django-blog-zinnia Documentation

Release 0.11.2

Fantomas42

July 18, 2012

CONTENTS

Welcome to the version 0.11.2 of the documentation.

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

CHAPTER

ONE

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 ?

1.1 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
- Edition 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 and MarkItUp support
- Ready to use and extendable templates
- Compass and Sass3 integration
- Windows Live Writer compatibility

1.2 Examples

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

- Fantomas' side / Mobile version.
- Professional Web Studio.
- mixedCase.
- Tryolabs.
- brainbreach.
- Mauro Bianchi.
- AR.Drone Best of User Videos.

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

1.3 Online resources

More information and help available at these URLs :

- Code repository.
- Documentation.
- API documentation.
- Code coverage.
- Jenkins CI server.
- Discussions and help at Google Group.
- For reporting a bug use Github Issues.

CHAPTER

TWO

GETTING STARTED

2.1 Installation

2.1.1 Dependencies

Make sure to install these packages prior to installation :

- Python 2.x >= 2.5
- Django >= 1.4
- django-mptt >= 0.5.1 < 0.6
- django-tagging >= 0.3.1
- BeautifulSoup >= 3.2.0

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

- pytz
- pyparsing >= 1.5.5
- django-xmlrpc >= 0.1.3

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

2.1.2 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
```

2.1.3 Applications

Then 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.comments',
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.contenttypes',
  'tagging',
  'mptt',
  'zinnia',
)
```

2.1.4 Template Context Processors

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

```
TEMPLATE_CONTEXT_PROCESSORS = (
  'django.contrib.auth.context_processors.auth',
  'django.core.context_processors.i18n',
  'django.core.context_processors.request',
  'django.core.context_processors.media',
  'django.core.context_processors.static',
  'zinnia.context_processors.version',) # Optional
```

2.1.5 URLs

Add the following lines to your project's urls.py in order to display the blog.

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

Note that the default zinnia URLset is provided for convenient usage, but you can customize your URLs if you want. Here's how:

```
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'^weblog/tags/', include('zinnia.urls.tags')),
url(r'^weblog/feeds/', include('zinnia.urls.feeds')),
url(r'^weblog/authors/', include('zinnia.urls.authors')),
url(r'^weblog/categories/', include('zinnia.urls.categories')),
url(r'^weblog/discussions/', include('zinnia.urls.discussions')),
url(r'^weblog/', include('zinnia.urls.entries')),
url(r'^weblog/', include('zinnia.urls.archives')),
url(r'^weblog/', include('zinnia.urls.shortlink')),
url(r'^weblog/', include('zinnia.urls.quick_entry')),
url(r'^comments/', include('django.contrib.comments.urls')),
```

2.1.6 Static Files

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

2.1.7 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 syncdb
```

If you are using South to manage your database, you will have to do the following.

```
$ python manage.py syncdb --migrate
```

2.2 Advanced Configuration

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

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

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

2.2.2 Akismet Anti-Spam

If you want to benefit of the Akismet spam protection on your comments, it's possible to do it by installing the akismet Python module, and add this setting:

```
ZINNIA_SPAM_CHECKER_BACKENDS = ('zinnia.spam_checker.backends.automattic',)
```

Important: You need an API key. If you don't have any, get one for free at http://akismet.com/signup/ then set it in your project's settings like this:

```
AKISMET_SECRET_API_KEY = 'your key'
```

2.2.3 TypePad Anti-Spam

It's also possible to benefit of the TypePad AntiSpam service to fight the spam. Like the Akismet protection you need to install the akismet Python module.

Then register the TypePad AntiSpam protection with this setting:

```
ZINNIA_SPAM_CHECKER_BACKENDS = ('zinnia.spam_checker.backends.typepad',)
```

Important: You need an API key. If you don't have any, get one for free at http://antispam.typepad.com/info/get-api-key.html then set it in your project's settings like this:

```
TYPEPAD_SECRET_API_KEY = 'your key'
```

2.2.4 Mollom Anti-Spam

Another approach to fight the spam is provided by Mollom, Zinnia implement a backend to use this spam filtering service. Before configuring the service, you need to install the PyMollom Python library and then register the Mollom spam checking protection with this setting:

```
ZINNIA_SPAM_CHECKER_BACKENDS = ('zinnia.spam_checker.backends.mollom',)
```

Important: You need a private and public keys to use this service. Get a free account at http://mollom.com/pricing then set your keys in your project's settings like this:

```
MOLLOM_PUBLIC_KEY = 'your public key'
MOLLOM_PRIVATE_KEY = 'your private key'
```

2.2.5 Bit.ly

You find http://bit.ly useful and want to use it for your blog entries ?

It's simple, install django-bitly in your project's settings and add these settings:

```
BITLY_LOGIN = 'your bit.ly login'
BITLY_API_KEY = 'your bit.ly api key'
ZINNIA_URL_SHORTENER_BACKEND = 'zinnia.url_shortener.backends.bitly'
```

Zinnia will do the rest.

2.2.6 Twitter

When you post a new entry on your blog you might want to tweet it as well.

In order to do that, you first need to activate the *Bit.ly* support like described above.

Then install tweepy and add these settings.

```
TWITTER_CONSUMER_KEY = 'Your Consumer Key'
TWITTER_CONSUMER_SECRET = 'Your Consumer Secret'
TWITTER_ACCESS_KEY = 'Your Access Key'
TWITTER_ACCESS_SECRET = 'Your Access Secret'
```

Note that the authentification for Twitter has changed since September 2010. The actual authentification system is based on oAuth. That's why now you need to set these 4 settings. If you don't know how to get these information, follow this excellent tutorial at:

http://jmillerinc.com/2010/05/31/twitter-from-the-command-line-in-python-using-oauth/

Now in the admin, you can post an update containing your entry's title and the shortened URL of your entry.

2.2.7 Django-CMS

If you use django-CMS 2.0, Zinnia can be integrated into your pages, thanks to the plugin system.

Warning: Changed in version 0.10.1. `zinnia.plugins` has been removed in favor of cmsplugin_zinnia.

Simply refer to cmsplugin_zinnia's documentation for more information about the install instructions and possibilities.

2.2.8 TinyMCE

If you want to replace WYMEditor by TinyMCE install django-tinymce and follow the installation instructions.

TinyMCE can be customized by overriding the `admin/zinnia/entry/tinymce_textareas.js` template.

2.2.9 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, simply set this variable as appropriate in your project's settings.

```
ZINNIA_MARKUP_LANGUAGE = 'restructuredtext'
```

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

More informations about the dependencies in `django.contrib.markup`.

2.2.10 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.

Now add these lines in your project's settings.

