detailled archive view for an Entry with password and login protections and restricted preview.

quick_entry Module

Views for Zinnia quick entry

class zinnia.views.quick_entry.**QuickEntry**(***kwargs*)
Bases: django.views.generic.base.View

View handling the quick post of a short Entry.

dispatch(**args*, ***kwargs*)
Decorate the view dispatcher with permission_required.

get(*request*, **args*, ***kwargs*)
GET only do a redirection to the admin for adding and entry.

htmlize(*content*)
Convert to HTML the content if the MARKUP_LANGUAGE is set to HTML to optimize the rendering and avoid ugly effect in WYMEditor.

post(*request*, **args*, ***kwargs*)
Handle the datas for posting a quick entry, and redirect to the admin in case of error or to the entry's page in case of success.

class zinnia.views.quick_entry.**QuickEntryForm**(*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*, *instance=None*, *use_required_attribute=None*)
Bases: django.forms.models.ModelForm

Form for posting an entry quickly.

class Meta

exclude = ('comment_count', 'pingback_count', 'trackback_count')

model
alias of Entry

QuickEntryForm.**base_fields = OrderedDict([('title', <django.forms.fields.CharField object>), ('slug', <django.for**

QuickEntryForm.**declared_fields = OrderedDict()**

QuickEntryForm.**media**

search Module

Views for Zinnia entries search

class zinnia.views.search.**BaseEntrySearch**
Bases: object

Mixin providing the behavior of the entry search view, by returning in the context the pattern searched, the error if something wrong has happened and finally the the queryset of published entries matching the pattern.

error = None

get_context_data(***kwargs*)
Add error and pattern in context.

get_queryset()
Overridde the get_queryset method to do some validations and build the search queryset.

pattern = ''

class zinnia.views.search.**EntrySearch**(**kwargs)
Bases: zinnia.views.mixins.prefetch_related.PrefetchCategoriesAuthorsMixin, *zinnia.views.search.BaseEntrySearch*, django.views.generic.list.ListView

Search view for entries combinating these mixins:

- PrefetchCategoriesAuthorsMixin to prefetch related Categories and Authors to belonging the entry list.
- BaseEntrySearch to provide the behavior of the view.
- ListView to implement the ListView and template name resolution.

paginate_by = 10

template_name_suffix = '_search'

sitemap Module

Views for Zinnia sitemap

class zinnia.views.sitemap.**Sitemap**(**kwargs)
Bases: django.views.generic.base.TemplateView

Sitemap view of the Weblog.

get_context_data(**kwargs)
Populate the context of the template with all published entries and all the categories.

template_name = 'zinnia/sitemap.html'

shortlink Module

Views for Zinnia shortlink

class zinnia.views.shortlink.**EntryShortLink**(**kwargs)
Bases: django.views.generic.base.RedirectView

View for handling the shortlink of an Entry, simply do a redirection.

get_redirect_url(**kwargs)
Get entry corresponding to 'pk' encoded in base36 in the 'token' variable and return the get_absolute_url of the entry.

permanent = True

tags Module

Views for Zinnia tags

class zinnia.views.tags.**BaseTagDetail**
Bases: object

Mixin providing the behavior of the tag detail view, by returning in the context the current tag and a queryset containing the entries published with the tag.

get_context_data(***kwargs*)
Add the current tag in context.

get_queryset()
Retrieve the tag by his name and build a queryset of his published entries.

class zinnia.views.tags.**TagDetail**(***kwargs*)
Bases: *zinnia.views.mixins.templates.EntryQuerysetTemplateResponseMixin*, zinnia.views.mixins.prefetch_related.PrefetchCategoriesAuthorsMixin, *zinnia.views.tags.BaseTagDetail*, django.views.generic.list.BaseListView

Detailed view for a Tag combinating these mixins:

- EntryQuerysetTemplateResponseMixin to provide custom templates for the tag display page.
- PrefetchCategoriesAuthorsMixin to prefetch related Categories and Authors to belonging the entry list.
- BaseTagDetail to provide the behavior of the view.
- BaseListView to implement the ListView.

get_model_name()
The model name is the tag slugified.

model_type = 'tag'

paginate_by = 10

class zinnia.views.tags.**TagList**(***kwargs*)
Bases: django.views.generic.list.ListView

View return a list of all published tags.

context_object_name = 'tag_list'

get_queryset()
Return a queryset of published tags, with a count of their entries published.

template_name = 'zinnia/tag_list.html'

trackback Module

Views for Zinnia trackback

class zinnia.views.trackback.**EntryTrackback**(***kwargs*)
Bases: django.views.generic.base.TemplateView

View for handling trackbacks on the entries.

content_type = 'text/xml'

dispatch(**args*, ***kwargs*)
Decorate the view dispatcher with csrf_exempt.

get(*request*, **args*, ***kwargs*)
GET only do a permanent redirection to the Entry.

get_object()
Retrieve the Entry trackbacked.

post(*request*, **args*, ***kwargs*)
Check if an URL is provided and if trackbacks are enabled on the Entry. If so the URL is registered one time as a trackback.

template_name = 'zinnia/entry_trackback.xml'

Subpackages

mixins Package

mixins Package

Mixins for Zinnia views

archives Module

Mixins for Zinnia archive views

class zinnia.views.mixins.archives.**ArchiveMixin**
Bases: object

Mixin centralizing the configuration of the archives views.

allow_empty = True

allow_future = True

date_field = 'publication_date'

month_format = '%m'

paginate_by = 10

week_format = '%W'

class zinnia.views.mixins.archives.**PreviousNextPublishedMixin**
Bases: object

Mixin for correcting the previous/next context variable to return dates with published datas.

get_next_day(*date*)
Get the next day with published entries.

get_next_month(*date*)
Get the next month with published entries.

get_next_week(*date*)
Get the next week with published entries.

get_next_year(*date*)
Get the next year with published entries.

get_previous_day(*date*)
Get the previous day with published entries.

get_previous_month(*date*)
Get the previous month with published entries.

get_previous_next_published(*date*)
Returns a dict of the next and previous date periods with published entries.

get_previous_week(*date*)
Get the previous wek with published entries.

get_previous_year(*date*)
Get the previous year with published entries.

callable_queryset Module

Callable Queryset mixins for Zinnia views

class zinnia.views.mixins.callable_queryset.**CallableQuerysetMixin**
Bases: object

Mixin for handling a callable queryset, which will force the update of the queryset. Related to issue http://code.djangoproject.com/ticket/8378

get_queryset()
Check that the queryset is defined and call it.

queryset = None

entry_cache Module

Cache mixins for Zinnia views

class zinnia.views.mixins.entry_cache.**EntryCacheMixin**
Bases: object

Mixin implementing cache on get_object method.

get_object(*queryset=None*)
Implement cache on get_object method to avoid repetitive calls, in POST.

entry_preview Module

Preview mixins for Zinnia views

class zinnia.views.mixins.entry_preview.**EntryPreviewMixin**
Bases: object

Mixin implementing the preview of Entries.

get_object(*queryset=None*)
If the status of the entry is not PUBLISHED, a preview is requested, so we check if the user has the 'zinnia.can_view_all' permission or if it's an author of the entry.

entry_protection Module

Protection mixins for Zinnia views

class zinnia.views.mixins.entry_protection.**EntryProtectionMixin**
Bases: *zinnia.views.mixins.entry_protection.LoginMixin*, *zinnia.views.mixins.entry_protection.PasswordMixin*

Mixin returning a login view if the current entry need authentication and password view if the entry is protected by a password.

get(*request*, **args*, ***kwargs*)
Do the login and password protection.

post(*request*, **args*, ***kwargs*)
Do the login and password protection.

session_key = 'zinnia_entry_%s_password'

class zinnia.views.mixins.entry_protection.**LoginMixin**
Bases: object

Mixin implemeting a login view configurated for Zinnia.

login()
Return the login view.

class zinnia.views.mixins.entry_protection.**PasswordMixin**
Bases: object

Mixin implementing a password view configurated for Zinnia.

error = False

password()
Return the password view.

templates Module

Template mixins for Zinnia views

class zinnia.views.mixins.templates.**EntryArchiveTemplateResponseMixin**
Bases: *zinnia.views.mixins.templates.EntryQuerysetArchiveTemplateResponseMixin*

Same as EntryQuerysetArchiveTemplateResponseMixin but use the template defined in the Entry instance as the base template name.

get_default_base_template_names()
Return the Entry.template value.

class zinnia.views.mixins.templates.**EntryQuerysetArchiveTemplateResponseMixin**
Bases: django.views.generic.base.TemplateResponseMixin

Return a custom template name for the archive views based on the type of the archives and the value of the date.

get_archive_part_value(*part*)
Method for accessing to the value of self.get_year(), self.get_month(), etc methods if they exists.

get_default_base_template_names()
Return a list of default base templates used to build the full list of templates.

get_template_names()
Return a list of template names to be used for the view.

template_name_suffix = '_archive'

class zinnia.views.mixins.templates.**EntryQuerysetArchiveTodayTemplateResponseMixin**
Bases: *zinnia.views.mixins.templates.EntryQuerysetArchiveTemplateResponseMixin*

Same as EntryQuerysetArchiveTemplateResponseMixin but use the current date of the day when getting archive part values.

get_archive_part_value(*part*)
Return archive part for today

today = None

class zinnia.views.mixins.templates.**EntryQuerysetTemplateResponseMixin**
Bases: django.views.generic.base.TemplateResponseMixin

Return a custom template name for views returning a queryset of Entry filtered by another model.

get_model_name()
Return the model name for templates.

get_model_type()
Return the model type for templates.

get_template_names()
Return a list of template names to be used for the view.

model_name = None

model_type = None

xmlrpc Package

xmlrpc Package

XML-RPC methods for Zinnia

metaweblog Module

XML-RPC methods of Zinnia metaWeblog API

zinnia.xmlrpc.metaweblog.**authenticate**(*username*, *password*, *permission=None*)
Authenticate staff_user with permission.

zinnia.xmlrpc.metaweblog.**author_structure**(*user*)
An author structure.

zinnia.xmlrpc.metaweblog.**blog_structure**(*site*)
A blog structure.

zinnia.xmlrpc.metaweblog.**category_structure**(*category*, *site*)
A category structure.

zinnia.xmlrpc.metaweblog.**delete_post**(*apikey*, *post_id*, *username*, *password*, *publish*)
blogger.deletePost(api_key, post_id, username, password, 'publish') => boolean

zinnia.xmlrpc.metaweblog.**edit_post**(*post_id*, *username*, *password*, *post*, *publish*)
metaWeblog.editPost(post_id, username, password, post, publish) => boolean

zinnia.xmlrpc.metaweblog.**get_authors**(*apikey*, *username*, *password*)
wp.getAuthors(api_key, username, password) => author structure[]

zinnia.xmlrpc.metaweblog.**get_categories**(*blog_id*, *username*, *password*)
metaWeblog.getCategories(blog_id, username, password) => category structure[]

zinnia.xmlrpc.metaweblog.**get_post**(*post_id*, *username*, *password*)
metaWeblog.getPost(post_id, username, password) => post structure

zinnia.xmlrpc.metaweblog.**get_recent_posts**(*blog_id*, *username*, *password*, *number*)
metaWeblog.getRecentPosts(blog_id, username, password, number) => post structure[]

zinnia.xmlrpc.metaweblog.**get_tags**(*blog_id*, *username*, *password*)
wp.getTags(blog_id, username, password) => tag structure[]

zinnia.xmlrpc.metaweblog.**get_user_info**(*apikey*, *username*, *password*)
blogger.getUserInfo(api_key, username, password) => user structure

zinnia.xmlrpc.metaweblog.**get_users_blogs**(*apikey*, *username*, *password*)
blogger.getUsersBlogs(api_key, username, password) => blog structure[]

zinnia.xmlrpc.metaweblog.**new_category**(*blog_id*, *username*, *password*, *category_struct*)
wp.newCategory(blog_id, username, password, category) => category_id

zinnia.xmlrpc.metaweblog.**new_media_object**(*blog_id*, *username*, *password*, *media*)
metaWeblog.newMediaObject(blog_id, username, password, media) => media structure

zinnia.xmlrpc.metaweblog.**new_post**(*blog_id*, *username*, *password*, *post*, *publish*)
metaWeblog.newPost(blog_id, username, password, post, publish) => post_id

zinnia.xmlrpc.metaweblog.**post_structure**(*entry*, *site*)
A post structure with extensions.

zinnia.xmlrpc.metaweblog.**tag_structure**(*tag*, *site*)
A tag structure.

zinnia.xmlrpc.metaweblog.**user_structure**(*user*, *site*)
An user structure.

pingback Module

XML-RPC methods of Zinnia Pingback

class zinnia.xmlrpc.pingback.**FakeRequest**
Bases: object

META = {}

zinnia.xmlrpc.pingback.**generate_pingback_content**(*soup*, *target*, *max_length*, *trunc_char='...'*)
Generate a description text for the pingback.

zinnia.xmlrpc.pingback.**pingback_extensions_get_pingbacks**(*target*)
pingback.extensions.getPingbacks(url) => '[url, url, ...]'

