Mezzanine Documentation

Release 4.2.2

Stephen McDonald

November 13, 2016

Contents

Welcome to the Mezzanine project. To learn more about Mezzanine please read the Overview which contains a feature list, installation guide and other general information. To get an idea of the types of sites possible with Mezzanine, have a look at the gallery of sites powered by Mezzanine.

Note: A working knowledge of Django is required to work with Mezzanine and the documentation assumes as much. If you're new to Django, you'll need to work through the Django tutorial before being able to understand the concepts used throughout the Mezzanine documentation. *A mantra for working with Mezzanine: Mezzanine Is Just Django* - Ken Bolton, Mezzanine core team member.

Front-end developers can read about how to set up templates for specific Device Handling such as phones and tablets. Mezzanine also comes with the ability for content authors to edit content directly within a page while viewing it on the website. You can read about this and how to implement this feature within templates under In-line Editing.

Back-end developers can get a better technical overview of how content is managed and how to customize Mezzanine in general by reading about Mezzanine's Content Architecture which describes the main components and how to extend them with your own custom content types, or by reading about Model Customization for implementing more low-level customizations as required. There is also a section on the Admin Customization provided by Mezzanine, as well as a Model Graph depicting the relationships between all the models.

System administrators can find out about some of the production requirements and operations in the Deployment and Caching Strategy sections.

Further reading includes Frequently Asked Questions, Utilities, a section on Public User Accounts, support for Multi-Lingual Sites, information about Mezzanine's Search Engine, and a section on Mezzanine's Configuration which outlines the various settings for configuring Mezzanine. Lastly, you can learn about Importing External Blogs into Mezzanine, Twitter Integration, or just browse the auto-generated docs for each of Mezzanine's Packages.

CHAPTER 1

Table Of Contents

Created by Stephen McDonald

1.1 Overview

Mezzanine is a powerful, consistent, and flexible content management platform. Built using the Django framework, Mezzanine provides a simple yet highly extensible architecture that encourages diving in and hacking on the code. Mezzanine is BSD licensed and supported by a diverse and active community.

In some ways, Mezzanine resembles tools such as Wordpress, providing an intuitive interface for managing pages, blog posts, form data, store products, and other types of content. But Mezzanine is also different. Unlike many other platforms that make extensive use of modules or reusable applications, Mezzanine provides most of its functionality by default. This approach yields a more integrated and efficient platform.

Visit the Mezzanine project page to see some of the great sites people have built using Mezzanine.

1.1.1 Features

In addition to the usual features provided by Django such as MVC architecture, ORM, templating, caching and an automatic admin interface, Mezzanine provides the following:

- Hierarchical page navigation
- Save as draft and preview on site
- Scheduled publishing
- Drag-and-drop page ordering
- WYSIWYG editing
- In-line page editing
- Drag-and-drop HTML5 forms builder with CSV export
- SEO friendly URLs and meta data
- Ecommerce / Shopping cart module (Cartridge)
- Configurable dashboard widgets
- Blog engine
- Tagging

- Free Themes, and a Premium Themes Marketplace
- User accounts and profiles with email verification
- Translated to over 35 languages
- Sharing via Facebook or Twitter
- Multi-lingual sites
- Custom templates per page or blog post
- Twitter Bootstrap integration
- API for custom content types
- Search engine and API
- Seamless integration with third-party Django apps
- Multi-device detection and template handling
- One step migration from other blogging engines
- Automated production provisioning and deployments
- Disqus integration, or built-in threaded comments
- Gravatar integration
- Google Analytics integration
- Twitter feed integration
- bit.ly integration
- Akismet spam filtering
- Built-in test suite
- JVM compatible (via Jython)

The Mezzanine admin dashboard:

1.1.2 Support

To **report a security issue**, please send an email privately to core-team@mezzaninecms.com. This gives us a chance to fix the issue and create an official release prior to the issue being made public.

For **all other Mezzanine support**, the primary channel is the mezzanine-users mailing list. Questions, comments, issues, feature requests, and all other related discussions should take place here.

If you're **certain** you've come across a bug, then please use the GitHub issue tracker, however it's crucial that enough information is provided to reproduce the bug, ideally with a small code sample repo we can simply fork, run, and see the issue with. Other useful information includes things such as the Python stack trace generated by error pages, as well as other aspects of the development environment used, such as operating system, database, and Python version. If **you're not sure you've found a reproducible bug**, then please try the mailing list first.

Finally, feel free to drop by the #mezzanine IRC channel on Freenode, for a chat! Lastly, communications in all Mezzanine spaces are expected to conform to the Django Code of Conduct.

1.1.3 Contributing

Mezzanine is an open source project managed using both the Git and Mercurial version control systems. These repositories are hosted on both GitHub and Bitbucket respectively, so contributing is as easy as forking the project on either of these sites and committing back your enhancements.

Please note the following guidelines for contributing:

- Before doing anything, discuss it on the mezzanine-users mailing list first.
- Contributed code must be written in the existing style. For Python (and to a decent extent, JavaScript as well), this is as simple as following the Django coding style and (most importantly) PEP 8. Front-end CSS should adhere to the Bootstrap CSS guidelines.
- Contributions must be available on a separately named branch based on the latest version of the main branch.
- Run the tests before committing your changes. If your changes cause the tests to break, they won't be accepted.
- If you are adding new functionality, you must include basic tests and documentation.

1.1.4 Donating

If you would like to make a donation to continue development of Mezzanine, you can do so via the Mezzanine Project website.

1.1.5 Quotes

- "I'm enjoying working with Mezzanine, it's good work" - Van Lindberg, Python Software Foundation chairman
- "Mezzanine looks like it may be Django's killer app" - Antonio Rodriguez, ex CTO of Hewlett Packard, founder of Tabblo
- "Mezzanine looks pretty interesting, tempting to get me off Wordpress" - Jesse Noller, Python core contributor, Python Software Foundation board member
- "I think I'm your newest fan. Love these frameworks" - Emile Petrone, integrations engineer at Urban Airship
- "Mezzanine is amazing" - Audrey Roy, founder of PyLadies and Django Packages
- "Mezzanine convinced me to switch from the Ruby world over to Python" - Michael Delaney, developer
- "Like Linux and Python, Mezzanine just feels right" - Phil Hughes, Linux For Dummies author, The Linux Journal columnist
- "Impressed with Mezzanine so far" - Brad Montgomery, founder of Work For Pie
- "From the moment I installed Mezzanine, I have been delighted, both with the initial experience and the community involved in its development" - John Campbell, founder of Head3 Interactive
- "You need to check out the open source project Mezzanine. In one word: Elegant" - Nick Hagianis, developer

1.1.6 Installation

The easiest method is to install directly from pypi using pip by running the command below, which will also install the required dependencies mentioned above:

```
$ pip install mezzanine
```

If you prefer, you can download Mezzanine and install it directly from source:

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

Once installed, the command `mezzanine-project` can be used to create a new Mezzanine project in similar fashion to `django-admin.py`:

```
$ mezzanine-project project_name
$ cd project_name
$ python manage.py createdb --noinput
$ python manage.py runserver
```

Note: The `createdb` command is a shortcut for using Django's `migrate` command, which will also install some demo content, such as a contact form, image gallery, and more. If you'd like to omit this step, use the `--nodata` option with `createdb`.

You should then be able to browse to http://127.0.0.1:8000/admin/ and log in using the default account (`username: admin, password: default`). If you'd like to specify a different username and password during set up, simply exclude the `--noinput` option included above when running `createdb`.

For information on how to add Mezzanine to an existing Django project, see the FAQ section of the documentation.

Mezzanine makes use of as few libraries as possible (apart from a standard Django environment), with the following dependencies, which unless noted as optional, should be installed automatically following the above instructions:

- Python 2.7 to 3.5
- Django 1.8 to 1.9
- django-contrib-comments - for built-in threaded comments
- Pillow - for image resizing (Python Imaging Library fork)
- grappelli-safe - admin skin (Grappelli fork)
- filebrowser-safe - for managing file uploads (FileBrowser fork)
- bleach and BeautifulSoup - for sanitizing markup in content
- pytz and tzlocal - for timezone support
- chardet - for supporting arbitrary encoding in file uploads
- django-modeltranslation - for multi-lingual content (optional)
- django-compressor - for merging JS/CSS assets (optional)
- requests and requests_oauthlib - for interacting with external APIs
- pyflakes and pep8 - for running the test suite (optional)

Users on Debian or Ubuntu will require some system packages to support the imaging library:

```
$ apt-get install libjpeg8 libjpeg8-dev
$ apt-get build-dep python-imaging
```

OSX users can do the same via Homebrew:

```
$ brew install libjpeg
```

1.1.7 Themes

A handful of attractive Free Themes are available thanks to @abhinavsohani, while there is also a marketplace for buying and selling Premium Themes thanks to @joshcartme.

1.1.8 Browser Support

Mezzanine's admin interface works with all modern browsers. Internet Explorer 7 and earlier are generally unsupported.

1.1.9 Third-Party Plug-Ins

The following plug-ins have been developed outside of Mezzanine. If you have developed a plug-in to integrate with Mezzanine and would like to list it here, send an email to the mezzanine-users mailing list, or better yet, fork the project and create a pull request with your plug-in added to the list below. We also ask that you add it to the Mezzanine Grid on djangopackages.com.

- Cartridge - ecommerce for Mezzanine.
- Drum - A Hacker News / Reddit clone powered by Mezzanine.
- mezzanine-html5boilerplate - Integrates the html5boilerplate project into Mezzanine.
- mezzanine-mdown - Adds Markdown support to Mezzanine's rich text editor.
- mezzanine-openshift - Setup for running Mezzanine on Redhat's OpenShift cloud platform.
- mezzanine-stackato - Setup for running Mezzanine on ActiveState's Stackato cloud platform.
- mezzanine-blocks - A Mezzanine flavored fork of django-flatblocks.
- mezzanine-widgets - Widget system for Mezzanine.
- mezzanine-themes - A collection of Django/Mezzanine templates.
- mezzanine-twittertopic - Manage multiple Twitter topic feeds from the Mezzanine admin interface.
- mezzanine-captcha - Adds CAPTCHA field types to Mezzanine's forms builder app.
- mezzanine-bookmarks - A multi-user bookmark app for Mezzanine.
- mezzanine-events - Events plugin for Mezzanine, with geocoding via Google Maps, iCalendar files, webcal URLs and directions via Google Calendar/Maps.
- mezzanine-polls - Polls application for Mezzanine.
- mezzanine-pagedown - Adds the Pagedown WYSIWYG editor to Mezzanine.
- mezzanine-careers - Job posting application for Mezzanine.
- mezzanine-recipes - Recipes plugin with built-in REST API.
- mezzanine-slides - Responsive banner slides app for Mezzanine.
- mezzyblocks - Another app for adding blocks/modules to Mezzanine.
- mezzanine-flexipage - Allows designers to manage content areas in templates.
- mezzanine-instagram - A simple Instagram app for Mezzanine.
- mezzanine-wiki - Wiki app for Mezzanine.
- mezzanine-calendar - Calendar pages in Mezzanine

- mezzanine-facebook - Simple Facebook integration for Mezzanine.
- mezzanine-instagram-gallery - Create Mezzanine galleries using Instagram images.
- mezzanine-cli - Command-line interface for Mezzanine.
- mezzanine-categorylink - Integrates Mezzanine's Link pages with its blog categories.
- mezzanine-podcast - A simple podcast streamer and manager for Mezzanine.
- mezzanine-linkcollection - Collect links. Feature them. Share them over RSS.
- cash-generator - Generate GnuCash invoices with Mezzanine.
- mezzanine-foundation - Zurb Foundation theme for Mezzanine.
- mezzanine-file-collections - Simple file collection page type for Mezzanine.
- mezzanine-wymeditor - WYMeditor adapted as the rich text editor for Mezzanine.
- mezzanine-meze - Adds support for reStructuredText, Pygments and more, to Mezzanine's rich text editing.
- mezzanine-pageimages - Add background and banner images per page in Mezzanine.
- mezzanine-protected-pages - Restrict access to pages by group membership.
- mezzanine-page-auth - A Mezzanine module for add group-level permission to pages.
- django-widgy - Widget-oriented content editing. Includes an adapter for Mezzanine and a powerful form builder.
- mezzanine-admin-backup - Export your Mezzanine database and assets directly from the admin.
- mezzanine-mailchimp - Integrate Mezzanine forms with a MailChimp subscription list.
- mezzanine-grappelli - Integrates latest upstream grappelli/filebrowser with Mezzanine.
- mezzanine-workout - Store and display FIT data in Mezzanine.
- mezzanine-agenda - Event functionality for your Mezzanine sites.
- mezzanine-dpaste - Integrate dpaste, a Django pastebin, into your Mezzanine site.
- mezzanine-linkdump - Create, display and track links in Mezzanine.
- mezzanine-people - Categorize and list people in Mezzanine.
- mezzanine-webf - Fabfile for deploying Mezzanine to Webfaction.
- mezzanineopenshift Another setup for Redhat's OpenShift cloud platform.
- mezzanine-bsbanners - Add Twitter Bootstrap Carousels and Jumbotrons to Mezzanine.
- mezzanine-business-theme - Starter business theme for Mezzanine.
- open-helpdesk - A helpdesk app built with Mezzanine.
- mezzanine-invites - Allow site registration via alphanumeric invite codes.
- ansible-mezzanine - Full pipeline (dev, staging, production) deployment of Mezzanine using Ansible.
- mezzanine-modal-announcements - Popup announcements for Mezzanine websites via Bootstrap modals.
- mezzanine-buffer - Buffer integration for Mezzanine.
- mezzanine-slideshows - Allows placement of Mezzanine galleries within other Mezzanine pages as slideshows.
- mezzanine-onepage - Design helper for single-page Mezzanine sites.
- mezzanine-api - RESTful web API for Mezzanine.
- mezzanine-smartling - Integrates Mezzanine content with Smartling Translations.