```
from zinnia.xmlrpc import ZINNIA_XMLRPC_METHODS
XMLRPC_METHODS = ZINNIA_XMLRPC_METHODS
```

`ZINNIA_XMLRPC_METHODS` is a simple list of tuples containing all the Webservices embedded in Zinnia.

If you only want to use the Pingback service import `ZINNIA_XMLRPC_PINGBACK`, or if you want you just want to enable the MetaWeblog API import `ZINNIA_XMLRPC_METAWEBLOG`.

You can also use your own mixins.

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

2.3 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.

2.3.1 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
```

2.3.2 Preparing the database

The main problem with the upgrade process is the database. The Zinnia's models can have changed with new or missing fields. That's why Zinnia use South's migrations to facilitate this step.

So we need to install the South package.

```
$ easy_install south
```

South needs to be registered in your project's settings as an `INSTALLED_APPS`. Once it is done, use syncdb to finish the installation of South in your project.

```
$ python manage.py syncdb
```

Now we will install the previous migrations of Zinnia to synchronize the current database schema with South.

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

2.3.3 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
```

2.3.4 Update the database

The database should probably be updated to the latest database schema of Zinnia, South will be useful.

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

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

2.3.5 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

THREE

TOPICS

3.1 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')),
```

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`.

3.2 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.

3.2.1 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`.

3.2.2 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.

The list of stop words is stored in the `ZINNIA_STOP_WORDS` setting.

3.3 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 *django-bitly*.

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

First of all Zinnia now provides his own short URLs for the entries, example:

> http://mydomain.com/blog/1/

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/1/ (40 characters !)

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

3.3.1 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/`.

3.4 Spam Checker

New in version 0.9. Spam protection is mandatory when you want to let your users to comment 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 moderation your comments.

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

3.4.1 Built-in spam checkers

- `zinnia.spam_checker.backends.all_is_spam`
- `zinnia.spam_checker.backends.automattic`
- `zinnia.spam_checker.backends.long_enough`
- `zinnia.spam_checker.backends.mollom`
- `zinnia.spam_checker.backends.typepad`

3.4.2 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` otherwise.
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/`.

3.5 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.

3.5.1 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

3.5.2 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.

3.5.3 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.

CHAPTER

FOUR

HOW-TOS

4.1 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 two applications which aim to turn Zinnia's templates HTML5 ready, which can be a good starting point to make your own at:

- 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.

4.1.1 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 redondant 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.

4.1.2 Compass and Sass

If you take a look at `zinnia/static/zinnia/css/screen.css` you will probably notice that the CSS is compressed. It has been generated by Compass and it is good pratice not to edit this file directly.

All the CSS documents are actually generated by the awesome Compass tool and you must take a look on this video tutorial if your are not familiar with it.

Compass is an open-source CSS authoring framework which uses the Sass stylesheet language to make writing stylesheets powerful and easy.

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

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

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. `screen.scss` will include all these files into a single CSS document.

This organization allow you to easily customize the default Zinnia's CSS by doing a copy of these files or reuse some parts of the Zinnia's CSS into third-party templates.

4.1.3 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/auhtor/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/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/my-entry/` the `EntryDetail` view will be called and return this list of template names:

```
['zinnia/archives/2012/04/21/entry_detail.html',
 'zinnia/archives/month/04/day/21/entry_detail.html',
 'zinnia/archives/2012/day/21/entry_detail.html',
 'zinnia/archives/day/21/entry_detail.html',
 'zinnia/archives/2012/month/04/entry_detail.html',
 'zinnia/archives/month/04/entry_detail.html',
 'zinnia/archives/2012/entry_detail.html',
 'zinnia/archives/entry_detail.html',
 'zinnia/entry_detail.html',
 'entry_detail.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_TEMPLATES settings, so with this option you can handle multiple presentation of your entries.

But if you have followed the latest example about the archives templates, you must have seen that even the EntryDetail view implement a custom list of templates, because this view is based on the archives.

So 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/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/custom.html',
 'zinnia/archives/month/04/custom.html',
 'zinnia/archives/2012/custom.html',
 'zinnia/archives/custom.html',
 'zinnia/custom.html',
 'custom.html']
```

4.1.4 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='zinnia_entry_search'),
    )
```

4.1.5 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. You can take a look at Zinnia-theme-html5 to have a good starting point of a packaged theme.

4.2 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.

4.2.1 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 class containing your customizations.
2. Register your class into Zinnia to be used.
3. Update the `EntryAdmin` class accordingly.

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

4.2.2 Writing model extension

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

```python
from django.db import models
from zinnia_gallery.models import Gallery
from zinnia.models import EntryAbstractClass

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

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

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

In this code sample, we add a new `ForeignKey` field named `gallery` pointing to a `Gallery` model defined in `zinnia_gallery.models` and we override the `EntryAbstractClass.__unicode__()` method.

Note: You have to respect **3 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. Do not put your abstract model in a file named `models.py`, it will not work for a non obvious reason.
3. 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.

Considerations about the database

If you do the extension of the `Entry` model after the `syncdb` command, you have to manually alter the Zinnia's tables for reflecting your changes made on the model class. In the case where your database is empty, you can simply execute the `reset` command on the Zinnia application for destroying the old database schema and installing the new one.

Now if you are using South and try to write a new migration for reflecting your changes, 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 South provides an elegant solution with the SOUTH_MIGRATION_MODULES setting. Once this setting done for the `'zinnia'` key, because you are now out the Zinnia's default migrations flow, you have to delete the ghost migrations for Zinnia. At this step you can now start to write new migrations.

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 can alter your database schema with the `migrate` command.

4.2.3 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.entry_gallery.EntryGallery'
```

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

4.2.4 Updating the admin interface

Now we should update the `Entry`'s 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 use the admin site register/unregister mechanism for using our new class.

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 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', 'content', 'image', 'status', 'gallery')}),) + \
    EntryAdmin.fieldsets[1:]

# Unregister the default EntryAdmin
# then register the EntryGalleryAdmin class
admin.site.unregister(Entry)
admin.site.register(Entry, EntryGalleryAdmin)
```

Note that the `zinnia_gallery` application must be registered in the `INSTALLED_APPS` setting after the `zinnia` application for applying the register/unregister mechanism in the admin site.

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

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

4.3 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.

4.3.1 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.

4.3.2 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 `EntryAbstractClass` with the new `get_absolute_url` method.

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

    @models.permalink
    def get_absolute_url(self):
        return ('zinnia_entry_detail', (),
                {'pk': self.id})

    class Meta(EntryAbstractClass.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.

4.3.3 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 import Entry
from zinnia.views.mixins.entry_protection import EntryProtectionMixin

class EntryDetail(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.