Returns an array of URLs that link to the specified url.

See: http://www.aquarionics.com/misc/archives/blogite/0198.html

zinnia.xmlrpc.pingback.**pingback_ping**(*source*, *target*)
pingback.ping(sourceURI, targetURI) => 'Pingback message'

Notifies the server that a link has been added to sourceURI, pointing to targetURI.

See: http://hixie.ch/specs/pingback/pingback-1.0

CHAPTER 7

Notes

Frequently Asked Questions

Contents

- *Frequently Asked Questions*
  - *Entries*
    - *I want to write my entries in MarkDown, RestructuredText or any lightweight markup language, is it possible ?*
    - *I want to have multilingual support on the entries, is it possible ?*
    - *Is Zinnia able to allow multiple users to edit it's own blog ?*
  - *Images*
    - *How can I use the image field for fitting to my skin ?*
    - *I want an image gallery in my posts, what can I do ?*
  - *Comments*
    - *Is it possible have a different comment system, with reply feature for example ?*

Entries

I want to write my entries in MarkDown, RestructuredText or any lightweight markup language, is it possible ?

Yes of course, Zinnia currently support MarkDown, Textile and reStructuredText as markup languages, but if you want to write your entries in a custom markup language a solution is to disable the WYSIWYG editor in the admin site with the `ZINNIA_WYSIWYG` setting, and use the appropriate template filter in your templates.

I want to have multilingual support on the entries, is it possible ?

Due to the extending capabilities of Zinnia, many solutions on this problematic are possible, but you must keep in mind that multiplingual entries is just a concept, the needs and the implementations can differ from a project to another. But you should take a look on this excellent tutorial to convert Zinnia into a multilingual Weblog with django-modeltranslation, which can be a good starting point for your needs.

Is Zinnia able to allow multiple users to edit it's own blog ?

Zinnia is designed to be multi-site. That's mean you can publish entries on several sites or share an admin interface for all the sites handled.

Zinnia also provides a new permission that's allow or not the user to change the authors. Useful for collaborative works.

But if you want to restrict the edition of the entries by site, authors or whatever you want, it's your job to implement this functionality in your project.

The simple way to do that, respecting the Django rules, is to override the admin classes provided by Zinnia, and register those classes in another admin site.

Images

How can I use the image field for fitting to my skin ?

Take a looks at sorl.thumbnail and use his templatetags.

You can do something like this in your templates:

```
<img src="{% thumbnail object.image 250x250 %}" />
```

I want an image gallery in my posts, what can I do ?

Simply create a new application with a model named `EntryImage` with a `ForeignKey` to the *`Entry`* model.

Then in the admin module of your app, unregister the *`EntryAdmin`* class, and use `InlineModelAdmin` in your new admin class.

Here an simple example :

```
# The model
from django.db import models
from django.utils.translation import ugettext_lazy as _

from zinnia.models.entry import Entry

class EntryImage(models.Model):
    """Image Model"""
    entry = models.ForeignKey(Entry, verbose_name=_('entry'))

    image = models.ImageField(_('image'), upload_to='uploads/gallery')
    title = models.CharField(_('title'), max_length=250)
    description = models.TextField(_('description'), blank=True)

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

```
# The admin

from django.contrib import admin

from zinnia.admin import EntryAdmin
from zinnia.models.entry import Entry
from gallery.models import EntryImage

class EntryImageInline(admin.TabularInline):
    model = EntryImage

class EntryAdminImage(EntryAdmin):
    inlines = (EntryImageInline,)