- mezzanine-shortcodes - Wordpress shortcodes for Mezzanine.

1.1.10 Sites Using Mezzanine

Got a site built with Mezzanine? You can add it to the gallery on the Mezzanine project page by adding it to the list below - just fork the project and create a pull request. Please omit the trailing slash in the URL, as we manually add that ourselves to feature certain sites.

- Citrus Agency
- Mezzanine Project
- Nick Hagianis
- Thomas Johnson
- Central Mosque Wembley
- Ovarian Cancer Research Foundation
- The Source Procurement
- Imageinary
- Brad Montgomery
- Jashua Cloutier
- Alpha & Omega Contractors
- Equity Advance
- Head3 Interactive
- PyLadies
- Ripe Maternity
- Cotton On
- List G Barristers
- Tri-Cities Flower Farm
- daon.ru
- autoindeks.ru
- immiau.ru
- ARA Consultants
- Boîte à Z'images
- The Melbourne Cup
- Diablo News
- Goldman Travel
- IJC Digital
- Coopers
- Joe Julian
- Sheer Ethic
- Salt Lake Magazine

- Boca Raton Magazine
- Photog.me
- Elephant Juice Soup
- National Positions
- Like Humans Do
- Connecting Countries
- tindie.com
- Environmental World Products
- Ross A. Laird
- Etienne B. Roesch
- Recruiterbox
- Mod Productions
- Appsembler
- Pink Twig
- Parfume Planet
- Trading 4 Us
- Chris Fleisch
- Theneum
- My Story Chest
- Philip Sahli
- Raymond Chandler
- Nashsb
- AciBASE
- Matthe Wahn
- Bit of Pixels
- European Crystallographic Meeting
- Dreamperium
- UT Dallas
- Go Yama
- Yeti LLC
- Li Xiong
- Pageworthy
- Prince Jets
- 30 sites in 30 days
- St Barnabas' Theological College
- Helios 3D

- Life is Good
- Building 92
- Pie Monster
- Cotton On Asia
- Ivan Diao
- Super Top Secret
- Jaybird Sport
- Manai Glitter
- Sri Emas International School
- Boom Perun
- Tactical Bags
- apps.de
- Sunfluence
- ggzpreventie.nl
- dakuaiba.com
- Wdiaz
- Hunted Hive
- mjollnir.org
- The Beancat Network
- Raquel Marón
- EatLove
- Hospitality Quotient
- The Andrew Story
- Charles Koll Jewelry
- Creuna (com/dk/fi/no/se)
- Coronado School of the Arts
- SiteComb
- Dashing Collective
- Puraforce Remedies
- Google's VetNet
- 1800RESPECT
- Evenhouse Consulting
- Humboldt Community Christian School
- Atlanta's Living Legacy
- Shipgistix
- Yuberactive

- Medical Myth Busters
- 4player Network
- Top500 Supercomputers
- Die Betroffenen
- uvena.de
- ezless.com
- Dominican Python
- Stackful.io
- Adrenaline
- ACE EdVenture Programme
- Butchershop Creative
- Sam Kingston
- Ludwig von Mises Institute
- Incendio
- Alexander Lillevik
- Walk In Tromsø
- Mandrivia Linux
- Crown Preschool
- Coronado Pathways Charter School
- Raindrop Marketing
- Web4py
- The Peculiar Store
- GrinDin
- 4Gume
- Skydivo
- Noshly
- Kabu Creative
- KisanHub
- Your Song Your Story
- Kegbot
- Fiz
- Willborn
- Copilot Co
- Amblitec
- Gold's Gym Utah
- Appsin - Blog to Native app

- Take Me East
- Code Raising
- ZigZag Bags
- VerifIP
- Clic TV
- JE Rivas
- Heather Gregory Nutrition
- Coronado Island Realty
- Loans to Homes
- Gensler Group
- SaniCo
- Grupo Invista
- Brooklyn Navy Yard
- MEZZaTHEME
- Nektra Advanced Computing
- Bootstrap ASAP
- California Center for Jobs
- Sam Kingston
- Code Juggle DJ
- Food News
- Australian Discworld Conventions
- Distilled
- OpenMRP
- Arkade Snowboarding
- Linktective The Link Checker
- Zetalab
- Make-Up Artists & Hair Stylists Guild
- Anywhereism
- Assistive Listening Device Locator
- Frank & Connie Spitzer
- Coronado Unified School District
- Coronado Inn
- Coronado Schools Foundation
- Light and Life Christian School
- The Morabito Group
- Law Offices of Nancy Gardner

- Soden & Steinberger APLC
- Stalwart Communications
- Ubuntu Consultants
- Wine a Bit Coronado
- Mercury Mastering
- Flowgrammable
- Shibe Mart
- Carlos Isaac Balderas
- Enrico Tröger
- Perugini
- YouPatch
- Batista Peniel
- Perceptyx
- Guddina Coffee
- Atami Escape Resort
- Philip Southwell
- Justine & Katie's Bowtique
- The Grantwell LLC
- PyCon Asia-Pacific
- Nerdot
- Coworking.do
- Arlette Pichardo
- Sani Dental Group
- Biocap 06
- Python Baja California
- The Art Rebellion
- Engineered Arts
- Paul Whipp Consulting
- Lipman Art
- MODCo Group
- Terminal Labs
- Resource Management Companies
- DollFires
- Quantifind
- ZHackers
- Open ERP Arabia

- DataKind
- New Zealand Institute of Economic Research
- CodingHouse
- Triple J Products
- Aaron E. Balderas
- DVD.nl
- Constantia Fabrics
- Potrillo al Pie
- Skyfalk Web Studio
- Firefox OS Partners
- You Name It
- Atlas of Human Infectious Diseases
- The Entrepreneurial School
- Wednesday Martin
- Avaris to Avanim
- Cognitions Coaching and Consulting
- Foundation Engineering Group
- Hivelocity
- Zooply
- Oceana Technologies
- TerraHub
- djangoproject.jp
- Joshua Ginsberg
- Savant Digital
- weBounty
- Oxfam America
- Artivest
- Dark Matter Sheep
- Mission Healthcare
- Two Forty Fives
- Rodeo Austin
- Krisers
- Intentional Creation
- BytesArea
- Debra Solomon
- Pampanga Food Company

- Aman Sinaya
- Deschamps osteo
- Deschamps kine
- Creactu
- scrunch
- App Dynamics
- Homespun Music Instruction
- Fusionbox
- The Street University
- Glebe
- CeoDental Seminars
- Pay By Super
- Noffs
- Spokade
- Brisbane Prosthodontics
- Carbonised
- Derry Donnell
- Dr Kenneth Cutbush
- American Institute for Foreign Study
- Camp America
- Code Source
- The Federation of Egalitarian Communities

1.2 Content Architecture

Content in Mezzanine primarily revolves around the models found in two packages, *mezzanine.core* and *mezzanine.pages*. Many of these models are abstract, and very small in scope, and are then combined together as the building blocks that form the models you'll actually be exposed to, such as *mezzanine.core.models.Displayable* and *mezzanine.pages.models.Page*, which are the two main models you will inherit from when building your own models for content types.

Before we look at *Displayable* and *Page*, here's a quick list of all the abstract models used to build them:

- *mezzanine.core.models.SiteRelated* - Contains a related `django.contrib.sites.models.Site` field.
- *mezzanine.core.models.Slugged* - Implements a title and URL (slug).
- *mezzanine.core.models.MetaData* - Provides SEO meta data, such as title, description and keywords.
- *mezzanine.core.models.TimeStamped* - Provides created and updated timestamps.
- *mezzanine.core.models.Displayable* - Combines all the models above, then implements publishing features, such as status and dates.

- *mezzanine.core.models.Ownable* - Contains a related user field, suitable for content owned by specific authors.
- *mezzanine.core.models.RichText* - Provides a WYSIWYG editable field.
- *mezzanine.core.models.Orderable* - Used to implement drag/drop ordering of content, whether out of the box as Django admin inlines, or custom such as Mezzanine's page tree.

And for completeness, here are the primary content types provided out of the box to end users, that make use of *Displayable* and *Page*:

- *mezzanine.blog.models.BlogPost* - Blog posts that subclass *Displayable* as they're not part of the site's navigation.
- *mezzanine.pages.models.RichTextPage* - Default *Page* subclass, providing a WYSIWYG editable field.
- *mezzanine.pages.models.Link* - *Page* subclass for links pointing to other URLs.
- *mezzanine.forms.models.Form* - *Page* subclass for building forms.
- *mezzanine.galleries.models.Gallery* - *Page* subclass for building image gallery pages.

These certainly serve as examples for implementing your own types of content.

1.2.1 `Displayable` vs `Page`

Displayable itself is also an abstract model, that at its simplest, is used to represent content that contains a URL (also known as a slug). It also provides the core features of content such as:

- Meta data such as a title, description and keywords.
- Auto-generated slug from the title.
- Draft/published status with the ability to preview drafts.
- Pre-dated publishing.
- Searchable by Mezzanine's Search Engine.

Subclassing *Displayable* best suits low-level content that doesn't form part of the site's navigation - such as blog posts, or events in a calendar. Unlike *Page*, there's nothing particularly special about the *Displayable* model - it simply provides a common set of features useful to content.

In contrast, the concrete *Page* model forms the primary API for building a Mezzanine site. It extends *Displayable*, and implements a hierarchical navigation tree. The rest of this section of the documentation will focus on the *Page* model, and the way it is used to build all the types of content a site will have available.

1.2.2 The `Page` Model

The foundation of a Mezzanine site is the model *mezzanine.pages.models.Page*. Each *Page* instance is stored in a hierarchical tree to form the site's navigation, and an interface for managing the structure of the navigation tree is provided in the admin via *mezzanine.pages.admin.PageAdmin*. All types of content inherit from the *Page* model and Mezzanine provides a default content type via the *mezzanine.pages.models.RichTextPage* model which simply contains a WYSIWYG editable field for managing HTML content.

1.2.3 Creating Custom Content Types

In order to handle different types of pages that require more structured content than provided by the *RichTextPage* model, you can simply create your own models that inherit from *Page*. For example if we wanted to have pages that were authors with books:

```
from django.db import models
from mezzanine.pages.models import Page

# The members of Page will be inherited by the Author model, such
# as title, slug, etc. For authors we can use the title field to
# store the author's name. For our model definition, we just add
# any extra fields that aren't part of the Page model, in this
# case, date of birth.

class Author(Page):
    dob = models.DateField("Date of birth")

class Book(models.Model):
    author = models.ForeignKey("Author")
    cover = models.ImageField(upload_to="authors")
```

Next you'll need to register your model with Django's admin to make it available as a content type. If your content type only exposes some new fields that you'd like to make editable in the admin, you can simply register your model using the *mezzanine.pages.admin.PageAdmin* class:

```
from django.contrib import admin
from mezzanine.pages.admin import PageAdmin
from .models import Author

admin.site.register(Author, PageAdmin)
```

Any regular model fields on your content type will be available when adding or changing an instance of it in the admin. This is similar to Django's behaviour when registering models in the admin without using an admin class, or when using an admin class without fieldsets defined. In these cases all the fields on the model are available in the admin.