4.3.4 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')),
...
```

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='zinnia_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.

4.4 Import / Export

If you already have a blog, Zinnia has the ability to import your posts from other blogging platforms. Useful for rapid migration.

4.4.1 From WordPress to Zinnia

Zinnia provides a command for importing export files from WordPress.

http://codex.wordpress.org/Tools_Export_SubPanel

Once you have the XML file, you simply have to do this.

```
$ python manage.py wp2zinnia path/to/your/wordpress.xml
```

This command will associate the post's authors to User and import the tags, categories, post and comments.

For the options execute this.

```
$ python manage.py help wp2zinnia
```

4.4.2 From Zinnia to WordPress

Zinnia also provides a command for exporting your blog to WordPress in the case you want to migrate on it.

Simply execute this command:

```
$ python manage.py zinnia2wp > export.xml
```

Once you have the XML export, you can import it into your WordPress site.

http://codex.wordpress.org/Importing_Content

4.4.3 From Blogger to Zinnia

If you are comming from Blogger, you can import your posts and comments with this simple command:

```
$ python manage.py blogger2zinnia
```

For the options execute this.

```
$ python manage.py help blogger2zinnia
```

Note that you need to install the gdata package to run the importation.

4.4.4 From Feed to Zinnia

If you don't have the possibility to export your posts but have a RSS or Atom feed on your Weblog, Zinnia can import it. This command is the most generic way to import content into Zinnia. Simply execute this command:

```
$ python manage.py feed2zinnia http://url.of/the/feed
```

For the options execute this.

```
$ python manage.py help feed2zinnia
```

Note that you need to install the feedparser package to run the importation.

CHAPTER

FIVE

DEVELOPMENT

5.1 Contributing to Zinnia

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

5.1.1 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 the code on Github.
2. Clone a local copy of your fork.
3. Write tests.
4. Develop your code.
5. Test your new code.
6. Update the documentation if needed.
7. Commit and push your changes.
8. Open a pull request.

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.

5.1.2 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 :

- Use **Sphinx** and **restructuredText**.
- Use **.rst** as file extension.
- Be written in English.
- 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.

5.1.3 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.

5.2 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.

5.2.1 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
```

5.2.2 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 cloisonate 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
$ virtualenv --no-site-packages django-blog-zinnia
$ cd django-blog-zinnia
$ source ./bin/activate
$ python bootstrap.py
$ ./bin/buildout
```

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

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

5.2.3 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/pyflakes zinnia
$ ./bin/pep8 --count -r --exclude=tests.py,migrations zinnia
```

For building the HTML documentation run this simple command:

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

5.2.4 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 syncdb
$ ./bin/demo runserver
```

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

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

Pretty easy no ?

5.3 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/`.

5.3.1 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.

If you want to make some speed optimizations or compare with your tests results, you can check the actual execution time of the tests at this URL:

http://django-blog-zinnia.com/documentation/xunit/

5.3.2 Coverage

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

As I write these lines the **143** tests in Zinnia cover **96%** 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.

You can check the actual coverage percent at this URL:

http://django-blog-zinnia.com/documentation/coverage/

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

CHAPTER

SIX

REFERENCES

6.1 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`.

6.1.1 Entry

ZINNIA_ENTRY_TEMPLATES

Default value: `()` (Empty tuple)

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

```
ZINNIA_ENTRY_TEMPLATES = (('entry_detail_alternate.html',
                           gettext('Alternative template')),)
```

ZINNIA_ENTRY_BASE_MODEL

Default value: `''` (Empty string)

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

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'`.

6.1.2 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']
```

The value of this variable will alter the value of `ZINNIA_WYSIWYG` if you don't set it.

ZINNIA_MARKDOWN_EXTENSIONS

Default value: `''` (Empty string)

Extensions names to be used for rendering the entries in MarkDown. Example:

```
ZINNIA_MARKDOWN_EXTENSIONS = 'extension1_name,extension2_name...'
```

ZINNIA_WYSIWYG

Default value:

```
WYSIWYG_MARKUP_MAPPING = {
    'textile': 'markitup',
    'markdown': 'markitup',
    'restructuredtext': 'markitup',
    'html': 'tinymce' in settings.INSTALLED_APPS and \
                'tinymce' or 'wymeditor'}

WYSIWYG = getattr(settings, 'ZINNIA_WYSIWYG',
                  WYSIWYG_MARKUP_MAPPING.get(ZINNIA_MARKUP_LANGUAGE))
```

Determining the WYSIWYG editor used for editing an entry. So if MarkDown, Textile or reStructuredText are used, the value will be `'markitup'`, but if you use HTML, TinyMCE will be used if *django-tinymce is installed*, else WYMEditor will be used.

This setting can also be used for disabling the WYSIWYG functionnality. Example:

```
ZINNIA_WYSIWYG = None
```

6.1.3 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.

6.1.4 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.

6.1.5 URLs

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`.

6.1.6 Comment moderation

ZINNIA_AUTO_MODERATE_COMMENTS

Default value: `False`

Determine if a new comment should be allowed to show up immediately or should be marked non-public and await approval.

ZINNIA_AUTO_CLOSE_COMMENTS_AFTER

Default value: `None`

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.

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 tuple)

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`.

6.1.7 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.

6.1.8 Similarity

ZINNIA_F_MIN

Default value: `0.1`

Float setting of the minimal word frequency for similar entries.

ZINNIA_F_MAX

Default value: `1.0`

Float setting of the minimal word frequency for similar entries.

6.1.9 Miscellaneous

ZINNIA_COPYRIGHT

Default value: `'Zinnia'`

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

ZINNIA_STOP_WORDS

Default value: See `zinnia/settings.py`

List of common words excluded from the advanced search engine to optimize the search querying and the results.

ZINNIA_USE_TWITTER

Default value: `True if python-twitter is in the PYTHONPATH`

Boolean telling if Zinnia can use Twitter.

6.2 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_tags %}
```

6.2.1 get_recent_entries

Display the latest entries.

`zinnia.templatetags.zinnia_tags.`**`get_recent_entries`**(*number=5*, *template='zinnia/tags/recent_entries.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" %}
```

6.2.2 get_featured_entries

Display the featured entries.

`zinnia.templatetags.zinnia_tags.`**`get_featured_entries`**(*number=5*, *template='zinnia/tags/featured_entries.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" %}
```

6.2.3 get_draft_entries

Display the latest entries marked as draft.

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

Return the latest 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" %}
```

6.2.4 get_random_entries

Display random entries.

`zinnia.templatetags.zinnia_tags.`**`get_random_entries`**(*number=5, template='zinnia/tags/random_entries.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" %}
```

6.2.5 get_popular_entries

Display popular entries.

`zinnia.templatetags.zinnia_tags.`**`get_popular_entries`**(*number=5, template='zinnia/tags/popular_entries.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" %}
```

6.2.6 get_similar_entries

Display entries similar to an existing entry.

