
django-blog-zinnia Documentation

Release 0.8.1

Fantomas42

October 25, 2011

CONTENTS

Contents:

DJANGO BLOG ZINNIA

Simple yet powerful application for managing a blog within your Django website.

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
- Widgets (Popular entries, Similar entries, ...)
- Spam protection with [Akismet](#)
- 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
- WYMeditor or TinyMCE support
- WordPress conversion utility
- Ready to use and extendables templates
- 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.
- MadCad's Page.

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

INSTALLATION

2.1 Dependencies

Make sure to install these packages prior to installation :

- Django `>= 1.2`
- django-mptt `>= 0.4.2`
- django-tagging `>= 0.3.1`
- BeautifulSoup `>= 3.2.0`

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

- 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.

2.2 Getting the code

You could retrieve the last sources from <http://github.com/Fantomas42/django-blog-zinnia> and run the installation script

```
$ python setup.py install
```

or use pip

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

For the latest stable version use *easy_install*

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

2.3 Applications

Then register **zinnia**, and these following applications in the `INSTALLED_APPS` section of your project's settings.

```
INSTALLED_APPS = (  
    # Your favorite apps  
    'django.contrib.contenttypes',  
    'django.contrib.comments',  
    'django.contrib.sessions',  
    'django.contrib.sites',  
    'django.contrib.admin',  
    'tagging',  
    'mptt',  
    'zinnia',)
```

2.4 Template Context Processors

Add these following template context processors if not already present.

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

2.5 Media Files

You have to make a symbolic link from `zinnia/media/zinnia` directory to your media directory or make a copy named **zinnia**, but if want to change this value, define `ZINNIA_MEDIA_URL` in the `settings.py` as appropriate.

And don't forget to serve this URL.

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

ADVANCED CONFIGURATION

3.1 Sitemaps

One of the cool features of Django is the sitemap application, so if you want to fill your website'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}),)
```

3.2 Akismet

By default the Akismet spam protection is enabled when anyone leaves a comment if you have installed the `akismet` python module.

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'
```

If you don't want spam protection for comments, you can disable it with this setting.

```
ZINNIA_AKISMET_COMMENT = False
```

3.3 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 will do the rest.

3.4 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 authentication for Twitter has changed since September 2010. The actual authentication 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.

3.5 Django-CMS

If you use `Django-cms 2.0`, Zinnia can be integrated into your pages, thanks to the plugin system.

Simply register `zinnia.plugins` in the `INSTALLED_APPS` section of your project's settings.

It will provides custom plugins for adding entries into your pages, an App-Hook and Menus for easy integration.

If you want to use the plugin system of `django-cms` in your entries, an extended `EntryModel` with a **PlaceholderField** is provided.

Add this line in your project's settings.

```
ZINNIA_ENTRY_BASE_MODEL = 'zinnia.plugins.placeholder.EntryPlaceholder'
```

3.6 TinyMCE

If you want to replace WYMEEditor by TinyMCE install `django-tinymce` and follow the [installation instructions](#).

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

3.7 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>

LIST OF SETTINGS

Zinnia has several parameters to configure the application according to your needs.

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

ZINNIA_PAGINATION Default : 10

Integer used to paginate the entries.

ZINNIA_ALLOW_EMPTY Default : True

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

ZINNIA_ALLOW_FUTURE Default : True

Used for allowing archives views in the future.

ZINNIA_ENTRY_TEMPLATES Default : ()

List of tuple for extending the list of templates availables for rendering the entry.

ZINNIA_ENTRY_BASE_MODEL Default : ''

String defining the base Model path for the Entry model. See `TODO_REF` for more informations.

ZINNIA_WYSIWYG Default : 'tinymce' if in settings.INSTALLED_APPS else 'wymeditor'

Used for determining the WYSIWYG editor for editing an entry. Can also be used for disabling the WYSIWYG functionality.

ZINNIA_FEEDS_FORMAT Default : 'rss'

String determining the format of the syndication feeds. Use 'atom' for Atom feeds.

ZINNIA_FEEDS_MAX_ITEMS Default : 15

Integer used to define the maximum items provided in the syndication feeds.

ZINNIA_COPYRIGHT Default : 'Zinnia'

String used for copyrighting the syndication feeds.

ZINNIA_UPLOAD_TO Default : 'uploads'

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

ZINNIA_PROTOCOL Default : 'http'

String representing the protocol of the site.

ZINNIA_MEDIA_URL Default : os.path.join(settings.MEDIA_URL, 'zinnia/')

String of the url that handles the media files of Zinnia.