If however you need to customize your admin class, you can inherit from *PageAdmin* and implement your own admin class. The only difference is that you'll need to take a copy of `PageAdmin.fieldsets` and modify it if you want to implement your own fieldsets, otherwise you'll lose the fields that the *Page* model implements:

```
from copy import deepcopy
from django.contrib import admin
from mezzanine.pages.admin import PageAdmin
from .models import Author, Book

author_extra_fieldsets = ((None, {"fields": ("dob",)}),)

class BookInline(admin.TabularInline):
    model = Book

class AuthorAdmin(PageAdmin):
    inlines = (BookInline,)
    fieldsets = deepcopy(PageAdmin.fieldsets) + author_extra_fieldsets

admin.site.register(Author, AuthorAdmin)
```

When registering content type models with *PageAdmin* or subclasses of it, the admin class won't be listed in the admin index page, instead being made available as a type of *Page* when creating new pages from the navigation tree.

Note: When creating custom content types, you must inherit directly from the *Page* model. Further levels of subclassing are currently not supported. Therefore you cannot subclass the *RichTextPage* or any other custom content types you create yourself. Should you need to implement a WYSIWYG editable field in the way the *RichTextPage* model does, you can simply subclass both *Page* and *RichText*, the latter being imported from *mezzanine.core.models*.

1.2.4 Displaying Custom Content Types

When creating models that inherit from the *Page* model, multi-table inheritance is used under the hood. This means that when dealing with the page object, an attribute is created from the subclass model's name. So given a *Page* instance using the previous example, accessing the `Author` instance would be as follows:

```
>>> Author.objects.create(title="Dr Seuss")
<Author: Dr Seuss>
>>> page = Page.objects.get(title="Dr Seuss")
>>> page.author
<Author: Dr Seuss>
```

And in a template:

```
<h1>{{ page.author.title }}</h1>
<p>{{ page.author.dob }}</p>
{% for book in page.author.book_set.all %}
<img src="{{ MEDIA_URL }}{{ book.cover }}">
{% endfor %}
```

The *Page* model also contains the method `Page.get_content_model()` for retrieving the custom instance without knowing its type:

```
>>> page.get_content_model()
<Author: Dr Seuss>
```

1.2.5 Page Templates

The view function *mezzanine.pages.views.page()* handles returning a *Page* instance to a template. By default the template `pages/page.html` is used, but if a custom template exists it will be used instead. The check for a custom template will first check for a template with the same name as the *Page* instance's slug, and if not then a template with a name derived from the subclass model's name is checked for. So given the above example the templates `pages/dr-seuss.html` and `pages/author.html` would be checked for respectively.

The view function further looks through the parent hierarchy of the *Page*. If a *Page* instance with slug `authors/dr-seuss` is a child of the *Page* with slug `authors`, the templates `pages/authors/dr-seuss.html`, `pages/authors/dr-seuss/author.html`, `pages/authors/author.html`, `pages/author.html`, and `pages/page.html` would be checked for respectively. This lets you specify a template for all children of a *Page* and a different template for the *Page* itself. For example, if an additional author were added as a child page of `authors/dr-seuss` with the slug `authors/dr-seuss/theo-lesieg`, the template `pages/authors/dr-seuss/author.html` would be among those checked.

1.2.6 Page Processors

So far we've covered how to create and display custom types of pages, but what if we want to extend them further with more advanced features? For example adding a form to the page and handling when a user submits the form. This type of logic would typically go into a view function, but since every *Page* instance is handled via the view function *mezzanine.pages.views.page()* we can't create our own views for pages. Mezzanine solves this problem using *Page Processors*.

Page Processors are simply functions that can be associated to any custom *Page* models and are then called inside the *mezzanine.pages.views.page()* view when viewing the associated *Page* instance. A Page Processor will always be passed two arguments - the request and the *Page* instance, and can either return a dictionary that will be added to the template context, or it can return any of Django's HttpResponse classes which will override the *mezzanine.pages.views.page()* view entirely.

To associate a Page Processor to a custom *Page* model you must create the function for it in a module called page_processors.py inside one of your INSTALLED_APPS and decorate it using the decorator *mezzanine.pages.page_processors.processor_for()*.

Continuing on from our author example, suppose we want to add an enquiry form to each author page. Our page_processors.py module in the author app would be as follows:

```
from django import forms
from django.http import HttpResponseRedirect
from mezzanine.pages.page_processors import processor_for
from .models import Author

class AuthorForm(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()

@processor_for(Author)
def author_form(request, page):
    form = AuthorForm()
    if request.method == "POST":
        form = AuthorForm(request.POST)
        if form.is_valid():
            # Form processing goes here.
            redirect = request.path + "?submitted=true"
            return HttpResponseRedirect(redirect)
    return {"form": form}
```

The *processor_for()* decorator can also be given a slug argument rather than a Page subclass. In this case the Page Processor will be run when the exact slug matches the page being viewed.

1.2.7 Page Permissions

The navigation tree in the admin where pages are managed will take into account any permissions defined using Django's permission system. For example if a logged in user doesn't have permission to add new instances of the Author model from our previous example, it won't be listed in the types of pages that user can add when viewing the navigation tree in the admin.

In conjunction with Django's permission system, the *Page* model also implements the methods *can_add()*, *can_change()*, *can_delete()*, and *can_move()*. These methods provide a way for custom page types to implement their own permissions by being overridden on subclasses of the *Page* model.

With the exception of *can_move()*, each of these methods takes a single argument which is the current request object, and return a Boolean. This provides the ability to define custom permission methods with access to the current user as well.

Note: The *can_add()* permission in the context of an existing page has a different meaning than in the context of an overall model as is the case with Django's permission system. In the case of a page instance, *can_add()* refers to the ability to add child pages.

The *can_move()* method has a slightly different interface, as it needs an additional argument, which is the new parent should the move be completed, and an additional output, which is a message to be displayed when the move is denied. The message helps justify reverting the page to its position prior to the move, and is displayed using Django messages framework. Instead of a Boolean return value, *can_move()* raises a *PageMoveException* when the move is denied, with an optional argument representing the message to be displayed. In any case, *can_move()* does not return any values.

Note: The *can_move()* permission can only constrain moving existing pages, and is not observed when creating a new page. If you want to enforce the same rules when creating pages, you need to implement them explicitly through other means, such as the `save` method of the model or the `save_model` method of the model's admin.

For example, if our `Author` content type should only contain one child page at most, can only be deleted when added as a child page (unless you're a superuser), and cannot be moved to a top-level position, the following permission methods could be implemented:

```
from mezzanine.pages.models import Page, PageMoveException

class Author(Page):
    dob = models.DateField("Date of birth")

    def can_add(self, request):
        return self.children.count() == 0

    def can_delete(self, request):
        return request.user.is_superuser or self.parent is not None

    def can_move(self, request, new_parent):
        if new_parent is None:
            msg = 'An author page cannot be a top-level page'
            raise PageMoveException(msg)
```

1.2.8 Page Menus

We've looked closely at the aspects of individual pages, now let's look at displaying all of the pages as a hierarchical menu. A typical site may contain several different page menus, for example a menu that shows primary pages on the header of the site, with secondary pages as drop-down lists. Another type of menu would be a full or partial tree in a side-bar on the site. The footer may display a menu with primary and secondary pages grouped together as vertical lists.

Mezzanine provides the *page_menu()* template tag for rendering the above types of page menus, or any other type you can think of. The *page_menu()* template tag is responsible for rendering a single branch of the page tree at a time, and accepts two optional arguments (you'll usually need to supply at least one of them) in either order. The arguments are the name of a menu template to use for a single branch within the page tree, and the parent menu item for the branch that will be rendered.

The page menu template will be provided with a variable `page_branch`, which contains a list of pages for the current branch. We can then call the *page_menu()* template tag for each page in the branch, using the page as the parent argument to render its children. When calling the `page_menu()` template tag from within a menu template, we don't need to supply the template name again, as it can be inferred. Note that by omitting the parent page argument

for the page_menu() template tag, the first branch rendered will be all of the primary pages, that is, all of the pages without a parent.

Here's a simple menu example using two template files, that renders the entire page tree using unordered list HTML tags:

```
<!-- First template: perhaps base.html, or an include file -->
{% load pages_tags %}
{% page_menu "pages/menus/my_menu.html" %}

<!-- Second template: pages/menus/my_menu.html -->
{% load pages_tags %}
<ul>
{% for page in page_branch %}
<li>
    <a href="{{ page.get_absolute_url }}">{{ page.title }}</a>
    {% page_menu page %}
</li>
{% endfor %}
</ul>
```

The first file starts off the menu without specifying a parent page so that primary pages are first rendered, and only passes in the menu template to use. The second file is the actual menu template that includes itself recursively for each branch in the menu. We could even specify a different menu template in the call to *page_menu()* in our menu template, if we wanted to use a different layout for child pages.

Filtering Menus

Each *Page* instance has a field in_menus which specifies which menus the page should appear in. In the admin interface, the in_menus field is a list of checkboxes for each of the menu templates. The menu choices for the in_menus field are defined by the *PAGE_MENU_TEMPLATES* setting, which is a sequence of menu templates. Each item in the sequence is a three item sequence, containing a unique ID for the template, a label for the template, and the template path. For example in your settings.py module:

```
PAGE_MENU_TEMPLATES = (
    (1, "Top navigation bar", "pages/menus/dropdown.html"),
    (2, "Left-hand tree", "pages/menus/tree.html"),
    (3, "Footer", "pages/menus/footer.html"),
)
```

Which of these entries is selected for new pages (all are selected by default) is controlled by the *PAGE_MENU_TEMPLATES_DEFAULT* setting. For example, PAGE_MENU_TEMPLATES_DEFAULT = (1, 3) will cause the admin section to pre-select the "Top navigation bar" and the "Footer" when using the example above.

The selections made for the in_menus field on each page don't actually filter a page from being included in the page_branch variable that contains the list of pages for the current branch. Instead it's used to set the value of page.in_menu for each page in the menu template, so it's up to your menu template to check the page's in_menu attribute explicitly, in order to exclude it:

```
<!-- Second template again, with in_menu support -->
{% load pages_tags %}
<ul>
{% for page in page_branch %}
{% if page.in_menu %}
<li>
    <a href="{{ page.get_absolute_url }}">{{ page.title }}</a>
    {% page_menu page %}
</li>
```

```
{% endif %}
{% endfor %}
</ul>
```

Note that if a menu template is not defined in the *PAGE_MENU_TEMPLATES* setting, the branch pages supplied to it will always have the `in_menu` attribute set to `True`, so the only way this will be `False` is if the menu template has been added to *PAGE_MENU_TEMPLATES*, and then *not* selected for a page in the admin interface.

Menu Variables

The *page_menu()* template tag provides a handful of variables, both in the template context, and assigned to each page in the branch, for helping you to build advanced menus.