`zinnia.templatetags.zinnia_tags.`**`get_similar_entries`**(*context, number=5, template='zinnia/tags/similar_entries.html', flush=False*)

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" %}
```

6.2.7 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_tags.`**`get_calendar_entries`**`(`*context*, *year=None*, *month=None*, *template='zinnia/tags/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 %}
```

6.2.8 get_archives_entries

Display the archives by month.

`zinnia.templatetags.zinnia_tags.`**`get_archives_entries`**`(`*template='zinnia/tags/archives_entries.html'*`)`
Return archives entries

Usage examples:

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

6.2.9 get_archives_entries_tree

Display all the archives as a tree.

`zinnia.templatetags.zinnia_tags.`**`get_archives_entries_tree`**`(`*template='zinnia/tags/archives_entries_tree.htm*
Return archives entries as a Tree

Usage examples:

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

6.2.10 get_authors

Display all the published authors.

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

Usage examples:

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

6.2.11 get_categories

Display all the categories available.

`zinnia.templatetags.zinnia_tags.`**`get_categories`**(*template='zinnia/tags/categories.html'*)

Return the categories

Usage examples:

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

6.2.12 get_tags

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

`zinnia.templatetags.zinnia_tags.`**`get_tags`**()

Return the published tags

Usage example:

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

6.2.13 get_tag_cloud

Display a cloud of published tags.

`zinnia.templatetags.zinnia_tags.`**`get_tag_cloud`**(*steps=6, 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 "custom_template.html" %}
{% get_tag_cloud template="custom_template.html" %}
```

6.2.14 get_recent_comments

Display the latest comments.

`zinnia.templatetags.zinnia_tags.`**`get_recent_comments`**(*number=5, template='zinnia/tags/recent_comments.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" %}
```

6.2.15 get_recent_linkbacks

Display the latest linkbacks.

`zinnia.templatetags.zinnia_tags.`**`get_recent_linkbacks`**(*number=5*, *template='zinnia/tags/recent_linkbacks.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" %}
```

6.2.16 zinnia_pagination

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

`zinnia.templatetags.zinnia_tags.`**`zinnia_pagination`**(*context*, *page*, *begin_pages=3*, *end_pages=3*, *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" %}
```

6.2.17 zinnia_breadcrumbs

Display the breadcrumbs for the pages handled by Zinnia.

`zinnia.templatetags.zinnia_tags.`**`zinnia_breadcrumbs`**(*context*, *root_name='Blog'*, *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" %}
```

6.2.18 zinnia_statistics

Display the statistics about the contents handled in Zinnia.

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

Return statistics on the content of Zinnia

Usage examples:

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

6.2.19 get_gravatar

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

`zinnia.templatetags.zinnia_tags.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" %}
```

6.3 Zinnia API

Contents

6.3.1 `zinnia` Package

Zinnia

6.3.2 `comparison` Module

Comparison tools for Zinnia Based on clustered_models app

class `zinnia.comparison.ClusteredModel`(*queryset*, *fields=['id']*)

Bases: `object`

Wrapper around Model class building a dataset of instances

dataset()
Generate a dataset with the queryset and specified fields

class zinnia.comparison.**VectorBuilder**(*queryset*, *fields*)
Bases: object

Build a list of vectors based on datasets

build_dataset()
Generate whole dataset

flush()
Flush the dataset

generate_key()
Generate key for this list of vectors

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

6.3.3 context_processors Module

Context Processors for Zinnia

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

6.3.4 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_author_link(*item*)
Author's URL of the discussion

item_author_name(*item*)
Author of the discussion

item_link(*item*)
URL of the discussion

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 in 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*)
Returns 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*)
Returns the first author's email

item_author_link(*item*)
Returns the author's URL

item_author_name(*item*)
Returns the first author of an entry

item_categories(*item*)
Entry's categories

item_enclosure_length(*item*)
Hardcoded enclosure length

item_enclosure_mime_type(*item*)
Hardcoded enclosure mimetype

item_enclosure_url(*item*)
Returns an image for enclosure

item_pubdate(*item*)
Publication date of an entry

title_template = 'feeds/entry_title.html'

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

Feed for pingbacks in 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 in 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.**LatestDiscussions**
: Bases: zinnia.feeds.DiscussionFeed

  Feed for the latest discussions

  description()
  : Description of the feed

  get_title(*obj*)
  : Title of the feed

  items()
  : Items are the discussions on the entries

  link()
  : URL of latest discussions

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

  Feed for the latest entries

  description()
  : Description of the feed

  get_title(*obj*)
  : Title of the feed

  items()
  : Items are published entries

  link()
  : URL of latest 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*, *slug*)
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'

get_title(*obj*)

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

6.3.5 managers Module

Managers of Zinnia

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

Manager to retrieve published authors

get_query_set()
Return published authors

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_query_set ()
Return published entries

on_site ()
Return entries published on current site

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

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

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

6.3.6 `models` Module

Models of Zinnia

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

Proxy Model around User

Parameters

- **id** (*AutoField*) – Id
- **username** (*CharField*) – Required. 30 characters or fewer. Letters, numbers and @/./+/-/_ characters
- **first_name** (*CharField*) – First name
- **last_name** (*CharField*) – Last name
- **email** (*EmailField*) – E-mail address
- **password** (*CharField*) – Password
- **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.
- **is_superuser** (*BooleanField*) – Designates that this user has all permissions without explicitly assigning them.
- **last_login** (*DateTimeField*) – Last login
- **date_joined** (*DateTimeField*) – Date joined

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

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

Author.**entries_published** ()
Return only the entries published

Author.**get_absolute_url**(**moreargs*, ***morekwargs*)
Return author's URL

Author.**objects = <django.db.models.manager.Manager object at 0x221e310>**

Author.**published = <zinnia.managers.AuthorPublishedManager object at 0x221e350>**

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

Category object for Entry

Parameters

- **id** (*AutoField*) – Id
- **title** (*CharField*) – Title
- **slug** (*SlugField*) – used for publication
- **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**

Category.**entries**

Category.**entries_published**()
Return only the entries published

Category.**get_absolute_url**(**moreargs*, ***morekwargs*)
Return category's URL

Category.**objects = <mptt.managers.TreeManager object at 0x2219090>**

Category.**parent**

Category.**tree = <mptt.managers.TreeManager object at 0x2219090>**

Category.**tree_path**
Return category's tree path, by his ancestors

class zinnia.models.**Entry**(**args*, ***kwargs*)
Bases: `zinnia.models.EntryAbstractClass`