admin.site.unregister(Entry)
admin.site.register(Entry, EntryAdminImage)
```

Another and better solution is to extend the *Entry* model like described in *Extending Entry model*.

Comments

Is it possible have a different comment system, with reply feature for example ?

Yes the comment system integrated in Zinnia is based on `django_comments` and can be extended or replaced if doesn't quite fit your needs. You should take a look on the customizing the comments framework documentation for more information.

Warning: The custom comment Model must be inherited from `Comment` and implement the `CommentManager` to properly work with Zinnia.

If you want the ability to reply on comments, you can take a look at zinnia-threaded-comments or at django-threadcomments.

Compatibility

Zinnia tries to fit a maximum to the Django's standards to gain in readability and to be always present when the version 3.4.2 of Django will be here. :)

Predicting the future is a good thing, because it's coming soon. Actually Zinnia is designed to handle the 1.10.x version and will reach the release 1.12 easily without major changes.

https://docs.djangoproject.com/en/dev/internals/deprecation/

But the evolution of Django causes some backward incompatible changes, so for the developers who have to maintain a project with an old version of Django, it can be difficult to find which version of Zinnia to choose.

Compatibility with Django

Here a list establishing the compatibility between Zinnia and Django:

Changed in version 0.18.

Backward incompatibilities with Django v1.9.x due to :

- Removal of allow_tags property in django.contrib.admin
- Changes in prototype of widget.render_options
- Use of the new property user.is_authenticated

Changed in version 0.17.

Backward incompatibilities with Django v1.8.x due to :

- Usage of Field.remote_field.
- Usage of the new template tag syntax.
- Changes around the application namespace.

Changed in version 0.16.

Backward incompatibilities with Django v1.7.x due to :

- Usage of the new TEMPLATES API.
- Remove of templates tags loaded from future.

Changed in version 0.15.

Backward incompatibilities with Django v1.6.x due to :

- Usage of the new migrations.
- Usage of the new lru_cache function.
- Usage of Admin.get_changeform_initial_data method.

Changed in version 0.14.

Backward incompatibilities with Django v1.5.x due to :

- Usage of Queryset.datetimes().
- Handle savepoints in tests.

Changed in version 0.13.

Backward incompatibilities with Django v1.4.x due to :

- Experimental support of Python 3.
- Remove of the Python 2.5 support.
- Changes related to the archives views.
- Usage of the new syntax for the url templatetag.

Changed in version 0.11.

Backward incompatibilities with Django v1.3.x due to :

- Time-zones support.
- Usage of the new features provided in the testrunner.

Changed in version 0.10.

Backward incompatibilities with Django v1.2.x due to :

- Migration to the class-based generic views.
- Intensive usage of `django.contrib.staticfiles`.

- Usage of the new features provided in the testrunner.

Changed in version 0.6.

Backward incompatibilities with Django v1.1.x due to :

- Migration of the feeds classes of `django.contrib.syndication`.

Changed in version 0.5.

Backward incompatibilities with Django v1.0.x due to :

- Intensive usage of the actions in `django.contrib.admin`.

Thanks

Zinnia cannot be a great application without great contributors who make this application greatest each day.

- Julien Fache <fantomas42@gmail.com>
- Mark Young <marky1991@gmail.com>
- Nick Jones <nick@itsnotworking.co.uk>
- Jose Ignacio Galarza <igalarzab@gmail.com>
- noobidoo <ttraskback@gmail.com>
- Mauro Bianchi <bianchimro@gmail.com>
- Tobias von Klipstein <tobias@uxebu.com>
- Edwin Nimell <edwin@nimell.com>
- Dan <ellisd23@gmail.com>
- alyoung <ak@ondv.ru>
- Ales Zabala Alava (Shagi) <shagi@gisa-elkartea.org>
- Aaron C. Spike <aaron@ekips.org>
- unknown <collin@iwanttocode.com>
- Totoro <qinjiannet@sina.com>
- smcoll <smcoll@gmail.com>
- Shannon Collins <smcoll@gmail.com>
- Jervis Whitley <jervis.whitley+git@gmail.com>
- Jens Heidrich <jens@jensheidrich.de>
- Bjorn Meyer <bjorn.m.meyer@gmail.com>
- benjaoming <benjaoming@gmail.com>
- Alexis Tabary <alexis.tabary@gmail.com>
- Viktor Nagy <viktor.nagy@gmail.com>
- Travis Jensen <travis@innerbrane.com>
- sandino <bdzhon@gmail.com>
- Peter Bittner <django@bittner.it>

- Pedro Ferreira <plferreira4@gmail.com>
- nl_0 <nl_0@mail.ru>
- mohammad taleb <simul14@gmail.com>
- Mike C. Fletcher <mcfletch@vrplumber.com>
- Matthew Tretter <matthew@exanimo.com>
- Matěj Cepl <mcepl@redhat.com>
- Mark Renron <indexofire@gmail.com>
- maccesch <maccesch@web.de>
- KyleSomogyi <kyle.somogyi@bawtreesoftware.com>
- kpengboy <kkpengboy@gmail.com>
- Justin Turner Arthur <justinarthur@gmail.com>
- Jonathan Stoppani <garetjax@macbook.local>
- Jason Davies <jason@jasondavies.com>
- Jannis Leidel <jannis@leidel.info>
- Habibutsu <habibutsu@gmail.com>
- Gvidon Malyarov <gvidon@ottofeller.com>
- Grzegorz Biały <grzegorz@grzegorz.(none)>
- gpciceri <gp.ciceri@gmail.com>
- Frank Bieniek <frank.bieniek@produktlaunch.de>
- Esau Rodriguez <esau.rodriguez@galotecnia.com>
- Daniel Jost <torpedojost@gmail.com>
- Chuzz <chuzz.chuzz@gmail.com>
- bill <bill.mill@gmail.com>
- Bernhard Vallant <bernhard@vallant.org>
- Benoit Grégoire <benoitg@coeus.ca>
- Andrew Cutler <macropin@gmail.com>
- Andreas Damgaard Pedersen <andreas.damgaard.pedersen@gmail.com>
- Alexander Clausen <alex@gc-web.de>
- adieu <adieu@adieu.me>

And also a special thank to GitHub.com, Transifex.net, Travis-CI, Coveralls, and ReadTheDocs.org for their services of great quality.

CHANGELOG

0.18.1

- Fix messed relationships in Python 3

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.18...v0.18.1

0.18

- Compatibility with Django 1.10

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.17...v0.18

0.17

- Compatibility with Django 1.9
- Fix RSS enclosure
- Fix paginator issue
- Implement Entry.lead_html method
- Usage of regex module to speed up

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.16...v0.17

0.16

- Improve testing
- Improve documentation
- Reduce queries on `entry_detail_view`
- Implement custom templates within a loop
- Add a publication_date field to remove ambiguosity
- Remove WXR template
- Remove usage of `Context`
- Remove BeautifulSoup warnings

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.15.2...v0.16

0.15.2

- Covering of the code at 100%
- Compatibility with Django 1.8
- Improvements on the Calendar
- Improvements on `CategoryFeed`
- Improvements on `Entry.save()`
- Improvements on ping of external URLs
- Improvements on the comparison module
- Improvements on the translation strings
- The default theme is now in HTML5

- The default theme supports Microdatas
- Add a `lead` field on the Entry model
- Add an `image_caption` field on the Entry model
- Fix preview of entries when not yet published
- Fix migrations when using custom `User` model
- More generic parameter for MarkDown extensions
- Import utilies were moved to their own package
- The fields where the search is done are now configurable

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.15.1...v0.15.2

0.15.1

- Documentation improvements
- Fix migration issues with Django 1.7

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.15...v0.15.1

0.15

- Django 1.6 is no longer supported.

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.14.3...v0.15

0.14.3

- Improvement on the default theme * RSS links * Better translations * Correct title markup * Fix anchors for linkbacks
- Reorder the provided statics
- Rename `zinnia_tags` to `zinnia`
- Fix calendar in archive day view
- Fix Textitle rendering on Python 3
- Fix feeds for authors with accents
- Fix admin issue with custom Entry model
- Do not include anymore jQuery in admin for entries
- Admin tag autocompletion with a widget based on select2
- Configurable upload path for image field with inheritance

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.14.2...v0.14.3

0.14.2

- Optimize sitemap page
- Smarter widont filter
- Fix issue on pagination
- Fix several admin issues
- Fix short link for unpublished entries
- Integration with Gulp.js
- HTML and CSS fixes on default theme
- Tested under PostGres and MySQL and SQLite
- URLs are now under the `zinnia` namespace
- Move Twitter support to `zinnia-twitter`
- Move Mollom support to `zinnia-spam-checker-mollom`
- Move Akismet support to `zinnia-spam-checker-akismet`
- Move Bit.ly support to `zinnia-url-shortener-bitly`
- Move TinyMCE support to `zinnia-wysiwyg-tinymce`
- Move MarkItUp support to `zinnia-wysiwyg-markitup`
- Move WYMEditor support to `zinnia-wysiwyg-wymeditor`
- Use `django_comments` instead of `django.contrib.comments`

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.14.1...v0.14.2

0.14.1

- Fix dates on WXR export
- Fix blogger2zinnia unicode issue
- Fix unicode issue on Category admin
- Fix URL errors with custom comment app
- Full support of custom User model
- Metrics for the content previews
- More useable pagination
- More blocks for customizing reactions
- Minor documentation updates
- Minor fixes for default skin
- Review admin form for editing the entries
- Restricted preview for unpublished entries

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.14...v0.14.1

0.14

- Full Python 3.0 support
- Django 1.5 is no longer supported
- Better support of custom User model
- Improvements on the archives by week
- Fix timezone issues in templatetags and archives
- Database query optimizations in the archives views

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.13...v0.14

0.13

- Start Python 3.0 support
- Display page number in list
- Basic support of custom User
- Django 1.4 is no longer supported

https://github.com/Fantomas42/django-blog-zinnia/compare/v0.12.3...v0.13

0.12.3

- Better `skeleton.html`
- Better rendering for the slider
- Add view for having a random entry
- Compatibility fix with Django 1.5 in admin
- Fix issue with author detail view paginated
- Better settings for `ZINNIA_AUTO_CLOSE_*_AFTER`

0.12.2

- CSS updates and fixes
- Fix viewport meta tag
- I18n support for the URLs
- Update MarkItUp to v1.1.13
- Update WYMeditor to v1.0.0b3
- Entry's content can be blank
- Compatibility fix for WXR > 1.0
- Fix potential issue on `check_is_spam`

0.12.1

- Microformats improved
- Improve Blogger importer
- Finest control on linkbacks
- Split Entry model into mixins
- Compatibility fix with Django 1.5
- Custom template for content rendering
- Fix Python 2.7 issues with `wp2zinnia`

0.12

- Optimizations on the templates
- Optimizations on the database queries
- Denormalization of the comments
- `get_authors` context improved
- `get_tag_cloud` context improved
- `get_categories` context improved
- Default theme declinations
- Default theme more responsive
- Updating `helloworld.json` fixture
- Fix issues with authors in `wp2zinnia`
- Better integration of the comments system
- Models has been splitted into differents modules

0.11.2

- New admin filter for authors
- Minor translation improvements
- Minor documentation improvements
- `wp2zinnia` handle wxr version 1.2
- Customizations of the templates with ease
- Define a custom `Author.__unicode__` method
- Fix issue with duplicate spam comments
- Fix bug in `PreviousNextPublishedMixin`
- Fix bug in `QuickEntry` with non ascii title
- Fix `collectstatic` with `CachedStaticFilesStorage`

0.11.1