- `page_branch` - a list of pages for the current branch
- `on_home` - a boolean for whether the homepage is being viewed
- `has_home` - a boolean for whether a page object exists for the homepage, which is used to check whether a hard-coded link to the homepage should be used in the page menu
- `branch_level` - an integer for the current branch depth
- `page_branch_in_menu` - a boolean for whether this branch should be in the menu (see "filtering menus" below)
- `parent_page` - a reference to the parent page
- `page.parent` - same as `parent_page`.
- `page.in_menu` - a boolean for whether the branch page should be in the menu (see "filtering menus" below)
- `page.has_children` - a boolean for whether the branch page has any child pages at all, disregarding the current menu
- `page.has_children_in_menu` - a boolean for whether the branch page has any child pages that appear in the current menu
- `page.num_children` - an integer for the number of child pages the branch page has in total, disregarding the current menu
- `page.num_children_in_menu` - an integer for the number of child pages the branch page has, that also appear in the current menu
- `page.is_current_child` - a boolean for whether the branch page is a child of the current page being viewed
- `page.is_current_sibling` - a boolean for whether the branch page is a sibling (has the same parent) of the current page being viewed
- `page.is_current_parent` - a boolean for whether the branch page is the direct parent of the current page being viewed.
- `page.is_current_or_ascendant` - a boolean for whether the branch page is the current page being viewed, or an ascendant (parent, grand-parent, etc) of the current page being viewed
- `page.is_primary` - a boolean for whether the branch page is a primary page (has no parent)
- `page.html_id` - a unique string that can be used as the HTML ID attribute
- `page.branch_level` - an integer for the branch page's depth

Here's a commonly requested example of custom menu logic. Suppose you have primary navigation across the top of the site showing only primary pages, representing sections of the site. You then want to have a tree menu in a sidebar, that displays all pages within the section of the site currently being viewed. To achieve this we recursively move through the page tree, only drilling down through child pages if `page.is_current_or_ascendant` is `True`, or if the page isn't a primary page. The key here is the `page.is_current_or_ascendant` check is only applied to the primary page, so all of its descendants end up being rendered. Finally, we also only display the link to each page if it isn't the primary page for the section:

```
{% load pages_tags %}
<ul>
{% for page in page_branch %}
{% if page.in_menu %}
{% if page.is_current_or_ascendant or not page.is_primary %}
<li>
    {% if not page.is_primary %}
    <a href="{{ page.get_absolute_url }}">{{ page.title }}</a>
    {% endif %}
    {% page_menu page %}
</li>
{% endif %}
{% endif %}
{% endfor %}
</ul>
```

1.2.9 Integrating Third-party Apps with Pages

Sometimes you might need to use regular Django applications within your site, that fall outside of Mezzanine's page structure. Of course this is fine since Mezzanine is just Django - you can simply add the app's urlpatterns to your project's `urls.py` module like a regular Django project.

A common requirement however is for pages in Mezzanine's navigation to point to the urlpatterns for these regular Django apps. Implementing this simply requires creating a page in the admin, with a URL matching a pattern used by the application. With that in place, the template rendered by the application's view will have a `page` variable in its context, that contains the current page object that was created with the same URL. This allows Mezzanine to mark the `page` instance as active in the navigation, and to generate breadcrumbs for the `page` instance as well.

An example of this setup is Mezzanine's blog application, which does not use *`Page`* content types, and is just a regular Django app.

1.3 Model Customization

So far under Content Architecture the concept of subclassing Mezzanine's models has been described. This section describes the hooks Mezzanine provides for directly modifying the behaviour of its models.

1.3.1 Field Injection

Mezzanine provides the setting *EXTRA_MODEL_FIELDS* which allows you to define a sequence of fields that will be injected into Mezzanine's (or any library's) models.

Note: Using the following approach comes with certain trade-offs described below in *Field Injection Caveats*. Be sure to fully understand these prior to using the *EXTRA_MODEL_FIELDS* setting.

Each item in the *EXTRA_MODEL_FIELDS* sequence is a four item sequence. The first two items are the dotted path to the model and its field name to be added, and the dotted path to the field class to use for the field. The third and fourth items are a sequence of positional args and a dictionary of keyword args, to use when creating the field instance.

For example suppose you want to inject a custom `ImageField` from a third party library into Mezzanine's *BlogPost* model, you would define the following in your project's settings module:

```
EXTRA_MODEL_FIELDS = (
    # Four-item sequence for one field injected.
    (
        # Dotted path to field.
        "mezzanine.blog.models.BlogPost.image",
        # Dotted path to field class.
        "somelib.fields.ImageField",
        # Positional args for field class.
        ("Image",),
        # Keyword args for field class.
        {"blank": True, "upload_to": "blog"},
    ),
)
```

Each *BlogPost* instance will now have an `image` attribute, using the `ImageField` class defined in the fictitious `somelib.fields` module.

Another interesting example would be adding a field to all of Mezzanine's content types by injecting fields into the *Page* class. Continuing on from the previous example, suppose you wanted to add a regular Django `IntegerField` to all content types:

```
EXTRA_MODEL_FIELDS = (
    (
        "mezzanine.blog.models.BlogPost.image",
        "somelib.fields.ImageField",
        ("Image",),
        {"blank": True, "upload_to": "blog"},
    ),
    # Example of adding a field to *all* of Mezzanine's content types:
    (
        "mezzanine.pages.models.Page.another_field",
        "IntegerField", # 'django.db.models.' is implied if path is omitted.
        ("Another name",),
        {"blank": True, "default": 1},
    ),
)
```

Note here that the full path for the field class isn't required since a regular Django field is used - the `django.db.models.` path is implied.

1.3.2 Field Injection Caveats

The above technique provides a great way of avoiding the performance penalties of SQL JOINS required by the traditional approach of subclassing models, however some extra consideration is required when used with the migrations management commands included in Django starting from version 1.7. In the first example above, Django's `makemigrations` command views the new `image` field on the *BlogPost* model of the *mezzanine.blog* app. As such, in order to create a migration for it, the migration must be created for the blog app itself and by default would end up in the migrations directory of the blog app, which completely goes against the notion of not modifying the blog app to add your own custom fields.

One approach to address this is to use Django's MIGRATION_MODULES setting and locate your own migration

files somewhere in your project or app. However, if you define a custom directory to store migrations for an app with injected field (e.g: `pages` in the second example above), make sure to do the same for apps that define models that are subclasses of the one you are injecting fields into. Failing to do so will result in broken dependencies for migration files.

The configuration for the second example should containt at least something that looks like:

```
MIGRATION_MODULES = {
    "pages": "path.to.migration.storage.pages_migration",
    "forms": "path.to.migration.storage.forms_migration",
    "galleries": "path.to.migration.storage.galleries_migration",
}
```

To create the new migration files and apply the changes, simply run:

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

Be warned that over time this approach will almost certainly require some manual intervention by way of editing migrations, or modifying the database manually to create the correct state. Ultimately there is a trade-off involved here.

1.3.3 Admin Fields

Whether using the above approach to inject fields into models, or taking the more traditional approach of subclassing models, most often you will also want to expose new fields to the admin interface. This can be achieved by simply unregistering the relevant admin class, subclassing it, and re-registering your new admin class for the associated model. Continuing on from the first example, the code below takes a copy of the `fieldsets` definition for the original *`BlogPostAdmin`*, and injects our custom field's name into the desired position.:

```
# In myapp/admin.py

from copy import deepcopy
from django.contrib import admin
from mezzanine.blog.admin import BlogPostAdmin
from mezzanine.blog.models import BlogPost

blog_fieldsets = deepcopy(BlogPostAdmin.fieldsets)
blog_fieldsets[0][1]["fields"].insert(-2, "image")

class MyBlogPostAdmin(BlogPostAdmin):
    fieldsets = blog_fieldsets

admin.site.unregister(BlogPost)
admin.site.register(BlogPost, MyBlogPostAdmin)
```

1.4 Admin Customization

Mezzanine uses the standard Django admin interface allowing you to add admin classes as you normally would with a Django project, but also provides the following enhancements to the admin interface that are configurable by the developer.

1.4.1 Navigation

When first logging into the standard Django admin interface a user is presented with the list of models that they have permission to modify data for. Mezzanine takes this feature and uses it to provide a navigation menu that persists across every section of the admin interface making the list of models always accessible.

Using the standard Django admin the grouping and ordering of these models aren't configurable, so Mezzanine provides the setting *ADMIN_MENU_ORDER* that can be used to control the grouping and ordering of models when listed in the admin area.

This setting is a sequence of pairs where each pair represents a group of models. The first item in each pair is the name to give the group and the second item is the sequence of app/model names to use for the group. The ordering of both the groups and their models is maintained when they are displayed in the admin area.

For example, to specify two groups `Content` and `Site` in your admin with the first group containing models from Mezzanine's *pages* and *blog* apps, and the second with the remaining models provided by Django, you would define the following in your projects's `settings` module:

```
ADMIN_MENU_ORDER = (
    ("Content", ("pages.Page", "blog.BlogPost", "blog.Comment",)),
    ("Site", ("auth.User", "auth.Group", "sites.Site", "redirects.Redirect")),
)
```

Any admin classes that aren't specifed are included using Django's normal approach of grouping models alphabetically by application name. You can also control this behavior by implementing a *has_module_permission()* method on your admin class, which should return `True` or `False`. When implemented, this method controls whether the admin class appears in the menu or not. Here's an advanced example that excludes the *BlogCategoryAdmin* class from the menu, unless it is explicitly defined in *ADMIN_MENU_ORDER*:

```
from django.contrib import admin

class BlogCategoryAdmin(admin.ModelAdmin):
    """
    Admin class for blog categories. Hides itself from the admin menu
    unless explicitly specified.
    """

    fieldsets = ((None, {"fields": ("title",)}),)

    def has_module_permission(self, request):
        """
        Hide from the admin menu unless explicitly set in ``ADMIN_MENU_ORDER``.
        """
        for (name, items) in settings.ADMIN_MENU_ORDER:
            if "blog.BlogCategory" in items:
                return True
        return False
```

1.4.2 Custom Items

It is possible to inject custom navigation items into the *ADMIN_MENU_ORDER* setting by specifying an item using a two item sequence, the first item containing the title and second containing the named urlpattern that resolves to the url to be used.

Continuing on from the previous example, Mezzanine includes a fork of the popular django-filebrowser application which contains a named urlpattern `fb_browse` and is given the title `Media Library` to create a custom navigation item:

```
ADMIN_MENU_ORDER = (
    ("Content", ("pages.Page", "blog.BlogPost", "blog.Comment",
        ("Media Library", "fb_browse"),)),
    ("Site", ("auth.User", "auth.Group", "sites.Site", "redirects.Redirect")),
)
```

You can also use this two-item sequence approach for regular app/model names if you'd like to give them a custom title.

1.4.3 Dashboard

When using the standard Django admin interface, the dashboard area shown when a user first logs in provides the list of available models and a list of the user's recent actions. Mezzanine makes this dashboard configurable by the developer by providing a system for specifying Django Inclusion Tags that will be displayed in the dashboard area.

The dashboard area is broken up into three columns, the first being wide and the second and third being narrow. Mezzanine then provides the setting *DASHBOARD_TAGS* which is a sequence of three sequences - one for each the three columns. Each sequence contains the names of the inclusion tags in the format `tag_lib.tag_name` that will be rendered in each of the columns .

The list of models and recent actions normally found in the Django admin are available as inclusion tags via *mezzanine_tags.app_list()* and *mezzanine_tags.recent_actions()* respectively. For example, to configure the dashboard with a blog form above the model list in the first column, a list of recent comments in the second column and the recent actions list in the third column, you would define the following in your projects's `settings` module:

```
DASHBOARD_TAGS = (
    ("blog_tags.quick_blog", "mezzanine_tags.app_list"),
    ("comment_tags.recent_comments",),
    ("mezzanine_tags.recent_actions",),
)
```

Here we can see the *quick_blog()* inclusion tag provided by the *mezzanine.blog.templatetags.blog_tags* module and the *recent_comments()* inclusion tag provided by the *mezzanine.generic.templatetags.comment_tags* module.

1.4.4 WYSIWYG Editor

By default, Mezzanine uses the TinyMCE editor to provide rich editing for all model fields of the type `mezzanine.core.fields.RichTextField`. The setting *RICHTEXT_WIDGET_CLASS* contains the import path to the widget class that will be used for editing each of these fields, which therefore provides the ability for implementing your own editor widget which could be a modified version of TinyMCE, a different editor or even no editor at all.

Note: If you'd only like to customize the TinyMCE options specified in its JavaScript setup, you can do so via the *TINYMCE_SETUP_JS* setting which lets you specify the URL to your own TinyMCE setup JavaScript file.

The default value for the *RICHTEXT_WIDGET_CLASS* setting is the string `"mezzanine.core.forms.TinyMceWidget"`. The *TinyMceWidget* class referenced here provides the necessary media files and HTML for implementing the TinyMCE editor, and serves as a good reference point for implementing your own widget class which would then be specified via the *RICHTEXT_WIDGET_CLASS* setting.

In addition to *RICHTEXT_WIDGET_CLASS* you may need to customize the way your content is rendered at the template level. Post processing of the content can be achieved through the *RICHTEXT_FILTERS* setting, which is a

sequence of string, each one containing the dotted path to a Python function, that will be used as a processing pipeline for the content. Think of them like Django's middleware or context processors.