The final Entry model based on inheritence. Check this out for customizing the Entry Model class: http://django-blog-zinnia.com/documentation/how-to/extending_entry_model/

Parameters

- **id** (*AutoField*) – Id
- **title** (*CharField*) – Title

- **image** (*ImageField*) – used for illustration
- **content** (*TextField*) – Content
- **excerpt** (*TextField*) – optional element
- **tags** (*TagField*) – Tags
- **slug** (*SlugField*) – used for publication
- **status** (*IntegerField*) – Status
- **featured** (*BooleanField*) – Featured
- **comment_enabled** (*BooleanField*) – Comment enabled
- **pingback_enabled** (*BooleanField*) – Linkback enabled
- **creation_date** (*DateTimeField*) – Creation date
- **last_update** (*DateTimeField*) – Last update
- **start_publication** (*DateTimeField*) – date start publish
- **end_publication** (*DateTimeField*) – date end publish
- **login_required** (*BooleanField*) – only authenticated users can view the entry
- **password** (*CharField*) – protect the entry with a password
- **template** (*CharField*) – template used to display the entry

exception `DoesNotExist`
Bases: `django.core.exceptions.ObjectDoesNotExist`

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

`Entry.authors`

`Entry.categories`

`Entry.get_absolute_url`(**moreargs*, ***morekwargs*)
Return entry's URL

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

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

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

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

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

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

`Entry.objects` **= <django.db.models.manager.Manager object at 0x222cc10>**

`Entry.published` **= <zinnia.managers.EntryPublishedManager object at 0x222cc90>**

`Entry.related`

`Entry.sites`

class `zinnia.models.EntryAbstractClass`(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Base Model design for publishing entries

Parameters

- **title** (*CharField*) – Title
- **image** (*ImageField*) – used for illustration
- **content** (*TextField*) – Content
- **excerpt** (*TextField*) – optional element
- **tags** (*TagField*) – Tags
- **slug** (*SlugField*) – used for publication
- **status** (*IntegerField*) – Status
- **featured** (*BooleanField*) – Featured
- **comment_enabled** (*BooleanField*) – Comment enabled
- **pingback_enabled** (*BooleanField*) – Linkback enabled
- **creation_date** (*DateTimeField*) – Creation date
- **last_update** (*DateTimeField*) – Last update
- **start_publication** (*DateTimeField*) – date start publish
- **end_publication** (*DateTimeField*) – date end publish
- **login_required** (*BooleanField*) – only authenticated users can view the entry
- **password** (*CharField*) – protect the entry with a password
- **template** (*CharField*) – template used to display the entry

class **Meta**

Entry's Meta

abstract = False

get_latest_by = 'creation_date'

ordering = ['-creation_date']

permissions = (('can_view_all', 'Can view all entries'), ('can_change_status', 'Can change status'), ('can_change_

verbose_name = <django.utils.functional.__proxy__ object at 0x2223410>

verbose_name_plural = <django.utils.functional.__proxy__ object at 0x2223450>

EntryAbstractClass.**STATUS_CHOICES = ((0, <django.utils.functional.__proxy__ object at 0x221eb10>), (1, <djang**

EntryAbstractClass.**authors**

EntryAbstractClass.**categories**

EntryAbstractClass.**comments**

Return published comments

EntryAbstractClass.**comments_are_open**

Check if comments are open

EntryAbstractClass.**discussions**

Return published discussions

EntryAbstractClass.**get_absolute_url**(**args*, ***kwargs*)

Return entry's URL

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

EntryAbstractClass.**get_next_by_last_update**(**moreargs*, ***morekwargs*)

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

EntryAbstractClass.**get_previous_by_last_update**(**moreargs*, ***morekwargs*)

EntryAbstractClass.**get_status_display**(**moreargs*, ***morekwargs*)

EntryAbstractClass.**get_template_display**(**moreargs*, ***morekwargs*)

EntryAbstractClass.**html_content**
Return the Entry.content attribute formatted in HTML

EntryAbstractClass.**is_actual**
Check if an entry is within publication period

EntryAbstractClass.**is_visible**
Check if an entry is visible on site

EntryAbstractClass.**next_entry**
Return the next entry

EntryAbstractClass.**objects = <django.db.models.manager.Manager object at 0x2223390>**

EntryAbstractClass.**pingbacks**
Return published pingbacks

EntryAbstractClass.**previous_entry**
Return the previous entry

EntryAbstractClass.**published = <zinnia.managers.EntryPublishedManager object at 0x22233d0>**

EntryAbstractClass.**related**

EntryAbstractClass.**related_published**
Return only related entries published

EntryAbstractClass.**short_url**
Return the entry's short url

EntryAbstractClass.**sites**

EntryAbstractClass.**trackbacks**
Return published trackbacks

EntryAbstractClass.**word_count**
Count the words of an entry

zinnia.models.**get_base_model**()
Determine the base Model to inherit in the Entry Model, this allow to overload it.

6.3.7 moderator Module

Moderator of Zinnia comments

class zinnia.moderator.**EntryCommentModerator**(*model*)
Bases: django.contrib.comments.moderation.CommentModerator

Moderate the comment of Entry

auto_close_field = 'start_publication'

auto_moderate_comments = False

close_after = None

`do_email_authors` (*comment*, *content_object*, *request*)
Send email notification of a new comment to the authors of the entry when email notifications have been requested.

`do_email_notification` (*comment*, *content_object*, *request*)
Send email notification of a new comment to site staff when email notifications have been requested.

`do_email_reply` (*comment*, *content_object*, *request*)
Send email notification of a new comment to the authors of the previous comments when email notifications have been requested.

`email` (*comment*, *content_object*, *request*)

`email_authors` = **True**

`email_reply` = **False**

`enable_field` = **'comment_enabled'**

`mail_comment_notification_recipients` = **[]**

`moderate` (*comment*, *content_object*, *request*)
Determine whether a given comment on a given object should be allowed to show up immediately, or should be marked non-public and await approval.