ZINNIA_MAIL_COMMENT Default : True

Boolean used for sending an email to managers when a comment is posted or not.

ZINNIA_MAIL_COMMENT_REPLY Default : False

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

ZINNIA_AKISMET_COMMENT Default : True

Boolean used for protecting your comments with Akismet or not.

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

List of the directories you want to ping.

ZINNIA_SAVE_PING_DIRECTORIES Default : bool(ZINNIA_PING_DIRECTORIES)

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

ZINNIA_PING_EXTERNAL_URLS Default : True

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

ZINNIA_PINGBACK_CONTENT_LENGTH Default : 300

Size of the excerpt generated on pingback.

ZINNIA_F_MIN Default : 0.1

Float setting of the minimal word frequency for similar entries.

ZINNIA_F_MAX Default : 1.0

Float setting of the minimal word frequency for similar entries.

ZINNIA_USE_BITLY Default : 'django_bitly' in settings.INSTALLED_APPS

Boolean telling if Zinnia can use Bit.ly.

ZINNIA_USE_TWITTER Default : True if python-twitter is in PYTHONPATH

Boolean telling if Zinnia can use Twitter.

4.1 CMS settings

ZINNIA_PLUGINS_TEMPLATES Default : ()

List of tuple for extending the CMS's plugins rendering templates.

ZINNIA_APP_MENUS Default : (EntryMenu, CategoryMenu, TagMenu, AuthorMenu)

List of Menu objects provided for the Zinnia AppHook.

ZINNIA_HIDE_ENTRY_MENU Default : True

Boolean used for displaying or not the entries in the EntryMenu object.

TEMPLATE TAGS

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

- `get_recent_entries(number=5, template="zinnia/tags/recent_entries.html")`

Display the latest entries.

- `get_featured_entries(number=5, template="zinnia/tags/featured_entries.html")`

Display the featured entries.

- `get_random_entries(number=5, template="zinnia/tags/random_entries.html")`

Display random entries.

- `get_popular_entries(number=5, template="zinnia/tags/popular_entries.html")`

Display popular entries.

- `get_similar_entries(number=5, template="zinnia/tags/similar_entries.html")`

Display entries similar to an existing entry.

- `get_calendar_entries(year=auto, month=auto, template="zinnia/tags/calendar.html")`

Display an HTML calendar with date of publications.

- `get_archives_entries(template="zinnia/tags/archives_entries.html")`

Display the archives by month.

- `get_archives_entries_tree(template="zinnia/tags/archives_entries_tree.html")`

Display all the archives as a tree.

- `get_categories(template="zinnia/tags/categories.html")`

Display all the categories available.

- `get_authors(template="zinnia/tags/authors.html")`

Display all the published authors.

- `get_recent_comments(number=5, template="zinnia/tags/recent_comments.html")`

Display the latest comments.

- `get_recent_linkbacks(number=5, template="zinnia/tags/recent_linkbacks.html")`

Display the latest linkbacks.

- `zinnia_breadcrumbs(separator="/", root_name="Blog", template="zinnia/tags/breadcrumbs.html")`

Display the breadcrumbs for the pages handled by Zinnia.

- `get_gravatar(email, size=80, rating='g', default=None)`

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

The usage of the *template* argument allow you to reuse and customize the rendering of a template tag in a generic way. Like this you can display the same template tag many times in your pages but with a different form.

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.

6.1 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'.

6.2 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.

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' written 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 a 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'.

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 *zinnia.views.channels.entry_channel*.

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

```
url(r'^weblog/$', 'zinnia.views.channels.entry_channel',
    {'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.

So our homepage will only display entries filled under the categories 'Python' and 'Django'.

The others parameters handled by the channel view are the same that the generic `object_list` view bundled in Django can handle.

EXTENDING ENTRY MODEL

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 allows Zinnia to be a really generic and reusable application.

Imagine that I find Zinnia really great, but that is misses some fields or features to be the blog app that I need for my django project. For example I need to add a custom field linking to an image gallery, 2 solutions :

- I search for another django blogging app fitting my needs.
- I make a monkey patch, but I won't be able to upgrade to future releases.

These 2 solutions are really bad, that's why Zinnia provides a third solution.

- Customizing the model noninvasively with the power of inheritance.

How do we do that ?

In fact, simply by creating an abstract model inherited from EntryBaseModel, adding fields or/and overriding his methods, and registering it with the ZINNIA_ENTRY_BASE_MODEL setting in your project.

Example for adding a gallery field.