Say, for example, you had a *RICHTEXT_WIDGET_CLASS* that allowed you to write your content in a popular wiki syntax such as markdown. You'd need a way to convert that wiki syntax into HTML right before the content was rendered:

```
# ... in myproj.filter
from markdown import markdown

def markdown_filter(content):
    """
    Converts markdown formatted content to html
    """
    return markdown(content)

# ... in myproj.settings
RICHTEXT_FILTERS = (
    "myproj.filter.markdown_filter",
)
```

With the above, you'd now see the converted HTML content rendered to the template, rather than the raw markdown formatting.

1.4.5 Media Library Integration

Mezzanine's Media Library (based on django-filebrowser) provides a jQuery UI dialog that can be used by custom widgets to allow users to select previously uploaded files.

When using a custom widget for the WYSIWYG editor via the *RICHTEXT_WIDGET_CLASS* setting, you can show the Media Library dialog from your custom widget, by doing the following:

1. Load the following media resources in your widget, perhaps using a Django Media inner class:

   css `filebrowser/css/smoothness/jquery-ui.min.css`

   js

   ```
   mezzanine/js/%s' % settings.JQUERY_FILENAME
   filebrowser/js/jquery-ui-1.8.24.min.js
   filebrowser/js/filebrowser-popup.js
   ```

2. Call the JavaScript function `browseMediaLibrary` to show the dialog. The function is defined in `filebrowser/js/filebrowser-popup.js`, and takes the following two arguments:

   Callback function The function that will be called after the dialog is closed. The function will be called with a single argument, which will be:

   - null: if no selection was made (e.g. dialog is closed by hitting *ESC*), or
   - the path of the selected file.

   Type (optional) Type of files that are selectable in the dialog. Defaults to image.

1.4.6 Singleton Admin

The admin class `mezzanine.utils.admin.SingletonAdmin` is a utility that can be used to create an admin interface for managing the case where only a single instance of a model should exist. Some cases include a single page site, where only a few fixed blocks of text need to be maintained. Perhaps a stand-alone admin section is required for

managing a site-wide alert. There's overlap here with Mezzanine's Configuration admin interface, but you may have a case that warrants its own admin section. Let's look at an example of a site-wide alert model, that should only ever have a single record in the database.

Here's a model with a text field for managing the alert:

```
from django.db import models

class SiteAlert(models.Model):

    message = models.TextField(blank=True)

    # Make the plural name singular, to correctly
    # label it in the admin interface.
    class Meta:
        verbose_name_plural = "Site Alert"
```

Here's our `admin.py` module in the same app:

```
from mezzanine.utils.admin import SingletonAdmin
from .models import SiteAlert

# Subclassing allows us to customize the admin class,
# but you could also register your model directly
# against SingletonAdmin below.
class SiteAlertAdmin(SingletonAdmin):
    pass

admin.site.register(SiteAlert, SiteAlertAdmin)
```

What we achieve by using `SingletonAdmin` above, is an admin interface that hides the usual listing interface that lists all records in the model's database table. When going to the "Site Alert" section of the admin, the user will be taken directly to the editing interface.

1.5 Multi-Lingual Sites

Mezzanine provides optional support for django-modeltranslation which enables content editors to provide multi-lingual content to support sites in multiple languages.

Note: Mezzanine only provides the integration of django-modeltranslation. For dedicated assistance, please check out the documentation for django-modeltranslation: documentation or its code.

1.5.1 Translation Fields in Mezzanine

In order to enable translation fields for Mezzanine content, you will need to install django-modeltranslation and activate the app in your `settings.py`. Once you have installed django-modeltranslation, you can enable support for it by setting the `USE_MODELTRANSLATION` setting to `True` in your project's `settings.py` module, and also defining at least two entries in the `LANGUAGES` setting.

For new projects, `manage.py createdb` will take care of creating extra columns in the database for each language. For current or older projects, you can catch up by running `manage.py sync_translation_fields` and then `manage.py update_translation_fields`.

Note: A django-modeltranslation setting that can help managing the transition for content *partially* in several languages is `MODELTRANSLATION_FALLBACK_LANGUAGES`. This setting allows you to avoid having empty content when the translation is not provided for a specific language. Please consult django-modeltranslation's documentation for more detail.

1.5.2 Translation Fields for Custom Applications

For models that don't inherit from Mezzanine's models, again please consult django-modeltranslation's documentation. To start with, you'll need to provide a `translation.py` module, containing classes describing which of your model fields you wish to translate, as well as registering your models using the `modeltranslation.translator.translator.register` method.

For models that extends Mezzanine capabilities, there are two rules:

Firstly, the app in which your model is defined must be listed *after* the app it is extending from in your `INSTALLED_APPS` setting. For example, *`mezzanine.forms`* extends models from *`mezzanine.pages`* and should appear after it.

Note: If your app defines both models that need to be translated and static content or templates that override default ones from Mezzanine, you'll need to split your app to distinguish between presentation and content. This is due to conflicting ideas with translated model inheritance, and template or static file overriding, in regard to the order of `INSTALLED_APPS`'

Secondly, for an external app, create a `translation.py` module at the root of your app. The content of this file might benefit from `mezzanine.core.translation` depending on what you are extending from. For example, to improve the model from Content Architecture and provide translatable fields:

```
from django.db import models
from mezzanine.pages.models import Page

class Author(Page):
    dob = models.DateField("Date of birth")
    trivia = models.TextField("Trivia")

class Book(models.Model):
    author = models.ForeignKey("Author")
    cover = models.ImageFiled(upload_to="authors")
    title = models.CharField("Title", max_length=200)
```

A corresponding `translation.py` module in this app would look like:

```
from modeltranslation.translator import translator, TranslationOptions
from .models import Author, Book

class TranslatedAuthor(TranslationOptions):
    fields = ('trivia',)

class TranslatedBook(TranslationOptions):
    fields = ('title',)

translator.register(Author, TranslatedAuthor)
translator.register(Book, TranslatedBook)
```

In this case, please note `mezzanine.pages.translation.TranslatedPage` is not referenced in any way. This is due to the fact that *mezzanine.pages.models.Page* is not abstract, and thus has its own table in the database. The fields have already been registered for translation and django-modeltranslation will happily handle it for you.

If you want to extend an abstract model, such as *mezzanine.core.models.Slugged* or *mezzanine.core.models.Displayable*, you will need to subclass their translation registration. An example of this is the *mezzanine.blog* app in its `translation.py` module:

```
from modeltranslation.translator import translator
from mezzanine.core.translation import (TranslatedSlugged,
                                        TranslatedDisplayable,
                                        TranslatedRichText)
from mezzanine.blog.models import BlogCategory, BlogPost

class TranslatedBlogPost(TranslatedDisplayable, TranslatedRichText):
    fields = ()

class TranslatedBlogCategory(TranslatedSlugged):
    fields = ()

translator.register(BlogPost, TranslatedBlogPost)
translator.register(BlogCategory, TranslatedBlogCategory)
```

You don't add translatable fields in your model beside those already defined inside Mezzanine's models. You need to extend from `mezzanine.core.translation` classes, so django-modeltranslation is aware of the abstract fields it will have to manage.

After that, you can `manage.py createdb` for a new project or `manage.py sync_translation_fields` and then `manage.py update_translation_fields` for an existing one.

1.5.3 Translation Fields and Migrations

Mezzanine is shipped with its own migration files but these do not take translation fields into account. These fields are created by every project's `LANGUAGES` setting and thus can't be provided by default. If you want to both manage migrations for your project and enable translation fields, there are two possibilities.

Either you disable translation fields while managing your migrations as usual and then catch up by adding the missing fields if any:

```
# edit settings.py to set USE_MODELTRANSLATION = False
$ python manage.py makemigrations
$ python manage.py migrate
# edit settings.py to set USE_MODELTRANSLATION back to True
$ python manage.py sync_translation_fields
```

This way, your migration files will never contains references to your specific `LANGUAGES` setting.

Or you create migration files including all the translation fields for your project. This way you won't need to rely on the `manage.py sync_translation_fields` command anymore. You will need to define a custom `MIGRATION_MODULES` and then run:

```
$ python manage.py makemigrations
```

Have a look at *Field Injection Caveats* for a better introduction to `MIGRATION_MODULES`‘

1.5.4 Translation for Injected Fields

If you added fields in Mezzanine's models through *EXTRA_MODEL_FIELDS* and want to add translations, you will need to create a custom app that will hold the necessary `translation.py` module. Adding a translation field to all of Mezzanine's content type would look like:

```
EXTRA_MODEL_FIELDS = (
    (
        "mezzanine.pages.models.Page.quote",
        "TextField",
        ("Page's Quote",),
        {"blank": True},
    ),
)
```

The app containing the corresponding `translation.py` module should be defined *after* *mezzanine.pages* in `INSTALLED_APPS` but *before* any app that contains models that subclass *mezzanine.pages.models.Page* (such as *mezzanine.forms*, *mezzanine.galleries* or `cartridge.shop`). The `translation.py` file itself would be:

```
from modeltranslation.translator import translator
from mezzanine.pages.translation import TranslatedPage
from mezzanine.pages.models import Page

class TranslatedInjectedPage(TranslatedPage):
    fields = ('quote',),

translator.unregister(Page)
translator.register(Page, TranslatedInjectedPage)
```

1.5.5 Redistributable Applications for Mezzanine

If you want to provide translation support for your Mezzanine app, make sure it works with both *USE_MODELTRANSLATION* set to `True` or `False`. Mezzanine enforces the value to `False` if django-modeltranslation is not installed.

The *USE_MODELTRANSLATION* setting can therefore be used to check against, when extra steps are required (such as saving an instance of a model in every language). In the case of a project with *USE_MODELTRANSLATION* set to `False`, the `translation.py` module will just be ignored.

The *USE_MODELTRANSLATION* setting is also available in the template's `settings` variable. Have a look at the `includes/language_selector.html` template in *mezzanine.core* for a working example.

1.6 Utilities

The following section documents general utilities available with Mezzanine. While these aren't a core part of Mezzanine itself, they're widely used across many areas of Mezzanine, and can be very useful in conjunction with your own custom content and features.

Firstly covered are the utilities found in the *mezzanine.generic* app, such as *Keywords*, *Threaded Comments*, and *Ratings*. Each of these form a common pattern:

- A model is provided containing generic relationships using Django's django.contrib.contenttypes app
- A custom model field is provided for defining relationships to the *mezzanine.generic* model, which can then be applied to any of your own models

- The custom field injects extra fields onto your model, with de-normalized data populated on save
- Template tags are provided for displaying the related data, forms for posting them, and views for handling form posts where applicable

For a complete implementation reference, take a look at the built-in blog app *mezzanine.blog* which makes use of all these.

Lastly, some of the *General Template Tags* found within *mezzanine.core.templatetags.mezzanine_tags* are covered.

1.6.1 Keywords

Keywords provided by the *mezzanine.generic* app are pervasive throughout Mezzanine. They're assigned to both the *Page* model and the *Displayable* model from which it's derived. Given that these models form the foundation of most content within Mezzanine, more often than not you're dealing with models that are already using keywords.

Suppose we have a regular Django model though, such as our `Book` example from the previous example in Content Architecture:

```
from django.db import models
from mezzanine.generic.fields import KeywordsField

class Book(models.Model):
    author = models.ForeignKey("Author")
    cover = models.ImageField(upload_to="authors")
    keywords = KeywordsField()
```

When editing `Book` instances in the admin, we'll now be able to choose keywords from the pool of keywords used throughout the site, and also assign new keywords if needed. We can then easily query for books given any keywords:

```
Book.objects.filter(keywords__keyword__title__in=["eggs", "ham"])
```

Given a `Book` instance in a template, we can also display the book's keywords using the `keywords_for()` template tag, which will inject a list of keywords into the template, using the `as var_name` variable name argument supplied to it:

```
{% load keyword_tags %}

{% keywords_for book as keywords %}
{% if keywords %}
<ul>
    <li>Keywords:</li>
    {% for keyword in keywords %}
    <li><a href="{% url "books_for_keyword" keyword.slug %}">{{ keyword }}</a></li>
    {% endfor %}
</ul>
{% endif %}
```

You'll see here each `Keyword` instance has a slug field - we use it in a fictitious urlpattern called `books_for_keyword`, which could then retrieve books for a given keyword by slug:

```
Book.objects.filter(keywords__keyword__slug=slug)
```

Any model with a *KeywordsField* field assigned to it will have a `FIELD_NAME_string` field assigned to it, where `FIELD_NAME` is the name given to the *KeywordsField* attribute on your model, which would be `Book.keywords_string` in the above example. Each time keywords change, the `keywords_string` field

is populated with a comma separated string list of each of the keywords. This can be used in conjunction with Mezzanine's Search Engine - behavior that is provided by default for the *Page* and *Displayable* models.