`spam_checker_backends` = **()**

6.3.8 `ping` Module

Pings utilities for Zinnia

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

Threaded 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*, *start_now=True*)
Bases: `threading.Thread`

Threaded ExternalUrls Pinger

`find_external_urls` (*entry*)
Find external urls in an entry

`find_pingback_href` (*content*)
Try to find Link markup to pingback url

`find_pingback_urls` (*urls*)
Find the pingback urls of each urls

`is_external_url` (*url*, *site_url*)
Check of the url in 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

6.3.9 `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.`**`createQ`**(*token*)

Creates the Q() object

`zinnia.search.`**`unionQ`**(*token*)

Appends all the Q() objects

6.3.10 `signals` Module

Signal handlers of Zinnia

`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_zinnia_signals`**()

Disconnect all the signals provided by Zinnia

`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

6.3.11 `sitemaps` Module

Sitemaps for Zinnia

class `zinnia.sitemaps.`**`AuthorSitemap`**

Bases: `zinnia.sitemaps.ZinniaSitemap`

Sitemap for authors

`changefreq` **= 'monthly'**

`items`()

Return published authors

`lastmod`(*obj*)

Return last modification of an author

`location`(*obj*)

Return url of an author

`priority` **= 0.5**

class `zinnia.sitemaps.`**`CategorySitemap`**
Bases: `zinnia.sitemaps.ZinniaSitemap`

Sitemap for categories

`cache`(*categories*)
Cache categorie's entries percent on total entries

`changefreq` = **'monthly'**

`items`()
Return all categories with coeff

`lastmod`(*obj*)
Return last modification of a category

`priority`(*obj*)
Compute priority with cached coeffs

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.ZinniaSitemap`

Sitemap for tags

`cache`(*tags*)
Cache tag's entries percent on total entries

`changefreq` = **'monthly'**

`items`()
Return all tags with coeff

`lastmod`(*obj*)
Return last modification of a tag

`location`(*obj*)
Return url of a tag

`priority`(*obj*)
Compute priority with cached coeffs

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

Base Sitemap class for Zinnia

`protocol` = **'http'**

6.3.12 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` = **['make_mine', 'make_published', 'make_hidden', 'close_comments', 'close_pingbacks', 'ping_directories', 'm**

`actions_on_bottom` = **True**

`actions_on_top` = **True**

`autocomplete_tags`(*request*)

View for tag autocompletion

`close_comments`(*request*, *queryset*)

Close the comments for selected entries

`close_pingbacks`(*request*, *queryset*)

Close the pingbacks for selected entries

`content_preview`(**args*, ***kwargs*)

Admin view to preview Entry.content in HTML, useful when using markups to write entries

`date_hierarchy` = **'creation_date'**

`fieldsets` = **((<django.utils.functional.__proxy__ object at 0x3b5dd90>, {'fields': ('title', 'content', 'image', 'status')}),**

`filter_horizontal` = **('categories', 'authors', 'related')**

`form`
alias of `EntryAdminForm`

`formfield_for_manytomany`(*db_field*, *request*, ***kwargs*)
Filters the disposable authors

`get_actions`(*request*)
Define user actions by permissions

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

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

`get_comments_are_open`(*entry*)
Admin wrapper for entry.comments_are_open

`get_is_actual`(*entry*)
Admin wrapper for entry.is_actual

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

`get_link`(*entry*)
Return a formated link to the entry

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

`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

`get_urls`()

`list_display` = **('get_title', 'get_authors', 'get_categories', 'get_tags', 'get_sites', 'get_comments_are_open', 'pingbacl**

`list_filter` = **('categories', <class 'zinnia.admin.filters.AuthorListFilter'>, 'status', 'featured', 'login_required', 'com**

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

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

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

`make_tweet`(*request*, *queryset*)
Post an update on Twitter

markitup(*request*)
View for serving the config of MarkItUp

media

ping_directories(*request*, *queryset*, *messages=True*)
Ping Directories for selected entries

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

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

queryset(*request*)
Make special filtering by user permissions

radio_fields = {'template': 2}

save_model(*request*, *entry*, *form*, *change*)
Save the authors, update time, make an excerpt

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

wymeditor(*request*)
View for serving the config of WYMEditor

`filters` Module

Filters for Zinnia admin

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

List filter for EntryAdmin filtering possibilities to published authors only.

lookups(*request*, *model_admin*)
Return published authors choices with their entries number

parameter_name = 'author'

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

title = <django.utils.functional.__proxy__ object at 0x37b9f90>

`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

model
alias of `Category`

`CategoryAdminForm.`**admin_site = <django.contrib.admin.sites.AdminSite object at 0x20f3210>**

`CategoryAdminForm.`**base_fields = {'title': <django.forms.fields.CharField object at 0x381b210>, 'slug': <django.**

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

CategoryAdminForm.**declared_fields** = {'parent': <zinnia.admin.widgets.TreeNodeChoiceField object at 0x37b9

CategoryAdminForm.**media**

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

Form for Entry's Admin

class Meta
EntryAdminForm's Meta

model
alias of Entry

EntryAdminForm.**admin_site** = <django.contrib.admin.sites.AdminSite object at 0x20f3210>

EntryAdminForm.**base_fields** = {'title': <django.forms.fields.CharField object at 0x3b5dc50>, 'image': <django.fo

EntryAdminForm.**declared_fields** = {'categories': <zinnia.admin.widgets.MPTTModelMultipleChoiceField objec

EntryAdminForm.**media**

widgets Module

Widgets for Zinnia admin

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

MPTT version of FilteredSelectMultiple

class Media
MPTTFilteredSelectMultiple's Media

js = ('/static/admin/js/core.js', '/static/zinnia/js/mptt_m2m_selectbox.js', '/static/admin/js/SelectFilter2.js')

MPTTFilteredSelectMultiple.**media**

MPTTFilteredSelectMultiple.**render_options**(*choices*, *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.**MPTTModelChoiceIterator**(*field*)
Bases: django.forms.models.ModelChoiceIterator

MPTT version of ModelChoiceIterator

choice(*obj*)
Overriding choice method

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

MPTT version of ModelMultipleChoiceField

choices
Overriding _get_choices

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

class zinnia.admin.widgets.**TreeNodeChoiceField**(*level_indicator=u'|–'*, **args*, ***kwargs*)
Bases: django.forms.models.ModelChoiceField

Duplicating the TreeNodeChoiceField bundled in django-mptt to avoid conflict with the TreeNodeChoiceField bundled in django-cms...

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

spam_checker Package

spam_checker Package

Spam checker for Zinnia

zinnia.spam_checker.**check_is_spam**(*content*, *content_object*, *request*, *backends=()*)
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**(*comment*, *content_object*, *request*)
Backend for setting all comments to spam

automattic Module Akismet spam checker backend for Zinnia

zinnia.spam_checker.backends.automattic.**backend**(*comment*, *content_object*, *request*)
Akismet spam checker backend for Zinnia

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/

`mollom` Module Mollom spam checker backend for Zinnia

`zinnia.spam_checker.backends.mollom.`**`backend`**(*comment*, *content_object*, *request*)
Mollom spam checker backend for Zinnia

`typepad` Module TypePad spam checker backend for Zinnia

class `zinnia.spam_checker.backends.typepad.`**`TypePad`**(*key=None*, *blog_url=None*, *agent=None*)
Bases: `akismet.Akismet`

TypePad version of the Akismet module

`baseurl` = 'api.antispam.typepad.com/1.1/'