- Fix issues with `get_absolute_url` and `zbreadcrumbs` when time-zone support is enabled.

0.11

- Class-based views
- Time zones support
- Pagination on archives
- Better archive by week view
- Update of the breadcrumbs tag
- Improving `wp2zinnia` command
- New `long_enough` spam checker
- Custom templates for archive views
- Publication dates become unrequired
- No runtime warnings on Django 1.4
- Django 1.3 is no longer supported
- And a lot of bug fixes

0.10.1

- Django 1.4 compatibility support
- Compatibility with django-mptt >= 5.1
- `zinnia.plugins` is now removed

0.10

- Better default templates
- CSS refactoring with Sass3
- Statistics about the content
- Improvement of the documentation
- Entry's Meta options can be extended
- Django 1.2 is no longer supported
- `zinnia.plugins` is deprecated in favor of `cmsplugin_zinnia`
- And a lot of bug fixes

0.9

- Improved URL shortening
- Improved moderation system
- Better support of django-tagging
- Blogger to Zinnia utility command
- OpenSearch capabilities
- Upgraded search engine
- Feed to Zinnia utility command
- And a lot of bug fixes

0.8

- Admin dashboard
- Featured entries
- Using Microformats
- Mails for comment reply
- Entry model can be extended
- More plugins for django-cms
- Zinnia to Wordpress utility command
- Code cleaning and optimizations
- And a lot of bug fixes

0.7

- Using signals
- Trackback support
- Ping external URLs
- Private posts
- Hierarchical categories
- TinyMCE integration
- Code optimizations
- And a lot of bug fixes

0.6

- Handling PingBacks
- Support MetaWeblog API
- Passing to Django 1.2.x

- Breadcrumbs templatetag
- Bug correction in calendar widget
- Wordpress to Zinnia utility command
- Major bug correction on publication system
- And a lot of bug fixes

0.5

- Packaging
- Tests added
- Translations
- Better templates
- New templatetags
- Plugins for django-cms
- Twitter and Bit.ly support
- Publishing sources on Github.com

0.4 and before

- The previous versions of Zinnia were not packaged, and were destinated for a personnal use.

CHAPTER 8

Related

- Build status
- Coverage report

CHAPTER 9

Indices and tables

If you can't find the information you're looking for, have a look at the index or try to find it using the search function:

- genindex
- search

Python Module Index

Z

zinnia, 7
zinnia.__init__, 54
zinnia.admin, 68
zinnia.admin.category, 68
zinnia.admin.entry, 68
zinnia.admin.fields, 70
zinnia.admin.filters, 70
zinnia.admin.forms, 71
zinnia.admin.widgets, 72
zinnia.apps, 54
zinnia.breadcrumbs, 55
zinnia.calendar, 55
zinnia.comparison, 56
zinnia.context, 57
zinnia.context_processors, 57
zinnia.feeds, 57
zinnia.flags, 62
zinnia.managers, 62
zinnia.markups, 63
zinnia.models, 72
zinnia.models.author, 77
zinnia.models.category, 78
zinnia.models.entry, 19
zinnia.models_bases, 82
zinnia.models_bases.entry, 83
zinnia.moderator, 63
zinnia.ping, 64
zinnia.preview, 64
zinnia.search, 16
zinnia.settings, 41
zinnia.signals, 65
zinnia.sitemaps, 66
zinnia.spam_checker, 18
zinnia.spam_checker.backends, 92
zinnia.spam_checker.backends.all_is_spam, 92
zinnia.spam_checker.backends.long_enough, 92
zinnia.templates, 67
zinnia.templatetags, 47
zinnia.templatetags.zinnia, 47
zinnia.tests, 39
zinnia.url_shortener, 17
zinnia.url_shortener.backends, 93
zinnia.url_shortener.backends.default, 93
zinnia.urls, 93
zinnia.urls.archives, 93
zinnia.urls.authors, 93
zinnia.urls.capabilities, 94
zinnia.urls.categories, 94
zinnia.urls.comments, 94
zinnia.urls.entries, 94
zinnia.urls.feeds, 94
zinnia.urls.quick_entry, 94
zinnia.urls.search, 94
zinnia.urls.shortlink, 94
zinnia.urls.sitemap, 94
zinnia.urls.tags, 94
zinnia.urls.trackback, 94
zinnia.views, 95
zinnia.views.archives, 95
zinnia.views.authors, 95
zinnia.views.capabilities, 98
zinnia.views.categories, 97
zinnia.views.channels, 15
zinnia.views.comments, 99
zinnia.views.entries, 99
zinnia.views.mixins, 103
zinnia.views.mixins.archives, 103
zinnia.views.mixins.callable_queryset, 104
zinnia.views.mixins.entry_cache, 104
zinnia.views.mixins.entry_preview, 104
zinnia.views.mixins.entry_protection, 104
zinnia.views.mixins.templates, 105
zinnia.views.quick_entry, 100

zinnia.views.search, 15
zinnia.views.shortlink, 101
zinnia.views.sitemap, 101
zinnia.views.tags, 101
zinnia.views.trackback, 102
zinnia.xmlrpc, 106
zinnia.xmlrpc.metaweblog, 106
zinnia.xmlrpc.pingback, 107

Index

A

abstract (zinnia.models.author.AuthorPublishedManager.Meta attribute), 78
abstract (zinnia.models_bases.entry.AbstractEntry.Meta attribute), 84
abstract (zinnia.models_bases.entry.AuthorsEntry.Meta attribute), 85
abstract (zinnia.models_bases.entry.CategoriesEntry.Meta attribute), 85
abstract (zinnia.models_bases.entry.ContentEntry.Meta attribute), 86
abstract (zinnia.models_bases.entry.ContentTemplateEntry.Meta attribute), 86
abstract (zinnia.models_bases.entry.CoreEntry.Meta attribute), 86
abstract (zinnia.models_bases.entry.DetailTemplateEntry.Meta attribute), 88
abstract (zinnia.models_bases.entry.DiscussionsEntry.Meta attribute), 89
abstract (zinnia.models_bases.entry.ExcerptEntry.Meta attribute), 90
abstract (zinnia.models_bases.entry.FeaturedEntry.Meta attribute), 90
abstract (zinnia.models_bases.entry.ImageEntry.Meta attribute), 90
abstract (zinnia.models_bases.entry.LeadEntry.Meta attribute), 90
abstract (zinnia.models_bases.entry.LoginRequiredEntry.Meta attribute), 91
abstract (zinnia.models_bases.entry.PasswordRequiredEntry.Meta attribute), 91
abstract (zinnia.models_bases.entry.RelatedEntry.Meta attribute), 91
abstract (zinnia.models_bases.entry.TagsEntry.Meta attribute), 92
AbstractEntry (class in zinnia.models_bases.entry), 83
AbstractEntry.Meta (class in zinnia.models_bases.entry), 84
actions (zinnia.admin.entry.EntryAdmin attribute), 68
actions_on_bottom (zinnia.admin.entry.EntryAdmin attribute), 68
actions_on_top (zinnia.admin.entry.EntryAdmin attribute), 68
admin_site (zinnia.admin.forms.CategoryAdminForm attribute), 71
admin_site (zinnia.admin.forms.EntryAdminForm attribute), 71
advanced_search() (in module zinnia.search), 65
advanced_search() (zinnia.managers.EntryPublishedManager method), 62
allow_empty (zinnia.views.mixins.archives.ArchiveMixin attribute), 103
allow_future (zinnia.views.mixins.archives.ArchiveMixin attribute), 103
append_position() (in module zinnia.templates), 67
ArchiveMixin (class in zinnia.views.mixins.archives), 103
authenticate() (in module zinnia.xmlrpc.metaweblog), 106
Author (class in zinnia.models), 74
Author (class in zinnia.models.author), 77
Author.DoesNotExist, 75, 78
Author.MultipleObjectsReturned, 75, 78
author_structure() (in module zinnia.xmlrpc.metaweblog), 106
AuthorDetail (class in zinnia.views.authors), 95
AuthorEntries (class in zinnia.feeds), 57
AuthorList (class in zinnia.views.authors), 95
AuthorListFilter (class in zinnia.admin.filters), 70
AuthorPublishedManager (class in zinnia.models.author), 78
AuthorPublishedManager.Meta (class in zinnia.models.author), 78
authors (zinnia.models.Entry attribute), 73
authors (zinnia.models.entry.Entry attribute), 81
authors (zinnia.models_bases.entry.AbstractEntry attribute), 84
authors (zinnia.models_bases.entry.AuthorsEntry at-

tribute), 85
AuthorsEntry (class in zinnia.models_bases.entry), 85
AuthorsEntry.Meta (class in zinnia.models_bases.entry), 85
AuthorSitemap (class in zinnia.sitemaps), 66
auto_close_field (zinnia.moderator.EntryCommentModerator attribute), 63
auto_moderate_comments (zinnia.moderator.EntryCommentModerator attribute), 63

B

backend() (in module zinnia.spam_checker.backends.all_is_spam), 92
backend() (in module zinnia.spam_checker.backends.long_enough), 92
backend() (in module zinnia.url_shortener.backends.default), 93
base36() (in module zinnia.url_shortener.backends.default), 93
base_fields (zinnia.admin.forms.CategoryAdminForm attribute), 71
base_fields (zinnia.admin.forms.EntryAdminForm attribute), 71
base_fields (zinnia.views.quick_entry.QuickEntryForm attribute), 100
BaseAuthorDetail (class in zinnia.views.authors), 95
BaseCategoryDetail (class in zinnia.views.categories), 97
BaseEntryChannel (class in zinnia.views.channels), 98
BaseEntrySearch (class in zinnia.views.search), 100
BaseTagDetail (class in zinnia.views.tags), 101
basic_search() (zinnia.managers.EntryPublishedManager method), 62
blog_structure() (in module zinnia.xmlrpc.metaweblog), 106
build_preview() (zinnia.preview.HTMLPreview method), 64

C

cache (zinnia.comparison.CachedModelVectorBuilder attribute), 56
cache_backend (zinnia.comparison.CachedModelVectorBuilder attribute), 56
cache_flush() (zinnia.comparison.CachedModelVectorBuilder method), 56
cache_infos() (zinnia.sitemaps.EntryRelatedSitemap method), 66
cache_infos() (zinnia.sitemaps.TagSitemap method), 67
cache_key (zinnia.comparison.CachedModelVectorBuilder attribute), 56
cache_key (zinnia.comparison.EntryPublishedVectorBuilder attribute), 56
CachedModelVectorBuilder (class in zinnia.comparison), 56
Calendar (class in zinnia.calendar), 55
CallableQuerysetMixin (class in zinnia.views.mixins.callable_queryset), 104
CapabilityView (class in zinnia.views.capabilities), 98
categories (zinnia.models.Entry attribute), 73
categories (zinnia.models.entry.Entry attribute), 81
categories (zinnia.models_bases.entry.AbstractEntry attribute), 84
categories (zinnia.models_bases.entry.CategoriesEntry attribute), 85
CategoriesEntry (class in zinnia.models_bases.entry), 85
CategoriesEntry.Meta (class in zinnia.models_bases.entry), 85
Category (class in zinnia.models), 75
Category (class in zinnia.models.category), 78
Category.DoesNotExist, 76, 79
Category.MultipleObjectsReturned, 76, 79
category_structure() (in module zinnia.xmlrpc.metaweblog), 106
CategoryAdmin (class in zinnia.admin.category), 68
CategoryAdminForm (class in zinnia.admin.forms), 71
CategoryAdminForm.Meta (class in zinnia.admin.forms), 71
CategoryDetail (class in zinnia.views.categories), 97
CategoryEntries (class in zinnia.feeds), 58
CategoryList (class in zinnia.views.categories), 97
CategoryListFilter (class in zinnia.admin.filters), 70
CategorySitemap (class in zinnia.sitemaps), 66
changefreq (zinnia.sitemaps.EntryRelatedSitemap attribute), 66
changefreq (zinnia.sitemaps.EntrySitemap attribute), 67
check_is_spam() (in module zinnia.spam_checker), 92
children (zinnia.models.Category attribute), 76
children (zinnia.models.category.Category attribute), 79
choice() (zinnia.admin.fields.MPTTModelChoiceIterator method), 70
choices (zinnia.admin.fields.MPTTModelMultipleChoiceField attribute), 70
clean_parent() (zinnia.admin.forms.CategoryAdminForm method), 71
close_after (zinnia.moderator.EntryCommentModerator attribute), 63
close_comments() (zinnia.admin.entry.EntryAdmin method), 68
close_pingbacks() (zinnia.admin.entry.EntryAdmin method), 68
close_trackbacks() (zinnia.admin.entry.EntryAdmin method), 68
columns (zinnia.comparison.ModelVectorBuilder attribute), 56
columns_dataset (zinnia.comparison.CachedModelVectorBuilder attribute), 56

columns_dataset (zinnia.comparison.ModelVectorBuilder attribute), 56
comment_admin_urlname template filter, 53
comment_admin_urlname() (in module zinnia.templatetags.zinnia), 53
comment_count (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
comment_enabled (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
comments (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
comments_are_open (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
CommentSuccess (class in zinnia.views.comments), 99
compute_related() (zinnia.comparison.ModelVectorBuilder method), 56