1.6.2 Threaded Comments

Threaded comments provided by the *mezzanine.generic* app are an extension of Django's django_comments app. Mezzanine's threaded comments fundamentally extend Django's comments to allow for threaded conversations, where comments can be made in reply to other comments.

Again as with our `Book` example, suppose we wanted to add threaded conversations to our book pages in templates, we first define comments on the `Book` model:

```
from django.db import models
from mezzanine.generic.fields import CommentsField

class Book(models.Model):
    author = models.ForeignKey("Author")
    cover = models.ImageField(upload_to="authors")
    comments = CommentsField()
```

Then given a `Book` instance named `book` in a template:

```
{% load comment_tags %}

<h3>There are {{ book.comments_count }} comment{{ book.comments_count|pluralize }}</h3>
{% comments_for book %}
```

The `comments_for` template tag is a Django inclusion tag, that includes the template `generic/includes/comments.html`, which recursively includes the template `generic/includes/comment.html` to build up the threaded conversation. To customize the look and feel of the threaded conversation, simply override these templates in your project.

As you can see in the template example we have a `Book.comments_count` field injected onto our `Book` model. This works the same way as described above for the *KeywordsField*, where the name is derived from the name given to the *CommentsField* attribute on the model, and updated each time the number of comments change.

You can also require that users must be logged in to comment. This is controlled by setting the *COMMENTS_ACCOUNT_REQUIRED* setting to `True`. In this case, the comment form will still be displayed, but on submitting a comment, the user will be redirected to the login/signup page, where after logging in, their comment will be posted without having to re-submit it. See the Public User Accounts section for full details on configuring public user accounts in Mezzanine.

1.6.3 Ratings

The ratings provided by the *mezzanine.generic* app allow people to give a rating for any model that has ratings set up. Suppose we wanted to allow people to rate our books from 1 to 10, first we define what the rating range is via the *RATINGS_RANGE* setting:

```
RATINGS_RANGE = range(1, 11)
```

And then add ratings to our `Book` model:

```
from django.db import models
from mezzanine.generic.fields import RatingField

class Book(models.Model):
    author = models.ForeignKey("Author")
```

```
    cover = models.ImageField(upload_to="authors")
    rating = RatingField()
```

And then in our book template:

```
{% load rating_tags %}

{% rating_for book %}
```

The `rating_for()` template tag is another inclusion tag, which uses the template `generic/includes/rating.html`. It simply displays the current average rating, and a form with radio buttons for rating. You may wish to customize this and use visual icons, like stars, for the ratings.

Like the other custom fields in *`mezzanine.generic`*, the *`RatingField`* will inject fields derived from its attribute name onto the model which it's assigned to, which are updated when a new rating is made. Given our `Book` example, the *`RatingField`* would inject:

- `Book.rating_average` - average rating
- `Book.rating_sum` - total sum of all ratings
- `Book.rating_count` - total count of all ratings

Like threaded comments, ratings can be limited to authenticated users by setting the *RATINGS_ACCOUNT_REQUIRED* setting to `True`.

1.6.4 General Template Tags

Following are some template tags defined in *`mezzanine.core.templatetags.mezzanine_tags`* - they're general purpose and can be used across a variety of scenarios.

`fields_for()`

The *`fields_for()`* template tag is a simple tag that takes a form object as its single argument, and renders the fields for the form. It uses the template `core/templates/includes/form_fields.html`, which can then be overridden to customize the look and feel of all forms throughout a Mezzanine site:

```
{% load mezzanine_tags %}

<form method="POST">
    {% fields_for some_form_object %}
    <input type="submit">
</form>
```

`errors_for()`

The *`errors_for()`* template tag is an inclusion tag that takes a form object and renders any error messages with the template `core/templates/includes/form_errors.html`. It plays well with *`fields_for()`*:

```
{% load mezzanine_tags %}

{% errors_for some_form_object %}
<form method="POST">
    {% fields_for some_form_object %}
    <input type="submit">
</form>
```

sort_by()

The *sort_by()* template tag is a general sorting utility. It's a filter tag similar to Django's dictsort filter tag, but instead of only accepting sequences of dicts and a key name, it also accepts sequences of objects and an attribute name, making it much more general purpose.

Here's an example with the *keywords_for()* tag described above, which assigns an `item_count()` attribute to each keyword returned to the template:

```
{% load mezzanine_tags keywords_tags %}

{% keywords_for book as keywords %}
{% for keyword in keywords|sort_by:"item_count" %}
... etc ...
{% endfor %}
```

thumbnail()

The *thumbnail()* template tag provides on-the-fly image resizing. It takes the relative path to the image file to resize, and mandatory width and height arguments.

When the *thumbnail()* template tag is called for a given set of arguments the first time, the thumbnail is generated and its relative path is returned. Subsequent calls with the same arguments will return the same thumbnail path, without resizing it again, so resizes only occur when first requested.

Given our book example's `Book.cover` field, suppose we wanted to render cover thumbnails with a 100 pixel width, and proportional height:

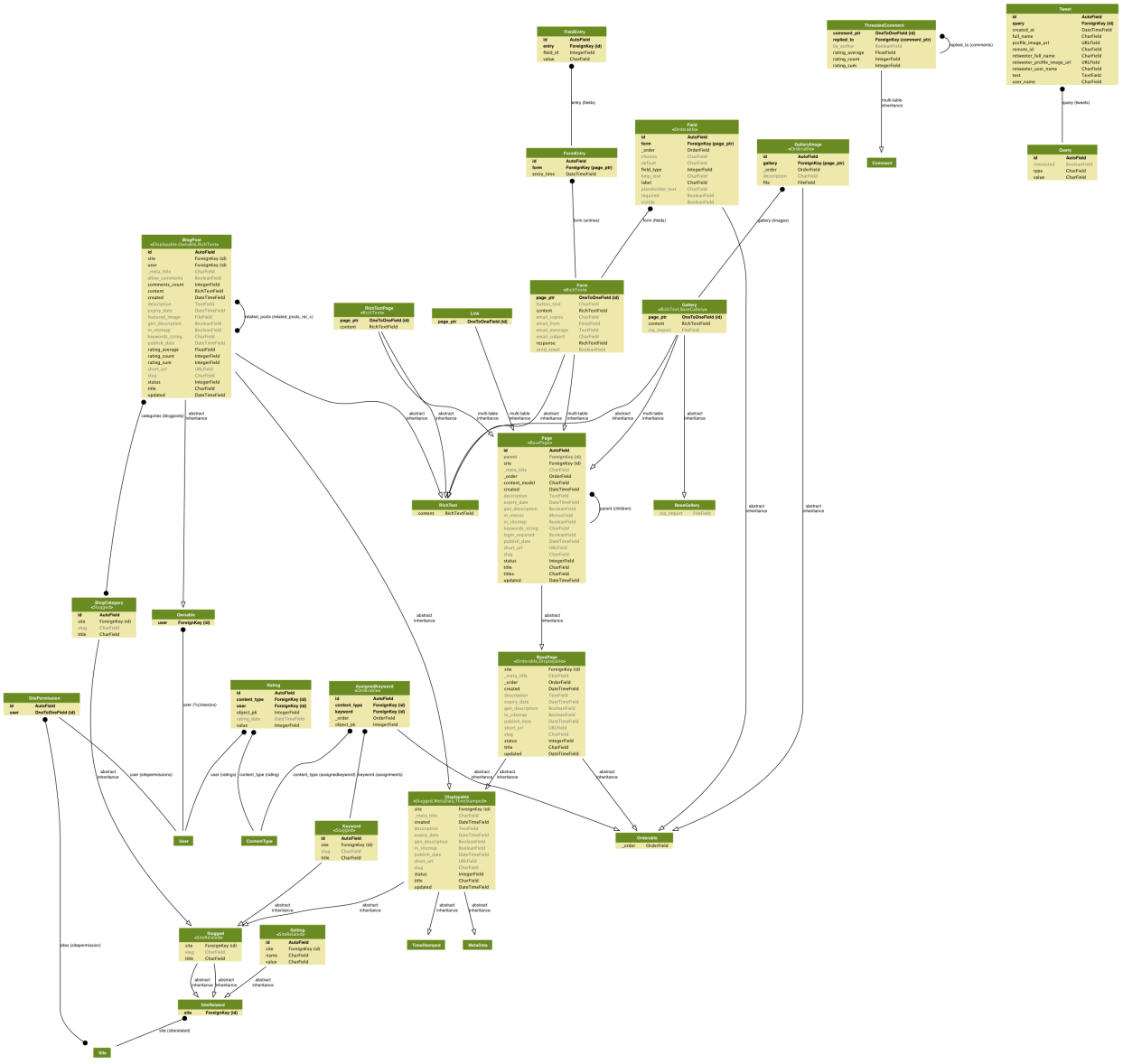
```
{% load mezzanine_tags %}

<img src="{{ MEDIA_URL }}{% thumbnail book.cover 100 0 %}">
```

The *thumbnail()* template tag also accepts several other optional arguments for controlling the generated thumbnail:

- **`upscale` - A boolean controlling whether the thumbnail should** grow beyond its original size when resizing (defaults to True)
- `quality` - A value from 0 to 100 controlling the JPG quality (defaults to 95)
- `left` and `top` - Values from 0 to 1 controlling where the image will be cropped (each defaults to 0.5, namely the center)
- `padding` - A boolean controlling whether the thumbnail will be padded rather than cropped (defaults to False)
- `padding_color` - RGB string controlling the background color when `padding` is True (defaults to "#fff")

1.7 Model Graph

The below diagram depicts the fields and relationships for all the models in Mezzanine. Click it to view a full version.



1.8 Device Handling

Mezzanine comes with the ability to use different sets of templates depending on the device being used to access the website. For example one set of templates may be used for desktop browsers with a corresponding set of templates being used for mobile phones.

Devices are grouped into types with each type being named after the sub-directory containing its specific set of templates. Each device is then defined by a list of strings that could be found in the user agent that matches the particular device. This mapping of device sub-directory names to user agent strings is defined in the setting *DEVICE_USER_AGENTS*:

```
DEVICE_USER_AGENTS = (
    ("mobile", ("Android", "BlackBerry", "iPhone")),
    ("desktop", ("Windows", "Macintosh", "Linux")),
)
```

Given the above example value for *DEVICE_USER_AGENTS* suppose a view or template referenced the template `blog/index.html`. If an iPhone made the request to the website, the template `mobile/blog/index.html` would be searched for, and if a Windows OS made the request then the template `desktop/blog/index.html` would be searched for.

Note: If the device specific templates don't exist or a user agent isn't matched to any of the device specific strings, then the original template name (`blog/index.html` in the above example) will be used as per usual with Django. This means that supporting device specific templates is entirely optional.

You can also specify which device should be treated as the default by defining the setting *DEVICE_DEFAULT*. For example to ensure templates for the `mobile` device group are used even when no matching user agent is found, simply define the following in your project's `settings` module:

```
DEVICE_DEFAULT = "mobile"
```

1.8.1 Mobile Theme

Mezzanine includes the app `mezzanine.mobile` which contains a full set of default templates and assets for creating a mobile version of your site. Simply add `mezzanine.mobile` to your `INSTALLED_APPS` setting to use it.

1.8.2 Implementation Considerations

Using the *DEVICE_USER_AGENTS* setting, Mezzanine simply prefixes any referenced template path with the device specific sub-directory name if a user agent matches one of the strings specified for the device. For example if a user agent matches the `mobile` device set of templates, a reference to `blog/index.html` will be changed to the list `["mobile/blog/index.html", "blog/index.html"]` under the hood.

To achieve this, the middleware *mezzanine.core.middleware.TemplateForDeviceMiddleware* catches Django `TemplateResponse` responses, and changes the template list prior to the response being rendered. As such, any views you implement should return `TemplateResponse` objects. The table below lists Mezzanine versions of Django features that can be used to ensure a `TemplateResponse` is returned.