`zinnia.spam_checker.backends.typepad.`**`backend`**(*comment*, *content_object*, *request*)
TypePad spam checker backend for Zinnia

templatetags Package

`templatetags` Package

Templatetags for Zinnia

`zbreadcrumbs` Module

Breadcrumb module for Zinnia templatetags

class `zinnia.templatetags.zbreadcrumbs.`**`Crumb`**(*name*, *url=None*)
Bases: `object`

Part of the Breadcrumbs

`zinnia.templatetags.zbreadcrumbs.`**`ZINNIA_ROOT_URL`**()

`zinnia.templatetags.zbreadcrumbs.`**`day_crumb`**(*creation_date*)
Crumb for a day

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

`zinnia.templatetags.zbreadcrumbs.`**`month_crumb`**(*creation_date*)
Crumb for a month

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

`zinnia.templatetags.zbreadcrumbs.`**`year_crumb`**(*creation_date*)
Crumb for a year

`zcalendar` Module

Calendar module for Zinnia templatetags

class `zinnia.templatetags.zcalendar.`**`ZinniaCalendar`**
Bases: `calendar.HTMLCalendar`

Override of 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.

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 Shortlink backends for Zinnia

`bitly` Module Bit.ly url shortener backend for Zinnia

`zinnia.url_shortener.backends.bitly.`**`backend`**(*entry*)
Bit.ly url shortener backend for Zinnia

`default` Module Default url shortener backend for Zinnia

`zinnia.url_shortener.backends.default.`**`backend`**(*entry*)
Default url shortener backend for Zinnia

urls Package

`urls` Package

Defaults urls for the Zinnia project

`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

`discussions` Module

Urls for the Zinnia discussions

`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`, `django.views.generic.list.BaseListView`

Display the entries of an author

`get_context_data``(`***kwargs*`)`
Add the current author in context

`get_model_name``()`
The model name is the author's username

`get_queryset``()`
Return a queryset of entries published belonging to the current author

`model_type` = 'author'

`paginate_by` = 10

class `zinnia.views.authors.`**`AuthorList`**`(`***kwargs*`)`

Bases: `zinnia.views.mixins.callable_queryset.CallableQuerysetMixin`, `django.views.generic.list.ListView`

View returning a list of all published authors

classmethod **`queryset`**`()`

`archives` Module

Views for Zinnia archives

class `zinnia.views.archives.`**`EntryArchiveMixin`**

Bases: `zinnia.views.mixins.archives.ArchiveMixin`, `zinnia.views.mixins.archives.PreviousNex`
`zinnia.views.mixins.callable_queryset.CallableQuerysetMixin`,
`zinnia.views.mixins.templates.EntryQuerysetArchiveTemplateResponseMixin`

Mixin combinating:

- •ArchiveMixin configuration centralizing conf for archive views
- •PreviousNextPublishedMixin for returning published archives
- •CallableQueryMixin to force the update of the queryset
- •EntryQuerysetArchiveTemplateResponseMixin to provide a custom templates for archives

classmethod **`queryset`**`()`

class `zinnia.views.archives.`**`EntryDay`**`(**kwargs)`
Bases: `zinnia.views.mixins.tz_fixes.EntryDayTZFix`, `zinnia.views.archives.EntryArchiveMixir`
`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.EntryQueryse`
`django.views.generic.dates.BaseArchiveIndexView`

View returning the archive index

`context_object_name` **= 'entry_list'**

class `zinnia.views.archives.`**`EntryMonth`**`(**kwargs)`
Bases: `zinnia.views.mixins.tz_fixes.EntryMonthTZFix`, `zinnia.views.archives.EntryArchiveMix`
`django.views.generic.dates.BaseMonthArchiveView`

View returning the archive for a month

`template_name_suffix` **= '_archive_month'**

class `zinnia.views.archives.`**`EntryToday`**`(**kwargs)`
Bases: `zinnia.views.mixins.tz_fixes.EntryDayTZFix`, `zinnia.views.archives.EntryArchiveMixir`
`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.mixins.tz_fixes.EntryWeekTZFix`, `zinnia.views.archives.EntryArchiveMixi`
`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.BaseYearArchive`

View returning the archive for a year

`make_object_list` **= True**

`template_name_suffix` **= '_archive_year'**

`categories` Module

Views for Zinnia categories

class `zinnia.views.categories.`**`CategoryDetail`**(***kwargs*)

Bases: `zinnia.views.mixins.templates.EntryQuerysetTemplateResponseMixin`, `django.views.generic.list.BaseListView`

View returning a list of all the entries belonging to a category

`get_context_data`(***kwargs*)

Add the current category in context

`get_model_name`()

The model name is the category's slug

`get_queryset`()

Return a queryset of entries published belonging to the current category

`model_type` = **'category'**

`paginate_by` = **10**

class `zinnia.views.categories.`**`CategoryList`**(***kwargs*)

Bases: `django.views.generic.list.ListView`

View returning a list of all the categories

`queryset` = **[]**

`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: `zinnia.views.mixins.mimetypes.TemplateMimeTypeView`

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/

`mimetype` = **'text/plain'**

`template_name` = **'zinnia/humans.txt'**

class `zinnia.views.capabilities.`**`OpenSearchXml`**(***kwargs*)

Bases: `zinnia.views.capabilities.CapabilityView`

http://www.opensearch.org/

`mimetype` = **'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

`mimetype` = **'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

`mimetype` = **'application/wlwmanifest+xml'**

`template_name` = **'zinnia/wlwmanifest.xml'**

`channels` Module

Views for Zinnia channels

class `zinnia.views.channels.`**`EntryChannel`**`(**kwargs)`
Bases: `django.views.generic.list.ListView`

View for displaying a custom selection of entries based on a search pattern, useful for SEO/SMO pages

`get_queryset``()`
Override the get_queryset method to do the search

`paginate_by` = **10**

`query` = **''**

`entries` Module

Views for Zinnia entries

class `zinnia.views.entries.`**`EntryDateDetail`**`(**kwargs)`
Bases: `zinnia.views.mixins.tz_fixes.EntryDateDetailTZFix`, `zinnia.views.mixins.archives.ArchiveMixin`, `zinnia.views.mixins.templates.EntryArchiveTem`
`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
- EntryDateDetailTZFix for handing the time-zones correctly in Django 1.4.