connect_discussion_signals() (in module zinnia.signals), 65
connect_entry_signals() (in module zinnia.signals), 65
content (zinnia.models_bases.entry.ContentEntry attribute), 86
content_template (zinnia.models_bases.entry.ContentTemplateEntry attribute), 86
content_type (zinnia.views.capabilities.HumansTxt attribute), 98
content_type (zinnia.views.capabilities.OpenSearchXml attribute), 98
content_type (zinnia.views.capabilities.RsdXml attribute), 98
content_type (zinnia.views.capabilities.WLWManifestXml attribute), 98
content_type (zinnia.views.trackback.EntryTrackback attribute), 102
ContentEntry (class in zinnia.models_bases.entry), 85
ContentEntry.Meta (class in zinnia.models_bases.entry), 86
ContentTemplateEntry (class in zinnia.models_bases.entry), 86
ContentTemplateEntry.Meta (class in zinnia.models_bases.entry), 86
context_object_name (zinnia.views.archives.EntryIndex attribute), 96
context_object_name (zinnia.views.tags.TagList attribute), 102
CoreEntry (class in zinnia.models_bases.entry), 86
CoreEntry.Meta (class in zinnia.models_bases.entry), 86
count_comments_handler() (in module zinnia.signals), 65
count_discussions_handler() (in module zinnia.signals), 65
count_pingbacks_handler() (in module zinnia.signals), 65
count_trackbacks_handler() (in module zinnia.signals), 65
create_q() (in module zinnia.search), 65
creation_date (zinnia.models_bases.entry.CoreEntry attribute), 87
Crumb (class in zinnia.breadcrumbs), 55

D

dataset (zinnia.comparison.ModelVectorBuilder attribute), 57
date_field (zinnia.views.mixins.archives.ArchiveMixin attribute), 103
date_hierarchy (zinnia.admin.entry.EntryAdmin attribute), 68
day_crumb() (in module zinnia.breadcrumbs), 55
declared_fields (zinnia.admin.forms.CategoryAdminForm attribute), 71
declared_fields (zinnia.admin.forms.EntryAdminForm attribute), 71
declared_fields (zinnia.views.quick_entry.QuickEntryForm attribute), 100
delete_post() (in module zinnia.xmlrpc.metaweblog), 106
description (zinnia.models.Category attribute), 76
description (zinnia.models.category.Category attribute), 79
description() (zinnia.feeds.AuthorEntries method), 57
description() (zinnia.feeds.CategoryEntries method), 58
description() (zinnia.feeds.EntryComments method), 58
description() (zinnia.feeds.EntryDiscussions method), 59
description() (zinnia.feeds.EntryPingbacks method), 60
description() (zinnia.feeds.EntryTrackbacks method), 60
description() (zinnia.feeds.LastDiscussions method), 60
description() (zinnia.feeds.LastEntries method), 61
description() (zinnia.feeds.SearchEntries method), 61
description() (zinnia.feeds.TagEntries method), 61
description_template (zinnia.feeds.DiscussionFeed attribute), 58
description_template (zinnia.feeds.EntryComments attribute), 58
description_template (zinnia.feeds.EntryFeed attribute), 59
description_template (zinnia.feeds.EntryPingbacks attribute), 60
description_template (zinnia.feeds.EntryTrackbacks attribute), 60
detail_template (zinnia.models_bases.entry.DetailTemplateEntry attribute), 88
DetailTemplateEntry (class in zinnia.models_bases.entry), 88
DetailTemplateEntry.Meta (class in zinnia.models_bases.entry), 88
DirectoryPinger (class in zinnia.ping), 64
disable_for_loaddata() (in module zinnia.signals), 66

disconnect_discussion_signals() (in module zinnia.signals), 66
disconnect_entry_signals() (in module zinnia.signals), 66
discussion_is_still_open() (zinnia.models_bases.entry.DiscussionsEntry method), 89
DiscussionFeed (class in zinnia.feeds), 58
discussions (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
DiscussionsEntry (class in zinnia.models_bases.entry), 88
DiscussionsEntry.Meta (class in zinnia.models_bases.entry), 89
dispatch() (zinnia.views.quick_entry.QuickEntry method), 100
dispatch() (zinnia.views.trackback.EntryTrackback method), 102
displayed_percent (zinnia.preview.HTMLPreview attribute), 64
displayed_words (zinnia.preview.HTMLPreview attribute), 65
do_email_authors() (zinnia.moderator.EntryCommentModerator method), 63
do_email_notification() (zinnia.moderator.EntryCommentModerator method), 63
do_email_reply() (zinnia.moderator.EntryCommentModerator method), 63

E

edit_post() (in module zinnia.xmlrpc.metaweblog), 106
email() (zinnia.moderator.EntryCommentModerator method), 63
email_authors (zinnia.moderator.EntryCommentModerator attribute), 63
email_reply (zinnia.moderator.EntryCommentModerator attribute), 63
enable_field (zinnia.moderator.EntryCommentModerator attribute), 63
end_publication (zinnia.models_bases.entry.CoreEntry attribute), 87
entries (zinnia.models.Author attribute), 75
entries (zinnia.models.author.Author attribute), 78
entries (zinnia.models.Category attribute), 76
entries (zinnia.models.category.Category attribute), 79
entries_published() (in module zinnia.managers), 62
entries_published() (zinnia.models.Author method), 75
entries_published() (zinnia.models.author.Author method), 78
entries_published() (zinnia.models.Category method), 76
entries_published() (zinnia.models.category.Category method), 79
Entry (class in zinnia.models), 72
Entry (class in zinnia.models.entry), 80
Entry.DoesNotExist, 73, 81
Entry.MultipleObjectsReturned, 73, 81
entry_breadcrumbs() (in module zinnia.breadcrumbs), 55
EntryAdmin (class in zinnia.admin.entry), 68
EntryAdminForm (class in zinnia.admin.forms), 71
EntryAdminForm.Meta (class in zinnia.admin.forms), 71
EntryArchiveMixin (class in zinnia.views.archives), 95
EntryArchiveTemplateResponseMixin (class in zinnia.views.mixins.templates), 105
EntryCacheMixin (class in zinnia.views.mixins.entry_cache), 104
EntryChannel (class in zinnia.views.channels), 99
EntryCommentModerator (class in zinnia.moderator), 63
EntryComments (class in zinnia.feeds), 58
EntryDateDetail (class in zinnia.views.entries), 99
EntryDay (class in zinnia.views.archives), 96
EntryDetail (class in zinnia.views.entries), 99
EntryDiscussions (class in zinnia.feeds), 59
EntryFeed (class in zinnia.feeds), 59
EntryIndex (class in zinnia.views.archives), 96
EntryMonth (class in zinnia.views.archives), 96
EntryPingbacks (class in zinnia.feeds), 60
EntryPreviewMixin (class in zinnia.views.mixins.entry_preview), 104
EntryProtectionMixin (class in zinnia.views.mixins.entry_protection), 104
EntryPublishedManager (class in zinnia.managers), 62
EntryPublishedVectorBuilder (class in zinnia.comparison), 56
EntryQuerysetArchiveTemplateResponseMixin (class in zinnia.views.mixins.templates), 105
EntryQuerysetArchiveTodayTemplateResponseMixin (class in zinnia.views.mixins.templates), 105
EntryQuerysetTemplateResponseMixin (class in zinnia.views.mixins.templates), 105
EntryRelatedPublishedManager (class in zinnia.managers), 62
EntryRelatedSitemap (class in zinnia.sitemaps), 66
EntrySearch (class in zinnia.views.search), 101
EntryShortLink (class in zinnia.views.shortlink), 101
EntrySitemap (class in zinnia.sitemaps), 67
EntryToday (class in zinnia.views.archives), 96
EntryTrackback (class in zinnia.views.trackback), 102
EntryTrackbacks (class in zinnia.feeds), 60
EntryWeek (class in zinnia.views.archives), 96
EntryYear (class in zinnia.views.archives), 97
error (zinnia.views.mixins.entry_protection.PasswordMixin attribute), 105
error (zinnia.views.search.BaseEntrySearch attribute), 100
excerpt (zinnia.models_bases.entry.ExcerptEntry attribute), 90
ExcerptEntry (class in zinnia.models_bases.entry), 89

ExcerptEntry.Meta (class in zinnia.models_bases.entry), 90
exclude (zinnia.views.quick_entry.QuickEntryForm.Meta attribute), 100
ExternalUrlsPinger (class in zinnia.ping), 64

F

FakeRequest (class in zinnia.xmlrpc.pingback), 107
featured (zinnia.models_bases.entry.FeaturedEntry attribute), 90
FeaturedEntry (class in zinnia.models_bases.entry), 90
FeaturedEntry.Meta (class in zinnia.models_bases.entry), 90
feed_copyright (zinnia.feeds.ZinniaFeed attribute), 61
feed_format (zinnia.feeds.ZinniaFeed attribute), 62
fields (zinnia.admin.category.CategoryAdmin attribute), 68
fields (zinnia.admin.forms.CategoryAdminForm.Meta attribute), 71
fields (zinnia.admin.forms.EntryAdminForm.Meta attribute), 71
fields (zinnia.comparison.EntryPublishedVectorBuilder attribute), 56
fields (zinnia.comparison.ModelVectorBuilder attribute), 57
fieldsets (zinnia.admin.entry.EntryAdmin attribute), 68
filter_horizontal (zinnia.admin.entry.EntryAdmin attribute), 68
find_external_urls() (zinnia.ping.ExternalUrlsPinger method), 64
find_pingback_href() (zinnia.ping.ExternalUrlsPinger method), 64
find_pingback_urls() (zinnia.ping.ExternalUrlsPinger method), 64
flush_similar_cache_handler() (in module zinnia.signals), 66
form (zinnia.admin.category.CategoryAdmin attribute), 68
form (zinnia.admin.entry.EntryAdmin attribute), 68
formatday() (zinnia.calendar.Calendar method), 55
formatfooter() (zinnia.calendar.Calendar method), 55
formatmonth() (zinnia.calendar.Calendar method), 55
formatmonthname() (zinnia.calendar.Calendar method), 55
formatweekday() (zinnia.calendar.Calendar method), 55
formatweekheader() (zinnia.calendar.Calendar method), 56
formfield_for_manytomany() (zinnia.admin.entry.EntryAdmin method), 69

G

generate_pingback_content() (in module zinnia.xmlrpc.pingback), 107
get() (zinnia.views.comments.CommentSuccess method), 99
get() (zinnia.views.mixins.entry_protection.EntryProtectionMixin method), 104
get() (zinnia.views.quick_entry.QuickEntry method), 100
get() (zinnia.views.trackback.EntryTrackback method), 102
get_absolute_url() (zinnia.models.Author method), 75
get_absolute_url() (zinnia.models.author.Author method), 78
get_absolute_url() (zinnia.models.Category method), 76
get_absolute_url() (zinnia.models.category.Category method), 79
get_absolute_url() (zinnia.models_bases.entry.CoreEntry method), 87
get_actions() (zinnia.admin.entry.EntryAdmin method), 69
get_archive_part_value() (zinnia.views.mixins.templates.EntryQuerysetArchiveTemplateRespo method), 105
get_archive_part_value() (zinnia.views.mixins.templates.EntryQuerysetArchiveTodayTemplate method), 105
get_archives_entries
template tag, 49
get_archives_entries() (in module zinnia.templatetags.zinnia), 49
get_archives_entries_tree
template tag, 49
get_archives_entries_tree() (in module zinnia.templatetags.zinnia), 50
get_authors
template tag, 50
get_authors() (in module zinnia.templatetags.zinnia), 50
get_authors() (in module zinnia.xmlrpc.metaweblog), 106
get_authors() (zinnia.admin.entry.EntryAdmin method), 69
get_cache() (zinnia.comparison.CachedModelVectorBuilder method), 56
get_calendar_entries
template tag, 49
get_calendar_entries() (in module zinnia.templatetags.zinnia), 49
get_categories
template tag, 50
get_categories() (in module zinnia.templatetags.zinnia), 50
get_categories() (in module zinnia.xmlrpc.metaweblog), 106
get_categories() (zinnia.admin.entry.EntryAdmin method), 69
get_categories_tree
template tag, 50
get_categories_tree() (in module zin-

nia.templatetags.zinnia), 50
get_category_or_404() (in module zinnia.views.categories), 97
get_changeform_initial_data() (zinnia.admin.entry.EntryAdmin method), 69
get_content_template_display() (zinnia.models.Entry method), 74
get_content_template_display() (zinnia.models.entry.Entry method), 82
get_content_template_display() (zinnia.models_bases.entry.AbstractEntry method), 84
get_content_template_display() (zinnia.models_bases.entry.ContentTemplateEntry method), 86
get_context_data() (zinnia.views.authors.BaseAuthorDetail method), 95
get_context_data() (zinnia.views.capabilities.CapabilityView method), 98
get_context_data() (zinnia.views.categories.BaseCategoryDetail method), 97
get_context_data() (zinnia.views.channels.BaseEntryChannel method), 98