Django	Mezzanine
`django.shortcuts.render`	*mezzanine.utils.views.render()*
`django.template.Library().inclusion_tag`	`mezzanine.template.Library().inclusion_tag()`
`django.views.generic.simple.direct_to_template`	*mezzanine.core.views.direct_to_template()*

1.9 In-line Editing

Mezzanine comes with the ability for content authors to edit content directly within a page while viewing it on the website, rather than having to log into the admin area. Content authors can simply log into the admin area as usual, but by selecting *Site* on the login screen the author will then be redirected back to the website where a small *Edit* icon will be found next to each piece of editable content, such as a page's title or a blog post's body. Clicking on the Edit icon will allow the author to update the individual piece of content without leaving the page.

In-line editing can be disabled by setting *INLINE_EDITING_ENABLED* to `False`.

1.9.1 Template Configuration

Making content in-line editable is as simple as wrapping model fields with a template tag in your templates. The default templates installed with Mezzanine all have their content configured to be in-line editable. When developing your own templates from scratch though, you'll need to perform this step yourself.

The first step is to ensure the *`editable_loader()`* template tag is called right before the closing `</body>` tag in each template. The recommended way to do this is to include `includes/footer_scripts` in your top-most base template:

```
{% load mezzanine_tags %}
<html>
<head>
    <title>My Website</title>
</head>
<body>
    <!-- Content goes here -->
    {% include "includes/footer_scripts.html" %}
</body>
</html>
```

If your site does not use jQuery, you'll need to include it conditionally in your template's *<head>* if the user is a staff member. If you're using a different JS library, you can use *jQuery.noConflict()* to avoid it overwriting the *$* symbol.

```
{% if user.is_staff %}
    <script src="{{ STATIC_URL }}mezzanine/js/jquery-1.8.3.min.js">
        jQuery.noConflict();
    </script>
{% endif %}
```

The second step is to wrap each instance of a model field with the *`editable()`* and `endeditable` template tags, with the field specified as the *`editable()`* tag's argument. The content between the two tags is what will visibly be hinted to the content author as being editable. It's possible to not provide any content between the two tags, in which case the value for the model field specified for the *`editable()`* tag will simply be used. The model field must always be specified in the format `instance_name.field_name` where `instance_name` is the name of a model instance in the template context. For example, suppose we had a `page` variable in our template with `title` and `content` fields:

```
{% load mezzanine_tags %}
<html>
<head>
    <title>{{ page.title }}</title>
</head>
<body>

    <!--
    No content is specified between the editable tags here, so the
    page.title field is simply displayed inside the <h1> tags.
    -->
    <h1>
        {% editable page.title %}{% endeditable %}
    </h1>

    <!--
    Here we are manipulating how the editable content will be regularly
    displayed on the page using Django's truncatewords_html filter as
```

```
    well as some in-line markup.
    -->
    <div>
        {% editable page.content %}
        <p style="text-align:justify;">
            {{ page.content|truncatewords_html:50 }}
        </p>
        {% endeditable %}
    </div>

    {% editable_loader %}
</body>
</html>
```

The *editable()* template tag also allows multiple fields for a model instance to be given as arguments to a single *editable()* tag. The result of this is still a single Edit icon, but when clicked will display each of the fields specified for editing, grouped together in a single form. Continuing on from the previous example, if we wanted to group together the `title` and `content` fields:

```
{% load mezzanine_tags %}
<html>
<head>
    <title>{{ page.title }}</title>
</head>
<body>

    <!--
    A single Edit icon will be displayed indicating the entire area
    around the h1 and div tags is editable. Clicking it reveals a form
    for editing both fields at once.
    -->
    {% editable page.title page.content %}
    <h1>
        {{ page.title }}
    </h1>
    <div>
        <p style="text-align:justify;">
            {{ page.content|truncatewords_html:50 }}
        </p>
    </div>
    {% endeditable %}

    {% editable_loader %}
</body>
</html>
```

The only caveat to consider with grouping together fields in a single *editable()* tag is that they must all belong to the same model instance.

1.10 Caching Strategy

Mezzanine takes great care to appropriately minimize database queries. This strategy enables Mezzanine to perform well without a caching configuration. However, caching is also well-supported in the event that you wish to implement customized caching for your Mezzanine site. Mezzanine is preconfigured to cache aggressively when deployed to a production site with a cache backend installed.

Note: By using Mezzanine's bundled deployment tools, Mezzanine's caching will be properly configured and in use for your production site. Consult the Deployment section for more information. If you would like to have a cache backend configured but to use a different caching strategy, simply remove the cache middleware described in the next section.

1.10.1 Cache Middleware

Mezzanine's caching system employs a hybrid approach which draws from several popular caching techniques and combines them into one overall implementation. Mezzanine provides its own implementation of Django's page-level cache middleware, and behaves in a similar way.

Pages are fetched from cache by *mezzanine.core.middleware.FetchFromCacheMiddleware*, which should appear at the end of the MIDDLEWARE_CLASSES setting and therefore be activated at the end of the request phase. If a cache miss occurs, the request is marked as requiring a cache update, which is handled by *mezzanine.core.middleware.UpdateCacheMiddleware*, which in turn should appear at the start of MIDDLEWARE_CLASSES and therefore be activated at the end of the response phase.

Mezzanine's cache middleware differs from its Django counterpart in a few subtle yet significant ways:

- Setting CACHE_ANONYMOUS_ONLY to False will have no effect, so authenticated users will never use the cache system.
- Cache keys include the ID for the current Django Site object, and device (see Device Handling).
- Cache keys do not take Vary headers into account, so all unauthenticated visitors will receive the same page content per URL.

1.10.2 Two-Phased Rendering

One approach to caching Django sites is to use template fragment caching, which defines the areas of templates to be cached. Another approach is two-phased rendering, which is the opposite. Using this method, all content is cached by default. We then define the sections of a template that should not be cached. These sections might be anything that makes use of the current request object, including session-specific data.

Accordingly, Mezzanine provides the start and end template tags nevercache() and endnevercache. Content wrapped in these tags will not be cached. With two-phased rendering, the page is cached without any of the template code inside nevercache() and endnevercache executed for the first phase. The second phase then occurs after the page is retrieved from cache (or not), and any template code inside nevercache() and endnevercache is then executed.

Mezzanine's two-phased rendering is based on Cody Soyland's django-phased and Adrian Holovaty's blog post which originally described the technique.

Note: The template code inside nevercache() and endnevercache will only have access to template tags and variables provided by a normal request context, with the exception of any variables passed to the template from a view function. Variables added via context processors such as the current request and via Mezzanine's settings will be available. Template tag libraries should be loaded inside these areas of content so as to make use of their template tags.

1.10.3 Mint Cache

The final step in Mezzanine's caching strategy involves a technique known as mint caching, in which the expiry value for any cache entry is stored in cache along with the cache entry itself. The real expiry value used is the given expiry plus the value defined by Mezzanine's *CACHE_SET_DELAY_SECONDS* setting. Each time a cache entry is requested, the original expiry time is checked, and, if the expiry time has passed, the stale cache entry is placed back into the cache along with a new expiry time using the value of *CACHE_SET_DELAY_SECONDS*. In this case, no cache entry is returned, which has the effect of essentially faking a cache miss, so that the caller can know to regenerate the cache entry. This approach ensures that cache misses never actually occur and that (almost) only one client will ever perform regeneration of a cache entry.

Mezzanine's mint cache is based on this snippet created by Disqus.

1.11 Multi-Tenancy

Mezzanine makes use of Django's `sites` app to support multiple sites in a single project. This functionality is always "turned on" in Mezzanine: a single `Site` record always exists, and is referenced when retrieving site related data, which most content in Mezzanine falls under.

Where Mezzanine diverges from Django is how the `Site` record is retrieved. Typically a running instance of a Django project is bound to a single site defined by the `SITE_ID` setting, so while a project may contain support for multiple sites, a separate running instance of the project is required per site.

Mezzanine uses a pipeline of checks to determine which site to reference when accessing content. The most important of these is the one where the host name of the current request is compared to the domain name specified for each `Site` record. With this in place, true multi-tenancy is achieved, and multiple sites can be hosted within a single running instance of the project.

Here's the list of checks in the pipeline, in order:

- The session variable `site_id`. This allows a project to include features where a user's session is explicitly associated with a site. Mezzanine uses this in its admin to allow admin users to switch between sites to manage, while accessing the admin on a single domain.
- The domain matching the host of the current request, as described above.
- The environment variable `MEZZANINE_SITE_ID`. This allows developers to specify the site for contexts outside of a HTTP request, such as management commands. Mezzanine includes a custom `manage.py` which will check for (and remove) a `--site=ID` argument.
- Finally, Mezzanine will fall back to the `SITE_ID` setting if none of the above checks can occur.

1.11.1 Per-site Themes

Consider a project for an organization or business with several related domains that are to be managed by the same people, or for which sharing of resources is a big benefit. These related domains can share the same Django process, which can offer easier management and reduced resource needs in the server environment.

The domains involved could have a direct subsidiary relationship, as with example.com and several subdomains, or they may be completely separate domains, as with example.com, example2.com, example3.com. Either way, the domains are different hosts to which themes may be independently associated using the *HOST_THEMES* setting:

```
# For a main domain and several subdomains.
HOST_THEMES = [('example.com', 'example_theme'),
               ('something.example.com', 'something_theme'),
               ('somethingelse.example.com', 'somethingelse_theme')]
```

```
# or for separate domains,
HOST_THEMES = [('www.example.com', 'example_theme'),
               ('example.com', 'example_theme'),
               ('www.example2.com', 'example2_theme'),
               ('www.example3.com', 'example3_theme')]
```

In either *HOST_THEMES* example above, there are three themes. Let's continue with the second case, for example.com, example2.com, and example3.com, for which there are three theme apps constructed: `example_theme`, `example2_theme`, and `example3_theme`. The following treatment illustrates a kind of theme inheritance across the domains, with `example_theme` serving as the "base" theme. Suppose `example_theme` contains a dedicated page content type (see *Creating Custom Content Types*), we'll call it the `HomePage`, which is given a model definition in `example_theme/models.py`, along with any other theme-related custom content definitions.

Here is the layout for `example_theme`, the "base" theme in this arrangement:

```
/example_theme
    admin.py
    models.py <-- has a HomePage type, subclassing Page
    ...
    /static
        /css
        /img
        /js
    /templates
        /base.html <-- used by this and the other two themes
        /pages
            /index.html <-- for the HomePage content type
    /templatetags
        some_tags.py <-- for code supporting HomePage functionality
```

The second and third themes, `example2_theme` and `example3_theme`, could be just as expansive, or they could be much simplified, as shown by this layout for `example2_theme` (`example3_theme` could be identical):

```
/example2_theme
    /templates
        /pages
            /index.html <-- for the HomePage content type
```

Each theme would be listed under the `INSTALLED_APPS` setting, with the "base" theme, `example_theme`, listed first.

The project's main `urls.py` would need the following line active, so that "/" is the target URL Mezzanine finds for home page rendering (via the `HomePage` content type):

```
url("^$", "mezzanine.pages.views.page", {"slug": "/"}, name="home"),
```

Mezzanine will look for a page instance at '/' for each theme. `HomePage` instances would be created via the admin system for each site, and given the URL of '/' under the "Meta data" URL field. (Log in to /admin, pick each site, in turn, creating a `HomePage` instance, and editing the "Meta data" URL of each).

Although these aren't the only commands involved, they are useful during the development process:

```
* ``python manage.py startapp theme`` - start a theme; add/edit files
  next; add to INSTALLED_APPS before restart
* ``python manage.py syncdb --migrate`` - after changes to themes;
  could require writing migrations
* ``python manage.py collectstatic`` - gather static resources from the
  themes on occasion
```

Finally, under /admin, these sites will share some resources, such as the media library, while there is separation of content stored in the database (independent `HomePage` instances, independant blog posts, an independent page hierarchy, etc.). Furthermore, the content types added to, say `example_theme`, e.g. `HomePage`, are shared and available in the different sites. Such nuances of sharing must be considered when employing this approach.

1.12 Deployment

Deployment of a Mezzanine site to production is mostly identical to deploying a regular Django site. For serving static content, Mezzanine makes full use of Django's `staticfiles` app. For more information, see the Django docs for deployment and staticfiles.

1.12.1 Fabric

Each Mezzanine project comes bundled with utilities for deploying production Mezzanine sites, using Fabric. The provided `fabfile.py` contains composable commands that can be used to set up all the system-level requirements on a new Debian based system, manage each of the project-level virtual environments for initial and continuous deployments, and much more.