`queryset` = **[]**

class `zinnia.views.entries.`**`EntryDetail`**`(**kwargs)`
Bases: `zinnia.views.mixins.entry_protection.EntryProtectionMixin`, `zinnia.views.entries.EntryDateDetail`

Detailled view archive view for an Entry with password and login protections

`mixins` Module

Mixins for Zinnia views

`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

`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='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

Bases: `django.forms.models.ModelForm`

Form for posting an entry quickly

class **`Meta`**

`model`

alias of `Entry`

`QuickEntryForm.`**`base_fields`** **= {'title': <django.forms.fields.CharField object at 0x4031050>, 'image': <django.fo**

`QuickEntryForm.`**`declared_fields`** **= {}**

`QuickEntryForm.`**`media`**

`search` Module

Views for Zinnia entries search

class `zinnia.views.search.`**`EntrySearch`**(***kwargs*)

Bases: `django.views.generic.list.ListView`

View for searching entries

`error` **= None**

`get_context_data`(***kwargs*)

Add error and pattern in the context

`get_queryset`()

Overridde the get_queryset method to do some validations and the search queryset

paginate_by = 10

pattern = ‘’

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 blog

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’ and return the get_absolute_url of the entry

`tags` Module

Views for Zinnia tags

class `zinnia.views.tags.`**`TagDetail`**`(`***kwargs*`)`

Bases: `zinnia.views.mixins.templates.EntryQuerysetTemplateResponseMixin`, `django.views.generic.list.BaseListView`

View return a list of all the entries published under the current tag

get_context_data`(`***kwargs*`)`

Add the current tag in context

get_model_name`()`

The model name is the tag slugified

get_queryset`()`

Return a queryset of entries published belonging to the current tag

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()
Override the get_queryset method to compute and return the published tags

template_name = 'zinnia/tag_list.html'

trackback Module

Views for Zinnia trackback

class zinnia.views.trackback.**EntryTrackback**(***kwargs*)
Bases: zinnia.views.mixins.mimetypes.TemplateMimeTypeView

View for handling trackbacks on the entries

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

mimetype = 'text/xml'

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 = 'creation_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_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*, *period*, *previous=True*)
    Return the next or previous published date period with 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. Who 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_protection` Module Protection mixins for Zinnia views

class `zinnia.views.mixins.entry_protection.`**`EntryProtectionMixin`**
Bases: `object`

Mixin returning a login view if the current entry need authentication and password view if the entry is protected by a password

`error` = False

`get`(*request*, **args*, ***kwargs*)
    Do the login protection

`login`()
    Return the login view

`password`()
    Return the password form

`post`(*request*, **args*, ***kwargs*)
    Do the login protection

`session_key` = 'zinnia_entry_%s_password'

`mimetypes` Module MimeType mixins for Zinnia views

class `zinnia.views.mixins.mimetypes.`**`MimeTypeMixin`**
Bases: `object`

Mixin for handling the mimetype parameter

`get_mimetype`()
    Return the mimetype of the response

mimetype = None

class zinnia.views.mixins.mimetypes.**TemplateMimeTypeView**(***kwargs*)
Bases: zinnia.views.mixins.mimetypes.MimeTypeMixin, django.views.generic.base.TemplateView

TemplateView with a configurable mimetype

render_to_response(*context*, ***kwargs*)
Render the view with a custom mimetype

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 Entr instance as the base template name.

get_default_base_template_name()
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_name()
Return the default base template used to build the 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

tz_fixes Module Mixins for fixing the time zones support in the class based generic views for archives.

This module must be removed in Django 1.5.

https://code.djangoproject.com/ticket/18217

class `zinnia.views.mixins.tz_fixes.`**`EntryDateDetailTZFix`**
Bases: `object`

`get_object`(*queryset=None*)
Get the object this request displays.

class `zinnia.views.mixins.tz_fixes.`**`EntryDayTZFix`**
Bases: `object`

class `zinnia.views.mixins.tz_fixes.`**`EntryMonthTZFix`**
Bases: `object`

`get_dated_items`()
Return (date_list, items, extra_context) for this request.

class `zinnia.views.mixins.tz_fixes.`**`EntryWeekTZFix`**
Bases: `object`

`get_dated_items`()
Return (date_list, items, extra_context) for this request.

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_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.`**`user_structure`**(*user*, *site*)
An user structure

`pingback` Module

XML-RPC methods of Zinnia Pingback

`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

SEVEN

NOTES

7.1 Frequently Asked Questions

Contents

7.1.1 Templates

The templates does not fit to my wishes. What can I do ?

You should take a look on *Customize Zinnia's look and feel*.

7.1.2 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.contrib.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 django-threadcomments for example.

7.1.3 Edition

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.

7.1.4 Authors

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.

7.1.5 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 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.models import Entry
from zinnia.admin import EntryAdmin
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*.

7.2 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 1.4.x version and will reach the release 1.6 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.

7.2.1 Compatibility with Django

Here a list establishing the compatibility between Zinnia and Django: 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`.

7.3 Thanks

Zinnia cannot be a great application without great contributors who make this application greatest each day.

- Bjorn Meyer (bmeyer71)
- Jannis Leideil (jezdez)
- Tobias von Klipstein (klipstein)
- Mark Renton (indexofire)
- Bill Mill (llimllib)
- Kevin Renskers (Bolhoed)
- Jonathan Stoppani (GaretJax)
- Elijah Rutschman (elijahr)
- Thomas Bartelmess (tbartelmess)
- Matthew Tretter (matthewwithanm)
- Mohammad Taleb (simul14)
- Frederic Le guluche (Bloxboy)
- Mauro Bianchi (bianchimro)
- Arion Sprague (asprague)
- Tom Träskbäck (Noobidoo)
- Franck Bret
- Jason Davies
- Brandon Taylor
- bernhardvallant
- nl0
- esauro
- 0Chuzz
- un33K
- orblivion
- kjikaqawej
- igalarzab
- jtrain
- and You.

I also want to thank GitHub.com, Transifex.net, Shining Panda.com and ReadTheDocs.org for their services of great quality.

7.4 CHANGELOG

7.4.1 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`

7.4.2 0.11.1

- Fix issues with `get_absolute_url` and `zbreadcrumbs` when time-zone support is enabled.

7.4.3 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

7.4.4 0.10.1

- Django 1.4 compatibility support
- Compatibility with django-mptt >= 5.1
- `zinnia.plugins` is now removed

7.4.5 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

7.4.6 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

7.4.7 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

7.4.8 0.7

- Using signals
- Trackback support
- Ping external URLs

- Private posts
- Hierarchical categories
- TinyMCE integration
- Code optimizations
- And a lot of bug fixes

7.4.9 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

7.4.10 0.5

- Packaging
- Tests added
- Translations
- Better templates
- New templatetags
- Plugins for django-cms
- Twitter and Bit.ly support
- Publishing sources on Github.com

7.4.11 0.4 and before

- The previous versions of Zinnia were not packaged, and were destinated for a personnal use.

CHAPTER

EIGHT

RELATED

- Test report
- Code coverage

CHAPTER

NINE

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