get_context_data() (zinnia.views.comments.CommentSuccess method), 99
get_context_data() (zinnia.views.search.BaseEntrySearch method), 100
get_context_data() (zinnia.views.sitemap.Sitemap method), 101
get_context_data() (zinnia.views.tags.BaseTagDetail method), 101
get_context_first_matching_object() (in module zinnia.context), 57
get_context_first_object() (in module zinnia.context), 57
get_context_loop_positions() (in module zinnia.context), 57
get_dated_items() (zinnia.views.archives.EntryToday method), 96
get_dated_items() (zinnia.views.archives.EntryWeek method), 96
get_default_base_template_names() (zinnia.views.mixins.templates.EntryArchiveTemplateResponseMixin method), 105
get_default_base_template_names() (zinnia.views.mixins.templates.EntryQuerysetArchiveTemplateResponseMixin method), 105
get_detail_template_display() (zinnia.models.Entry method), 74
get_detail_template_display() (zinnia.models.entry.Entry method), 82
get_detail_template_display() (zinnia.models_bases.entry.AbstractEntry method), 84
get_detail_template_display() (zinnia.models_bases.entry.DetailTemplateEntry method), 88
get_draft_entries
    template tag, 48
get_draft_entries() (in module zinnia.templatetags.zinnia), 48
get_featured_entries
    template tag, 47
get_featured_entries() (in module zinnia.templatetags.zinnia), 48
get_gravatar
    template tag, 52
get_gravatar() (in module zinnia.templatetags.zinnia), 53
get_is_visible() (zinnia.admin.entry.EntryAdmin method), 69
get_latest_by (zinnia.models_bases.entry.CoreEntry.Meta attribute), 86
get_model_name() (zinnia.views.authors.AuthorDetail method), 95
get_model_name() (zinnia.views.categories.CategoryDetail method), 97
get_model_name() (zinnia.views.mixins.templates.EntryQuerysetTemplateResponseMixi method), 106
get_model_name() (zinnia.views.tags.TagDetail method), 102
get_model_type() (zinnia.views.mixins.templates.EntryQuerysetTemplateR method), 106
get_next_by_creation_date() (zinnia.models.Entry method), 74
get_next_by_creation_date() (zinnia.models.entry.Entry method), 82
get_next_by_creation_date() (zinnia.models_bases.entry.AbstractEntry method), 84
get_next_by_creation_date() (zinnia.models_bases.entry.CoreEntry method), 87
get_next_by_last_update() (zinnia.models.Entry method), 74
get_next_by_last_update() (zinnia.models.entry.Entry method), 82
get_next_by_last_update() (zinnia.models_bases.entry.AbstractEntry method), 84
get_next_by_last_update() (zinnia.models_bases.entry.CoreEntry method), 87

get_next_by_publication_date() (zinnia.models.Entry method), 74
get_next_by_publication_date() (zinnia.models.entry.Entry method), 82
get_next_by_publication_date() (zinnia.models_bases.entry.AbstractEntry method), 84
get_next_by_publication_date() (zinnia.models_bases.entry.CoreEntry method), 87
get_next_day() (zinnia.views.mixins.archives.PreviousNextPublishedMixin method), 103
get_next_month() (zinnia.views.mixins.archives.PreviousNextPublishedMixin method), 103
get_next_week() (zinnia.views.mixins.archives.PreviousNextPublishedMixin method), 103
get_next_year() (zinnia.views.mixins.archives.PreviousNextPublishedMixin method), 103
get_object() (zinnia.feeds.AuthorEntries method), 57
get_object() (zinnia.feeds.CategoryEntries method), 58
get_object() (zinnia.feeds.EntryDiscussions method), 59
get_object() (zinnia.feeds.SearchEntries method), 61
get_object() (zinnia.feeds.TagEntries method), 61
get_object() (zinnia.views.mixins.entry_cache.EntryCacheMixin method), 104
get_object() (zinnia.views.mixins.entry_preview.EntryPreviewMixin method), 104
get_object() (zinnia.views.trackback.EntryTrackback method), 102
get_popular_entries template tag, 48
get_popular_entries() (in module zinnia.templatetags.zinnia), 48
get_post() (in module zinnia.xmlrpc.metaweblog), 106
get_previous_by_creation_date() (zinnia.models.Entry method), 74
get_previous_by_creation_date() (zinnia.models.entry.Entry method), 82
get_previous_by_creation_date() (zinnia.models_bases.entry.AbstractEntry method), 84
get_previous_by_creation_date() (zinnia.models_bases.entry.CoreEntry method), 87
get_previous_by_last_update() (zinnia.models.Entry method), 74
get_previous_by_last_update() (zinnia.models.entry.Entry method), 82
get_previous_by_last_update() (zinnia.models_bases.entry.AbstractEntry method), 84
get_previous_by_last_update() (zinnia.models_bases.entry.CoreEntry method), 87
get_previous_by_publication_date() (zinnia.models.Entry method), 74
get_previous_by_publication_date() (zinnia.models.entry.Entry method), 82
get_previous_by_publication_date() (zinnia.models_bases.entry.AbstractEntry method), 84
get_previous_by_publication_date() (zinnia.models_bases.entry.CoreEntry method), 87
get_previous_day() (zinnia.views.mixins.archives.PreviousNextPublishedMixin method), 103
get_previous_month() (zinnia.views.mixins.archives.PreviousNextPublishedMixin method), 103
get_previous_next_published() (zinnia.views.mixins.archives.PreviousNextPublishedMixin method), 103
get_previous_week() (zinnia.views.mixins.archives.PreviousNextPublishedMixin method), 103
get_previous_year() (zinnia.views.mixins.archives.PreviousNextPublishedMixin method), 103
get_queryset() (zinnia.admin.entry.EntryAdmin method), 69
get_queryset() (zinnia.managers.EntryPublishedManager method), 62
get_queryset() (zinnia.managers.EntryRelatedPublishedManager method), 62
get_queryset() (zinnia.sitemaps.EntryRelatedSitemap method), 66
get_queryset() (zinnia.sitemaps.TagSitemap method), 67
get_queryset() (zinnia.views.authors.AuthorList method), 95
get_queryset() (zinnia.views.authors.BaseAuthorDetail method), 95
get_queryset() (zinnia.views.categories.BaseCategoryDetail method), 97
get_queryset() (zinnia.views.categories.CategoryList method), 97
get_queryset() (zinnia.views.channels.BaseEntryChannel method), 98
get_queryset() (zinnia.views.mixins.callable_queryset.CallableQuerysetMix method), 104
get_queryset() (zinnia.views.search.BaseEntrySearch method), 100
get_queryset() (zinnia.views.tags.BaseTagDetail method), 102
get_queryset() (zinnia.views.tags.TagList method), 102
get_random_entries template tag, 48
get_random_entries() (in module zin-

nia.templatetags.zinnia), 48
get_readonly_fields() (zinnia.admin.entry.EntryAdmin method), 69
get_recent_comments
template tag, 51
get_recent_comments() (in module zinnia.templatetags.zinnia), 51
get_recent_entries
template tag, 47
get_recent_entries() (in module zinnia.templatetags.zinnia), 47
get_recent_linkbacks
template tag, 51
get_recent_linkbacks() (in module zinnia.templatetags.zinnia), 51
get_recent_posts() (in module zinnia.xmlrpc.metaweblog), 106
get_redirect_url() (zinnia.views.shortlink.EntryShortLink method), 101
get_related() (zinnia.comparison.CachedModelVectorBuilder method), 56
get_related() (zinnia.comparison.ModelVectorBuilder method), 57
get_short_url() (zinnia.admin.entry.EntryAdmin method), 69
get_similar_entries
template tag, 49
get_similar_entries() (in module zinnia.templatetags.zinnia), 49
get_sites() (zinnia.admin.entry.EntryAdmin method), 69
get_spam_checker() (in module zinnia.spam_checker), 92
get_status_display() (zinnia.models.Entry method), 74
get_status_display() (zinnia.models.entry.Entry method), 82
get_status_display() (zinnia.models_bases.entry.AbstractEntry method), 84
get_status_display() (zinnia.models_bases.entry.CoreEntry method), 87
get_tag_cloud
template tag, 51
get_tag_cloud() (in module zinnia.templatetags.zinnia), 51
get_tags
template tag, 50
get_tags() (in module zinnia.templatetags.zinnia), 50
get_tags() (in module zinnia.xmlrpc.metaweblog), 106
get_tags() (zinnia.admin.entry.EntryAdmin method), 69
get_tags() (zinnia.admin.widgets.TagAutoComplete method), 72
get_template_names() (zinnia.views.mixins.templates.EntryQuerysetArchiveTemplateResponseMixin method), 105
get_template_names() (zinnia.views.mixins.templates.EntryQuerysetTemplateResponseMixi method), 106
get_title() (zinnia.admin.entry.EntryAdmin method), 69
get_title() (zinnia.feeds.AuthorEntries method), 57
get_title() (zinnia.feeds.CategoryEntries method), 58
get_title() (zinnia.feeds.EntryComments method), 58
get_title() (zinnia.feeds.EntryDiscussions method), 59
get_title() (zinnia.feeds.EntryPingbacks method), 60
get_title() (zinnia.feeds.EntryTrackbacks method), 60
get_title() (zinnia.feeds.LastDiscussions method), 60
get_title() (zinnia.feeds.LastEntries method), 61
get_title() (zinnia.feeds.SearchEntries method), 61
get_title() (zinnia.feeds.TagEntries method), 61
get_title() (zinnia.feeds.ZinniaFeed method), 62
get_tree_path() (zinnia.admin.category.CategoryAdmin method), 68
get_url_shortener() (in module zinnia.url_shortener), 93
get_user_flagger() (in module zinnia.flags), 62
get_user_info() (in module zinnia.xmlrpc.metaweblog), 106
get_users_blogs() (in module zinnia.xmlrpc.metaweblog), 107

H

handle_page_crumb() (in module zinnia.breadcrumbs), 55
has_more (zinnia.preview.HTMLPreview attribute), 65
html_content (zinnia.models_bases.entry.ContentEntry attribute), 86
html_format() (in module zinnia.markups), 63
html_lead (zinnia.models_bases.entry.LeadEntry attribute), 90
html_preview (zinnia.models_bases.entry.ContentEntry attribute), 86
htmlize() (zinnia.views.quick_entry.QuickEntry method), 100
HTMLPreview (class in zinnia.preview), 64
HumansTxt (class in zinnia.views.capabilities), 98

I

i18n_url() (in module zinnia.urls), 93
id (zinnia.models.Category attribute), 76
id (zinnia.models.category.Category attribute), 79
id (zinnia.models.Entry attribute), 74
id (zinnia.models.entry.Entry attribute), 82
image_caption (zinnia.models_bases.entry.ImageEntry attribute), 90
image_upload_to() (zinnia.models_bases.entry.ImageEntry method), 90
image_upload_to_dispatcher() (in module zinnia.models_bases.entry), 92
ImageEntry (class in zinnia.models_bases.entry), 90

ImageEntry.Meta (class in zinnia.models_bases.entry), 90
index_together (zinnia.models_bases.entry.CoreEntry.Meta attribute), 86
is_actual (zinnia.models_bases.entry.CoreEntry attribute), 87
is_external_url() (zinnia.ping.ExternalUrlsPinger method), 64
is_visible (zinnia.models_bases.entry.CoreEntry attribute), 87
item_author_email() (zinnia.feeds.DiscussionFeed method), 58
item_author_email() (zinnia.feeds.EntryFeed method), 59
item_author_link() (zinnia.feeds.DiscussionFeed method), 58
item_author_link() (zinnia.feeds.EntryFeed method), 59
item_author_name() (zinnia.feeds.DiscussionFeed method), 58
item_author_name() (zinnia.feeds.EntryFeed method), 59
item_categories() (zinnia.feeds.EntryFeed method), 59
item_enclosure_length() (zinnia.feeds.EntryComments method), 58
item_enclosure_length() (zinnia.feeds.EntryFeed method), 59
item_enclosure_mime_type() (zinnia.feeds.EntryComments method), 59
item_enclosure_mime_type() (zinnia.feeds.EntryFeed method), 59
item_enclosure_url() (zinnia.feeds.EntryComments method), 59
item_enclosure_url() (zinnia.feeds.EntryFeed method), 59
item_link() (zinnia.feeds.DiscussionFeed method), 58
item_link() (zinnia.feeds.EntryComments method), 59
item_link() (zinnia.feeds.EntryPingbacks method), 60
item_link() (zinnia.feeds.EntryTrackbacks method), 60
item_pubdate() (zinnia.feeds.DiscussionFeed method), 58
item_pubdate() (zinnia.feeds.EntryFeed method), 60
item_updateddate() (zinnia.feeds.EntryFeed method), 60
items() (zinnia.feeds.AuthorEntries method), 58
items() (zinnia.feeds.CategoryEntries method), 58
items() (zinnia.feeds.EntryComments method), 59
items() (zinnia.feeds.EntryDiscussions method), 59
items() (zinnia.feeds.EntryPingbacks method), 60
items() (zinnia.feeds.EntryTrackbacks method), 60
items() (zinnia.feeds.LastDiscussions method), 60