```
from django.db import models
from mygalleryapp.models import Gallery
from zinnia.models import EntryAbstractClass

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

    class Meta:
        abstract = True
```

Now you register the EntryGallery model like this in your project's settings.

```
ZINNIA_ENTRY_BASE_MODEL = 'full.path.to.EntryGallery'
```

You can see another example in the zinnia/plugins/placeholder.py file.

But you have to note 4 important things :

- Do not import the Entry model in your file defining the extended model because it will cause a circular importation.
- Do not put your abstract model in a file named models.py, it will not work for a non obvious reason.
- 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.

- If you extend the Entry model after the syncdb command, you will have to reset the Zinnia application to reflect your changes.

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
```


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

DEVELOPERS

Your contributions are welcomed and needed.

11.1 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.

First of all, please use [VirtualEnv](#) to protect your system, it's not mandatory but handy.

Follow these steps 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.

Once these operations are done, you are ready to develop the zinnia project.

Run this command to launch the tests.

```
$ ./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
```

To launch the demo site, execute these commands.

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

And for building the HTML documentation run this.

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

Pretty easy no ?

11.2 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 :

<http://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](#).

11.3 Online resources

- [Code repository](#).
- [Documentation](#).
- [API documentation](#).
- [Code coverage](#).
- [Discussions and help at Google Group](#).
- For reporting a bug use [Github Issues](#).

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.

12.1 Dumping

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

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

12.2 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
```

12.3 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 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
```

12.4 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.

12.5 Check list

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

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

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
- <http://docs.djangoproject.com/en/dev/topics/testing/>

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

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.

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

As I write these lines the **92** tests in Zinnia cover **87%** of the code bundled in Zinnia. It’s not bad, but the goal (*realistic*) is to reach a **95%** coverage.

I know that a coverage percent does not represent the quality of the tests, but increasing this percent will ensure 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. :)

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 will be here. :)

Predicting the future is a good thing, because it will be soon. Actually Zinnia is designed to handle the 1.2.x version and will reach the release 1.5 easily without major changes.

<http://docs.djangoproject.com/en/dev/internals/deprecation/>

If you are running on the 1.1.x versions you can also use Zinnia by applying the patch located in **patches/compatibility_django_1.1.patch**.

But the patch is not 100% efficient for 1 thing.

The feeds API provided by the `django.contrib.syndication` in the 1.1 versions is deprecated and the Feed classes provided by has been migrated to the new API. This migration is actually incompatible with the 1.1 versions.

The patch only avoid the generation of errors when the tests are runned.

So if someone find a good solution to this problem, the patch will be integrated in the development branch.

FREQUENTLY ASKED QUESTIONS

15.1 Templates

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

The templates provided for Zinnia are simple but complete and as generic as possible. But you can easily change them by [specifying a template directory](#).

You can also create your own app containing some Zinnia's templates based on inheritance. You can find an app example with HTML5 templates for Zinnia which can be a good starting point to make your own at : [Django Blog Quintet](#).

15.2 Comments

- Is it possible to have a better 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 you want the ability to reply on comments, you can take a look at [django-threadcomments](#) for example.

15.3 Edition

- I want to write my entries in [Markdown](#), [RestructuredText](#) or any lightweight markup language, is it possible ?

Yes of course, disable de WYSIWYG in the admin site with the `ZINNIA_WYSIWYG` setting, and use the the [markup application](#) in your templates.

15.4 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.

15.5 Images

- How can I use the image field for fitting to my skin ?

Take a look at [sorl.thumbnail](#) and use his templatetags.

You can do something like this in your templates :

```

```

- 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 ModelInline in your new admin class.

Here an 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 solution is to extend the EntryModel *Extending Entry model*

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 (lilmlib)
- Kevin Renskers (Bolhoed)
- Jonathan Stoppani (GaretJax)
- Franck Bret
- Jason Davies
- Brandon Taylor
- bernhardvallant
- n10
- esauro
- 0Chuzz
- un33K
- orblivion
- kjikaqawej
- igalarzab
- and You.

I also want to thanks Transifex.net and ReadTheDocs.org for their services of great quality.

CHANGELOG

17.1 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

17.2 0.7

- Using signals
- Trackback support
- Ping external urls
- Private posts
- Hierarchical categories
- TinyMCE integration
- Code optimizations
- And a lot of bug fixes

17.3 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

17.4 0.5

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

17.5 0.4 and before

- The previous versions of Zinnia were not packaged, and were destined for a personal use.

RELATED

- [Zinnia's API](#)
- [Code coverage](#)

INDICES AND TABLES

- *search*