items() (zinnia.feeds.LastEntries method), 61
items() (zinnia.feeds.SearchEntries method), 61
items() (zinnia.feeds.TagEntries method), 61
items() (zinnia.sitemaps.EntryRelatedSitemap method), 66
items() (zinnia.sitemaps.EntrySitemap method), 67

L

label (zinnia.apps.ZinniaConfig attribute), 55
label_from_instance() (zinnia.admin.fields.MPTTModelMultipleChoiceField method), 70
last_update (zinnia.models_bases.entry.CoreEntry attribute), 87
LastDiscussions (class in zinnia.feeds), 60
LastEntries (class in zinnia.feeds), 61
lastmod() (zinnia.sitemaps.EntryRelatedSitemap method), 66
lastmod() (zinnia.sitemaps.EntrySitemap method), 67
lead (zinnia.models_bases.entry.LeadEntry attribute), 91
LeadEntry (class in zinnia.models_bases.entry), 90
LeadEntry.Meta (class in zinnia.models_bases.entry), 90
level (zinnia.models.Category attribute), 76
level (zinnia.models.category.Category attribute), 79
lft (zinnia.models.Category attribute), 76
lft (zinnia.models.category.Category attribute), 79
limit (zinnia.comparison.EntryPublishedVectorBuilder attribute), 56
limit (zinnia.comparison.ModelVectorBuilder attribute), 57
limit (zinnia.feeds.ZinniaFeed attribute), 62
link() (zinnia.feeds.AuthorEntries method), 58
link() (zinnia.feeds.CategoryEntries method), 58
link() (zinnia.feeds.EntryDiscussions method), 59
link() (zinnia.feeds.LastDiscussions method), 60
link() (zinnia.feeds.LastEntries method), 61
link() (zinnia.feeds.SearchEntries method), 61
link() (zinnia.feeds.TagEntries method), 61
list_display (zinnia.admin.category.CategoryAdmin attribute), 68
list_display (zinnia.admin.entry.EntryAdmin attribute), 69
list_filter (zinnia.admin.category.CategoryAdmin attribute), 68
list_filter (zinnia.admin.entry.EntryAdmin attribute), 69
load_model_class() (in module zinnia.models_bases), 82
location() (zinnia.sitemaps.TagSitemap method), 67
login() (zinnia.views.mixins.entry_protection.LoginMixin method), 105
login_required (zinnia.models_bases.entry.LoginRequiredEntry attribute), 91
LoginMixin (class in zinnia.views.mixins.entry_protection), 105
LoginRequiredEntry (class in zinnia.models_bases.entry), 91
LoginRequiredEntry.Meta (class in zinnia.models_bases.entry), 91
lookup_key (zinnia.admin.filters.AuthorListFilter attribute), 70
lookup_key (zinnia.admin.filters.CategoryListFilter attribute), 70

lookup_key (zinnia.admin.filters.RelatedPublishedFilter attribute), 70
lookups() (zinnia.admin.filters.RelatedPublishedFilter method), 71
loop_template_list() (in module zinnia.templates), 67

M

mail_comment_notification_recipients (zinnia.moderator.EntryCommentModerator attribute), 63
make_hidden() (zinnia.admin.entry.EntryAdmin method), 69
make_mine() (zinnia.admin.entry.EntryAdmin method), 69
make_object_list (zinnia.views.archives.EntryYear attribute), 97
make_published() (zinnia.admin.entry.EntryAdmin method), 69
mark_featured() (zinnia.admin.entry.EntryAdmin method), 69
markdown() (in module zinnia.markups), 63
media (zinnia.admin.category.CategoryAdmin attribute), 68
media (zinnia.admin.entry.EntryAdmin attribute), 69
media (zinnia.admin.forms.CategoryAdminForm attribute), 71
media (zinnia.admin.forms.EntryAdminForm attribute), 71
media (zinnia.admin.widgets.MiniTextarea attribute), 72
media (zinnia.admin.widgets.MPTTFilteredSelectMultiple attribute), 72
media (zinnia.admin.widgets.TagAutoComplete attribute), 72
media (zinnia.views.quick_entry.QuickEntryForm attribute), 100
META (zinnia.xmlrpc.pingback.FakeRequest attribute), 107
MiniTextarea (class in zinnia.admin.widgets), 72
model (zinnia.admin.filters.AuthorListFilter attribute), 70
model (zinnia.admin.filters.CategoryListFilter attribute), 70
model (zinnia.admin.filters.RelatedPublishedFilter attribute), 71
model (zinnia.admin.forms.CategoryAdminForm.Meta attribute), 71
model (zinnia.admin.forms.EntryAdminForm.Meta attribute), 71
model (zinnia.sitemaps.AuthorSitemap attribute), 66
model (zinnia.sitemaps.CategorySitemap attribute), 66
model (zinnia.sitemaps.EntryRelatedSitemap attribute), 67
model (zinnia.views.quick_entry.QuickEntryForm.Meta attribute), 100
model_name (zinnia.views.mixins.templates.EntryQuerysetTemplateRespon attribute), 106
model_type (zinnia.views.authors.AuthorDetail attribute), 95
model_type (zinnia.views.categories.CategoryDetail attribute), 97
model_type (zinnia.views.mixins.templates.EntryQuerysetTemplateRespons attribute), 106
model_type (zinnia.views.tags.TagDetail attribute), 102
ModelVectorBuilder (class in zinnia.comparison), 56
moderate() (zinnia.moderator.EntryCommentModerator method), 63
month_crumb() (in module zinnia.breadcrumbs), 55
month_format (zinnia.views.mixins.archives.ArchiveMixin attribute), 103
MPTTFilteredSelectMultiple (class in zinnia.admin.widgets), 72
MPTTModelChoiceIterator (class in zinnia.admin.fields), 70
MPTTModelMultipleChoiceField (class in zinnia.admin.fields), 70

N

name (zinnia.apps.ZinniaConfig attribute), 55
new_category() (in module zinnia.xmlrpc.metaweblog), 107
new_media_object() (in module zinnia.xmlrpc.metaweblog), 107
new_post() (in module zinnia.xmlrpc.metaweblog), 107
next_entry (zinnia.models_bases.entry.CoreEntry attribute), 87

O

objects (zinnia.models.Category attribute), 76
objects (zinnia.models.category.Category attribute), 80
objects (zinnia.models_bases.entry.CoreEntry attribute), 87
on_site() (zinnia.managers.EntryPublishedManager method), 62
OpenSearchXml (class in zinnia.views.capabilities), 98
ordering (zinnia.models_bases.entry.CoreEntry.Meta attribute), 86

P

paginate_by (zinnia.views.authors.AuthorDetail attribute), 95
paginate_by (zinnia.views.categories.CategoryDetail attribute), 97
paginate_by (zinnia.views.channels.EntryChannel attribute), 99
paginate_by (zinnia.views.mixins.archives.ArchiveMixin attribute), 103
paginate_by (zinnia.views.search.EntrySearch attribute), 101

paginate_by (zinnia.views.tags.TagDetail attribute), 102
parameter_name (zinnia.admin.filters.AuthorListFilter attribute), 70
parameter_name (zinnia.admin.filters.CategoryListFilter attribute), 70
parent (zinnia.models.Category attribute), 76
parent (zinnia.models.category.Category attribute), 80
parent_id (zinnia.models.Category attribute), 77
parent_id (zinnia.models.category.Category attribute), 80
password (zinnia.models_bases.entry.PasswordRequiredEntry attribute), 91
password() (zinnia.views.mixins.entry_protection.PasswordMixin method), 105
PasswordMixin (class in zinnia.views.mixins.entry_protection), 105
PasswordRequiredEntry (class in zinnia.models_bases.entry), 91
PasswordRequiredEntry.Meta (class in zinnia.models_bases.entry), 91
pattern (zinnia.views.search.BaseEntrySearch attribute), 101
pearson_score() (in module zinnia.comparison), 57
permanent (zinnia.views.shortlink.EntryShortLink attribute), 101
permissions (zinnia.models_bases.entry.CoreEntry.Meta attribute), 87
ping_directories() (zinnia.admin.entry.EntryAdmin method), 69
ping_directories_handler() (in module zinnia.signals), 66
ping_entry() (zinnia.ping.DirectoryPinger method), 64
ping_external_urls_handler() (in module zinnia.signals), 66
pingback_count (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
pingback_enabled (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
pingback_extensions_get_pingbacks() (in module zinnia.xmlrpc.pingback), 107
pingback_ping() (in module zinnia.xmlrpc.pingback), 107
pingback_url() (zinnia.ping.ExternalUrlsPinger method), 64
pingbacks (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
pingbacks_are_open (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
post() (zinnia.views.mixins.entry_protection.EntryProtectionMixin method), 104
post() (zinnia.views.quick_entry.QuickEntry method), 100
post() (zinnia.views.trackback.EntryTrackback method), 102
post_structure() (in module zinnia.xmlrpc.metaweblog), 107
prepopulated_fields (zinnia.admin.category.CategoryAdmin attribute), 68
prepopulated_fields (zinnia.admin.entry.EntryAdmin attribute), 69
preview (zinnia.preview.HTMLPreview attribute), 65
previous_entry (zinnia.models_bases.entry.CoreEntry attribute), 87
previous_next_entries (zinnia.models_bases.entry.CoreEntry attribute), 87
PreviousNextPublishedMixin (class in zinnia.views.mixins.archives), 103
priority (zinnia.sitemaps.EntrySitemap attribute), 67
priority() (zinnia.sitemaps.EntryRelatedSitemap method), 67
protocol (zinnia.feeds.ZinniaFeed attribute), 62
protocol (zinnia.sitemaps.ZinniaSitemap attribute), 67
publication_date (zinnia.models_bases.entry.CoreEntry attribute), 87
published (zinnia.models.author.AuthorPublishedManager attribute), 78
published (zinnia.models.Category attribute), 77
published (zinnia.models.category.Category attribute), 80
published (zinnia.models_bases.entry.CoreEntry attribute), 87
put_on_top() (zinnia.admin.entry.EntryAdmin method), 69
Python Enhancement Proposals
    PEP 8, 35

Q

query (zinnia.views.channels.BaseEntryChannel attribute), 98
queryset (zinnia.comparison.EntryPublishedVectorBuilder attribute), 56
queryset (zinnia.comparison.ModelVectorBuilder attribute), 57
queryset (zinnia.views.mixins.callable_queryset.CallableQuerysetMixin attribute), 104
queryset() (zinnia.admin.filters.RelatedPublishedFilter method), 71
queryset() (zinnia.views.archives.EntryArchiveMixin method), 96
queryset() (zinnia.views.entries.EntryDateDetail method), 99
QuickEntry (class in zinnia.views.quick_entry), 100
QuickEntryForm (class in zinnia.views.quick_entry), 100
QuickEntryForm.Meta (class in zinnia.views.quick_entry), 100

R

radio_fields (zinnia.admin.entry.EntryAdmin attribute), 69
raw_clean() (zinnia.comparison.ModelVectorBuilder method), 57
raw_dataset (zinnia.comparison.ModelVectorBuilder attribute), 57
ready() (zinnia.apps.ZinniaConfig method), 55
related (zinnia.models.Entry attribute), 74
related (zinnia.models.entry.Entry attribute), 82
related (zinnia.models_bases.entry.AbstractEntry attribute), 84
related (zinnia.models_bases.entry.RelatedEntry attribute), 91
related_published (zinnia.models_bases.entry.RelatedEntry attribute), 91
RelatedEntry (class in zinnia.models_bases.entry), 91
RelatedEntry.Meta (class in zinnia.models_bases.entry), 91
RelatedPublishedFilter (class in zinnia.admin.filters), 70
remaining_percent (zinnia.preview.HTMLPreview attribute), 65
remaining_words (zinnia.preview.HTMLPreview attribute), 65
render() (zinnia.admin.widgets.TagAutoComplete method), 72
render_option() (zinnia.admin.widgets.MPTTFilteredSelectMultiple method), 72
render_options() (zinnia.admin.widgets.MPTTFilteredSelectMultiple method), 72
restructuredtext() (in module zinnia.markups), 63
retrieve_breadcrumbs() (in module zinnia.breadcrumbs), 55
rght (zinnia.models.Category attribute), 77
rght (zinnia.models.category.Category attribute), 80
rows (zinnia.admin.widgets.MiniTextarea attribute), 72
RsdXml (class in zinnia.views.capabilities), 98
run() (zinnia.ping.DirectoryPinger method), 64
run() (zinnia.ping.ExternalUrlsPinger method), 64

S

safe_get_user_model() (in module zinnia.models.author), 78
save() (zinnia.models_bases.entry.CoreEntry method), 87
save() (zinnia.models_bases.entry.ExcerptEntry method), 90
search() (zinnia.managers.EntryPublishedManager method), 62
search_fields (zinnia.admin.category.CategoryAdmin attribute), 68
search_fields (zinnia.admin.entry.EntryAdmin attribute), 69
SearchEntries (class in zinnia.feeds), 61
session_key (zinnia.views.mixins.entry_protection.EntryProtectionMixin attribute), 105
set_cache() (zinnia.comparison.CachedModelVectorBuilder method), 56
set_max_entries() (zinnia.sitemaps.EntryRelatedSitemap method), 67
setting
    ZINNIA_ALLOW_EMPTY, 43
    ZINNIA_ALLOW_FUTURE, 44
    ZINNIA_AUTO_CLOSE_COMMENTS_AFTER, 45
    ZINNIA_AUTO_CLOSE_PINGBACKS_AFTER, 46
    ZINNIA_AUTO_CLOSE_TRACKBACKS_AFTER, 46
    ZINNIA_AUTO_MODERATE_COMMENTS, 45
    ZINNIA_COMMENT_MIN_WORDS, 45
    ZINNIA_COMPARISON_FIELDS, 47
    ZINNIA_COPYRIGHT, 47
    ZINNIA_DEFAULT_USER_ID, 45
    ZINNIA_ENTRY_BASE_MODEL, 41
    ZINNIA_ENTRY_CONTENT_TEMPLATES, 42
    ZINNIA_ENTRY_DETAIL_TEMPLATES, 42
    ZINNIA_ENTRY_LOOP_TEMPLATES, 42
    ZINNIA_FEEDS_FORMAT, 44
    ZINNIA_FEEDS_MAX_ITEMS, 44
    ZINNIA_MAIL_COMMENT_AUTHORS, 45
    ZINNIA_MAIL_COMMENT_NOTIFICATION_RECIPIENTS, 45
    ZINNIA_MAIL_COMMENT_REPLY, 45
    ZINNIA_MARKDOWN_EXTENSIONS, 43
    ZINNIA_MARKUP_LANGUAGE, 42
    ZINNIA_PAGINATION, 43
    ZINNIA_PING_DIRECTORIES, 46
    ZINNIA_PING_EXTERNAL_URLS, 46
    ZINNIA_PINGBACK_CONTENT_LENGTH, 46
    ZINNIA_PREVIEW_MAX_WORDS, 43
    ZINNIA_PREVIEW_MORE_STRING, 43
    ZINNIA_PREVIEW_SPLITTERS, 43
    ZINNIA_PROTOCOL, 44
    ZINNIA_RESTRUCTUREDTEXT_SETTINGS, 43
    ZINNIA_SAVE_PING_DIRECTORIES, 46
    ZINNIA_SEARCH_FIELDS, 47
    ZINNIA_SPAM_CHECKER_BACKENDS, 45
    ZINNIA_TRANSLATED_URLS, 44
    ZINNIA_UPLOAD_TO, 42
    ZINNIA_URL_SHORTENER_BACKEND, 44
short_url (zinnia.models_bases.entry.CoreEntry attribute), 87
site (zinnia.feeds.ZinniaFeed attribute), 62
site_url (zinnia.feeds.ZinniaFeed attribute), 62
Sitemap (class in zinnia.views.sitemap), 101
sites (zinnia.models.Entry attribute), 74
sites (zinnia.models.entry.Entry attribute), 82

sites (zinnia.models_bases.entry.AbstractEntry attribute), 84
sites (zinnia.models_bases.entry.CoreEntry attribute), 88
slug (zinnia.models.Category attribute), 77
slug (zinnia.models.category.Category attribute), 80
slug (zinnia.models_bases.entry.CoreEntry attribute), 88
spam_checker_backends (zinnia.moderator.EntryCommentModerator attribute), 63
split() (zinnia.preview.HTMLPreview method), 65
start_publication (zinnia.models_bases.entry.CoreEntry attribute), 88
status (zinnia.models_bases.entry.CoreEntry attribute), 88
STATUS_CHOICES (zinnia.models_bases.entry.CoreEntry attribute), 87

T

tag_structure() (in module zinnia.xmlrpc.metaweblog), 107
TagAutoComplete (class in zinnia.admin.widgets), 72
TagDetail (class in zinnia.views.tags), 102
TagEntries (class in zinnia.feeds), 61
TagList (class in zinnia.views.tags), 102
tags_list (zinnia.models_bases.entry.TagsEntry attribute), 92
tags_published() (in module zinnia.managers), 62
TagsEntry (class in zinnia.models_bases.entry), 91
TagsEntry.Meta (class in zinnia.models_bases.entry), 92
TagSitemap (class in zinnia.sitemaps), 67
template filter
    comment_admin_urlname, 53
    user_admin_urlname, 53
    week_number, 53
    widont, 53
template tag
    get_archives_entries, 49
    get_archives_entries_tree, 49
    get_authors, 50
    get_calendar_entries, 49
    get_categories, 50
    get_categories_tree, 50
    get_draft_entries, 48
    get_featured_entries, 47
    get_gravatar, 52
    get_popular_entries, 48
    get_random_entries, 48
    get_recent_comments, 51
    get_recent_entries, 47
    get_recent_linkbacks, 51
    get_similar_entries, 49
    get_tag_cloud, 51
    get_tags, 50
    zinnia_breadcrumbs, 52
    zinnia_loop_template, 52
    zinnia_pagination, 51
    zinnia_statistics, 52
template_name (zinnia.views.capabilities.HumansTxt attribute), 98
template_name (zinnia.views.capabilities.OpenSearchXml attribute), 98
template_name (zinnia.views.capabilities.RsdXml attribute), 98
template_name (zinnia.views.capabilities.WLWManifestXml attribute), 98
template_name (zinnia.views.comments.CommentSuccess attribute), 99
template_name (zinnia.views.sitemap.Sitemap attribute), 101
template_name (zinnia.views.tags.TagList attribute), 102
template_name (zinnia.views.trackback.EntryTrackback attribute), 102
template_name_suffix (zinnia.views.archives.EntryDay attribute), 96
template_name_suffix (zinnia.views.archives.EntryMonth attribute), 96
template_name_suffix (zinnia.views.archives.EntryToday attribute), 96
template_name_suffix (zinnia.views.archives.EntryWeek attribute), 97
template_name_suffix (zinnia.views.archives.EntryYear attribute), 97
template_name_suffix (zinnia.views.mixins.templates.EntryQuerysetArchiveTemplateRespo attribute), 105
template_name_suffix (zinnia.views.search.EntrySearch attribute), 101
textile() (in module zinnia.markups), 63
title (zinnia.admin.filters.AuthorListFilter attribute), 70
title (zinnia.admin.filters.CategoryListFilter attribute), 70
title (zinnia.models.Category attribute), 77
title (zinnia.models.category.Category attribute), 80
title (zinnia.models_bases.entry.CoreEntry attribute), 88
title() (zinnia.feeds.ZinniaFeed method), 62
title_template (zinnia.feeds.DiscussionFeed attribute), 58
title_template (zinnia.feeds.EntryComments attribute), 59
title_template (zinnia.feeds.EntryFeed attribute), 60
title_template (zinnia.feeds.EntryPingbacks attribute), 60
title_template (zinnia.feeds.EntryTrackbacks attribute), 60
today (zinnia.views.mixins.templates.EntryQuerysetArchiveTodayTemplate attribute), 105
total_words (zinnia.preview.HTMLPreview attribute), 65
trackback_count (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
trackback_enabled (zin-

nia.models_bases.entry.DiscussionsEntry attribute), 89
trackbacks (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
trackbacks_are_open (zinnia.models_bases.entry.DiscussionsEntry attribute), 89
tree_id (zinnia.models.Category attribute), 77
tree_id (zinnia.models.category.Category attribute), 80
tree_path (zinnia.models.Category attribute), 77
tree_path (zinnia.models.category.Category attribute), 80
truncate() (zinnia.preview.HTMLPreview method), 65

U

union_q() (in module zinnia.search), 65
unmark_featured() (zinnia.admin.entry.EntryAdmin method), 69
URLRessources (class in zinnia.ping), 64
user_admin_urlname
template filter, 53
user_admin_urlname() (in module zinnia.templatetags.zinnia), 53
user_structure() (in module zinnia.xmlrpc.metaweblog), 107

V

verbose_name (zinnia.apps.ZinniaConfig attribute), 55
verbose_name (zinnia.models_bases.entry.CoreEntry.Meta attribute), 87
verbose_name_plural (zinnia.models_bases.entry.CoreEntry.Meta attribute), 87
version() (in module zinnia.context_processors), 57

W

week_format (zinnia.views.mixins.archives.ArchiveMixin attribute), 103
week_number
template filter, 53
week_number() (in module zinnia.templatetags.zinnia), 53
widgets (zinnia.admin.forms.EntryAdminForm.Meta attribute), 71
widont
template filter, 53
widont() (in module zinnia.templatetags.zinnia), 53
WLWManifestXml (class in zinnia.views.capabilities), 98
word_count (zinnia.models_bases.entry.ContentEntry attribute), 86

Y

year_crumb() (in module zinnia.breadcrumbs), 55

Z

zinnia (module), 7
zinnia.__init__ (module), 54
zinnia.admin (module), 68
zinnia.admin.category (module), 68
zinnia.admin.entry (module), 68
zinnia.admin.fields (module), 70
zinnia.admin.filters (module), 70
zinnia.admin.forms (module), 71
zinnia.admin.widgets (module), 72
zinnia.apps (module), 54
zinnia.breadcrumbs (module), 55
zinnia.calendar (module), 55
zinnia.comparison (module), 56
zinnia.context (module), 57
zinnia.context_processors (module), 57
zinnia.feeds (module), 57
zinnia.flags (module), 62
zinnia.managers (module), 62
zinnia.markups (module), 63
zinnia.models (module), 72
zinnia.models.author (module), 77
zinnia.models.category (module), 78
zinnia.models.entry (module), 19, 28, 31, 80
zinnia.models_bases (module), 82
zinnia.models_bases.entry (module), 83
zinnia.moderator (module), 63
zinnia.ping (module), 64
zinnia.preview (module), 64
zinnia.search (module), 16, 65
zinnia.settings (module), 41
zinnia.signals (module), 65
zinnia.sitemaps (module), 10, 66
zinnia.spam_checker (module), 18, 92
zinnia.spam_checker.backends (module), 92
zinnia.spam_checker.backends.all_is_spam (module), 92
zinnia.spam_checker.backends.long_enough (module), 92
zinnia.templates (module), 67
zinnia.templatetags (module), 47
zinnia.templatetags.zinnia (module), 47
zinnia.tests (module), 39
zinnia.url_shortener (module), 17, 93
zinnia.url_shortener.backends (module), 93
zinnia.url_shortener.backends.default (module), 93
zinnia.urls (module), 93
zinnia.urls.archives (module), 93
zinnia.urls.authors (module), 93
zinnia.urls.capabilities (module), 94
zinnia.urls.categories (module), 94
zinnia.urls.comments (module), 94
zinnia.urls.entries (module), 94
zinnia.urls.feeds (module), 94
zinnia.urls.quick_entry (module), 94

zinnia.urls.search (module), 94
zinnia.urls.shortlink (module), 94
zinnia.urls.sitemap (module), 94
zinnia.urls.tags (module), 94
zinnia.urls.trackback (module), 94
zinnia.views (module), 95
zinnia.views.archives (module), 95
zinnia.views.authors (module), 95
zinnia.views.capabilities (module), 98
zinnia.views.categories (module), 97
zinnia.views.channels (module), 15, 98
zinnia.views.comments (module), 99
zinnia.views.entries (module), 99
zinnia.views.mixins (module), 103
zinnia.views.mixins.archives (module), 103
zinnia.views.mixins.callable_queryset (module), 104
zinnia.views.mixins.entry_cache (module), 104
zinnia.views.mixins.entry_preview (module), 104
zinnia.views.mixins.entry_protection (module), 104
zinnia.views.mixins.templates (module), 105
zinnia.views.quick_entry (module), 100
zinnia.views.search (module), 15, 100
zinnia.views.shortlink (module), 101
zinnia.views.sitemap (module), 101
zinnia.views.tags (module), 101
zinnia.views.trackback (module), 102
zinnia.xmlrpc (module), 12, 106
zinnia.xmlrpc.metaweblog (module), 106
zinnia.xmlrpc.pingback (module), 107
ZINNIA_ALLOW_EMPTY
    setting, 43
ZINNIA_ALLOW_FUTURE
    setting, 44
ZINNIA_AUTO_CLOSE_COMMENTS_AFTER
    setting, 45
ZINNIA_AUTO_CLOSE_PINGBACKS_AFTER
    setting, 46
ZINNIA_AUTO_CLOSE_TRACKBACKS_AFTER
    setting, 46
ZINNIA_AUTO_MODERATE_COMMENTS
    setting, 45
zinnia_breadcrumbs
    template tag, 52
zinnia_breadcrumbs() (in module zinnia.templatetags.zinnia), 52
ZINNIA_COMMENT_MIN_WORDS
    setting, 45
ZINNIA_COMPARISON_FIELDS
    setting, 47
ZINNIA_COPYRIGHT
    setting, 47
ZINNIA_DEFAULT_USER_ID
    setting, 45
ZINNIA_ENTRY_BASE_MODEL
    setting, 41
ZINNIA_ENTRY_CONTENT_TEMPLATES
    setting, 42
ZINNIA_ENTRY_DETAIL_TEMPLATES
    setting, 42
ZINNIA_ENTRY_LOOP_TEMPLATES
    setting, 42
ZINNIA_FEEDS_FORMAT
    setting, 44
ZINNIA_FEEDS_MAX_ITEMS
    setting, 44
zinnia_loop_template
    template tag, 52
zinnia_loop_template() (in module zinnia.templatetags.zinnia), 52
ZINNIA_MAIL_COMMENT_AUTHORS
    setting, 45
ZINNIA_MAIL_COMMENT_NOTIFICATION_RECIPIENTS
    setting, 45
ZINNIA_MAIL_COMMENT_REPLY
    setting, 45
ZINNIA_MARKDOWN_EXTENSIONS
    setting, 43
ZINNIA_MARKUP_LANGUAGE
    setting, 42
ZINNIA_PAGINATION
    setting, 43
zinnia_pagination
    template tag, 51
zinnia_pagination() (in module zinnia.templatetags.zinnia), 51
ZINNIA_PING_DIRECTORIES
    setting, 46
ZINNIA_PING_EXTERNAL_URLS
    setting, 46
ZINNIA_PINGBACK_CONTENT_LENGTH
    setting, 46
ZINNIA_PREVIEW_MAX_WORDS
    setting, 43
ZINNIA_PREVIEW_MORE_STRING
    setting, 43
ZINNIA_PREVIEW_SPLITTERS
    setting, 43
ZINNIA_PROTOCOL
    setting, 44
ZINNIA_RESTRUCTUREDTEXT_SETTINGS
    setting, 43
ZINNIA_SAVE_PING_DIRECTORIES
    setting, 46
ZINNIA_SEARCH_FIELDS
    setting, 47
ZINNIA_SPAM_CHECKER_BACKENDS
    setting, 45
zinnia_statistics

template tag, 52
zinnia_statistics() (in module zinnia.templatetags.zinnia), 52
ZINNIA_TRANSLATED_URLS
setting, 44
ZINNIA_UPLOAD_TO
setting, 42
ZINNIA_URL_SHORTENER_BACKEND
setting, 44
ZinniaConfig (class in zinnia.apps), 54
ZinniaFeed (class in zinnia.feeds), 61
ZinniaSitemap (class in zinnia.sitemaps), 67