Server Stack

The deployed stack consists of the following components:

- NGINX - public facing web server
- gunicorn - internal HTTP application server
- PostgreSQL - database server
- memcached - in-memory caching server
- supervisord - process control and monitor
- virtualenv - isolated Python environments for each project
- git or mercurial - version control systems (optional)

Note: None of the items listed above are required for deploying Mezzanine, they're simply the components that have been chosen for use in the bundled `fabfile.py`. Alternatives such as Apache and MySQL will work fine, but you'll need to take care of setting these up and deploying yourself. Consult the Django documentation for more information on using different web and database servers.

Configuration

Configurable variables are implemented in the project's `local_settings.py` module. Here's an example, that leverages some existing setting names:

```
# Domains for public site
ALLOWED_HOSTS = ["example.com"]

FABRIC = {
    "DEPLOY_TOOL": "rsync",  # Deploy with "git", "hg", or "rsync"
    "SSH_USER": "server_user",  # VPS SSH username
    "HOSTS": ["123.123.123.123"],  # The IP address of your VPS
```

```
    "DOMAINS": ALLOWED_HOSTS,  # Edit domains in ALLOWED_HOSTS
    "REQUIREMENTS_PATH": "requirements.txt",  # Project's pip requirements
    "LOCALE": "en_US.UTF-8",  # Should end with ".UTF-8"
    "DB_PASS": "",  # Live database password
    "ADMIN_PASS": "",  # Live admin user password
    "SECRET_KEY": SECRET_KEY,
    "NEVERCACHE_KEY": NEVERCACHE_KEY,
}
```

Commands

Here's the list of commands provided in a Mezzanine project's `fabfile.py`. Consult the Fabric documentation for more information on working with these:

- `fab all` - Installs everything required on a new system and deploy.
- `fab apt` - Installs one or more system packages via apt.
- `fab backup` - Backs up the project database.
- `fab create` - Creates the environment needed to host the project.
- `fab deploy` - Deploy latest version of the project.
- `fab install` - Installs the base system and Python requirements for the entire server.
- `fab manage` - Runs a Django management command.
- `fab pip` - Installs one or more Python packages within the virtual environment.
- `fab psql` - Runs SQL against the project's database.
- `fab python` - Runs Python code in the project's virtual environment, with Django loaded.
- `fab remove` - Blow away the current project.
- `fab restart` - Restart gunicorn worker processes for the project.
- `fab restore` - Restores the project database from a previous backup.
- `fab rollback` - Reverts project state to the last deploy.
- `fab run` - Runs a shell comand on the remote server.
- `fab secure` - Minimal security steps for brand new servers.
- `fab sudo` - Runs a command as sudo on the remote server.

1.12.2 Tutorial

CASE 1: Deploying to a brand new server

1. Get your sever. Anything that grants you root access works. VPS's like those from Digital Ocean work great and are cheap.
2. Fill the `ALLOWED_HOSTS` and `FABRIC` settings in `local_settings.py` as shown in the *Configuration* section above. For `SSH_USER` provide any username you want (not root), and the fabfile will create it for you.
3. Run `fab secure`. You simply need to know the root password to your VPS. The new user will be created and you can SSH with that from now on (if needed). For security reason, root login via SSH is disabled by this task.

4. Run `fab all`. It will take a while to install the required environment, but after that, your Mezzanine site will be live.

Notice that not even once you had to manually SSH into your VPS. *Note: some server providers (like Digital Ocean) require you to login as root once to change the default password. It should be the only time you are required to SSH into the sever.*

CASE 2: Deploying to an existing server

If you already have a server, and you already have created a non-root user with sudo privileges:

1. Fill the `ALLOWED_HOSTS` and `FABRIC` settings in `local_settings.py` as shown in the *Configuration* section above. For `SSH_USER` provide the user with sudo privileges.
2. Run `fab install` to install system-wide requirements.
3. Run `fab deploy` to deploy your project.

Deploying more than one site to the server

After you have completed your first deployment, for all subsequent deployments in the same server (either new sites or updates to your existing sites) you only need to run `fab deploy`.

Fixing bugs pushed by accident to the server

1. Run `fab rollback`. This will roll back your project files, database, and static files to how they were in the last (working) deployment.
2. Work on the fixes in your development machine.
3. Run `fab deploy` to push your fixes to production.

1.13 Frequently Asked Questions

These are some of the most frequently asked questions on the Mezzanine mailing list.

- *What do I need to know to use Mezzanine?*
- *Why aren't my JavaScript and CSS files showing up?*
- *Why does the WYSIWYG editor strip out my custom HTML?*
- *Why isn't the homepage a Page object I can edit via the admin?*
- *Why is Mezzanine a Django project, and not a Django app?*
- *Where are all the templates I can modify?*
- *How do I create/install a theme?*
- *Why does Mezzanine contain its own [FEATURE] instead of using [PACKAGE]?*
- *How can I add Mezzanine to an existing Django project?*
- *Why are Grappelli and Filebrowser forked?*
- *What is this Pillow dependency?*
- *Why doesn't Mezzanine have [FEATURE]?*

- *Can I use Cartridge without Mezzanine?*
- *I don't know how to code, how can I contribute?*

1.13.1 What do I need to know to use Mezzanine?

First and foremost, Mezzanine is based on the Django framework. All aspects of working with Mezzanine will benefit from a good understanding of how Django works. Many questions that are asked within the Mezzanine community can easily be answered by reading the Django documentation.

Setting up a development environment, and deploying a Mezzanine site, is the same process as doing so with a regular Django site. Areas such as version control, installing Python packages, and setting up a web server such as Apache or NGINX, will all be touched upon.

Modifying the look and feel of a Mezzanine powered site requires at least an understanding of HTML, CSS and Django's templating system.

Extending Mezzanine by *Creating Custom Content Types* or using additional Django apps, will require some knowledge of programming with Python, as well as a good understanding of Django's components, such as models, views, urlpatterns and the admin.

Back to top

1.13.2 Why aren't my JavaScript and CSS files showing up?

Mezzanine makes exclusive use of Django's staticfiles app, for managing static files such as JavaScript, CSS, and images.

When the `DEBUG` setting is set to `True`, as it would be during development, the URL defined by the setting `STATIC_URL` (usually `/static/`), will host any files found in the `static` directory of any application listed in the `INSTALLED_APPS` setting.

When `DEBUG` is set to `False`, as it would be for your deployed production site, you must run the `collectstatic` command on your live site, which will copy all of the files from the `static` directory in each application, to the location defined by the `STATIC_ROOT` setting. You then need to configure an alias in your web server's config (Apache, NGINX, etc) that maps the URL defined by `STATIC_URL` to serve files from this directory.

Long story short, Django doesn't serve static content when deployed in production, leaving this up to the public facing web server, which is absolutely the best tool for this job. Consult Django's staticfiles guide for more information.

Back to top

1.13.3 Why does the WYSIWYG editor strip out my custom HTML?

By default, Mezzanine strips out potentially dangerous HTML from fields controlled by the WYSIWYG editor, such as tags and attributes that could be used to inject JavaScript into a page. If this didn't occur, a clever staff member could potentially add JavaScript to a page, that when viewed by an administrator (a staff member with superuser status), would cause the administrator's browser to post an update via the admin, that updates the staff member's user account and assigns them superuser status.

The above scenario is a fairly obscure one, so it's possible to customise the level of filtering that occurs. Three levels of filtering are implemented by default, that can be controlled in the settings section of the admin. These are High (the default), Low (which allows extra tags such as those required for embedding videos), and None (no filtering occurs). This is implemented via the *RICHTEXT_FILTER_LEVEL* setting.

If your situation is one where your staff members are completely trusted, and custom HTML within WYSIWYG fields is required, then you can modify the filter level accordingly. Further customisation is possible via the

RICHTEXT_ALLOWED_TAGS, *RICHTEXT_ALLOWED_ATTRIBUTES* and *RICHTEXT_ALLOWED_STYLES* settings, which can have extra allowed values appended to using the `append` argument in Mezzanine's settings API. See the *Registering Settings* section for more information.

Back to top

1.13.4 Why isn't the homepage a Page object I can edit via the admin?

In our experience, the homepage of a beautiful, content driven website, is quite different from other pages of the site, that all fall under sets of repeatable page types. The homepage also differs greatly from site to site. Given this, Mezzanine doesn't presume how your homepage will be structured and managed. It's up to you to implement how it works per site.

By default, the homepage provided with Mezzanine is a static template, namely `mezzanine/core/templates/index.html` (or `templates/index.html` if stored directly in your project). You can change the `urlpattern` for the homepage in your project's `urls.py` module. Be certain to take a look at the urls.py module, as it contains several examples of different types of homepages. In `urls.py` you'll find examples of pointing the homepage to a *Page* object in the page tree, or pointing the homepage to the blog post listing page, which is useful for sites that are primarily blogs.

Of course with Django's models, admin classes, and template tags, the sky is the limit and you're free to set up the homepage to be managed in any way you like.

Back to top

1.13.5 Why is Mezzanine a Django project, and not a Django app?

Mezzanine comes with many features that are related to content driven websites, yet are quite distinct from each other. For example user-built forms and blog posts are both common requirements for a website, yet aren't particularly related to each other. So Mezzanine as a whole is a collection of different Django apps, all packaged together to work seamlessly.

Mezzanine provides its own project template, with `settings.py` and `urls.py` modules that configure all of Mezzanine's apps, which you can (and should) modify per project.

Back to top

1.13.6 Where are all the templates I can modify?

Each of the templates Mezzanine provides can be found in the `templates` directory of each Django app that Mezzanine is comprised of. Take the time to explore the structure of these, starting with the base template `mezzanine/core/templates/base.html` (or `templates/base.html` if stored directly in your project) which is the foundation for the entire site, going more granular as needed.

Once you're familiar with the templates you'd like to modify, copy them into your project's `templates` directory and modify them there. You can also use the `collecttemplates` command to copy templates over automatically. Run `python manage.py collecttemplates --help` for more info. Be mindful that this means the copied templates will always be used, rather than the ones stored within Mezzanine itself, which is something to keep in mind if you upgrade to a newer version of Mezzanine.

Back to top

1.13.7 How do I create/install a theme?

Prior to version 1.0, Mezzanine had a set of features for creating and installing themes. These mostly were in place to address handling static files, since at that time Mezzanine was not integrated with Django's staticfiles app. Mezzanine 1.0 makes full use of `staticfiles`, and so the theming features were removed since they became redundant.

From that point on, a theme in Mezzanine can be implemented entirely as a standard Django app. Simply create a Django app with `templates` and `static` directories, copy the relevant HTML, CSS and JavaScript files into it from Mezzanine that you wish to modify, and then add the theme app's name to your project's `INSTALLED_APPS` setting. Be sure to add the theme to the top of the `INSTALLED_APPS` list, so that its templates are found before Mezzanine's versions of the templates.

Have you created a cool theme that you'd like to share with the community? Package your theme up and put it on PyPI and let us know via the mailing list- that way people can automatically install it along with their Mezzanine project.

Back to top

1.13.8 Why does Mezzanine contain its own [FEATURE] instead of using [PACKAGE]?

To be honest you could implement most of Mezzanine's features by gluing together dozens of smaller, stand-alone, open source Django apps. Several larger Django site-building frameworks take this approach, and it's a noble one. The downside to this is that a significant portion of time on your project will be spent maintaining the glue between these apps, as their development evolves independently from each other, as well as from your project itself. At best you'll be able to work with the apps' developers to ease this evolution, at worst you'll be stuck hacking work-arounds for incompatibilities between the apps.

One of the core goals of Mezzanine is to avoid this situation, by providing all of the features commonly required by content driven sites, with just the right level of extensibility to customize your Mezzanine powered site as required. By taking this approach, the team behind Mezzanine is in complete control over its components, and can ensure they work together seamlessly.

Back to top

1.13.9 How can I add Mezzanine to an existing Django project?

Mezzanine is a Django project made up of multiple Django apps, and is geared towards being used as the basis for new Django projects, however adding Mezzanine to an existing Django project should be as simple as adding the necessary settings and urlpatterns.

Mezzanine contains a project_template directory, which it uses to create new projects. In here you'll find the necessary `settings.py` and `urls.py` modules, containing the project-level setup for Mezzanine. Of particular note are the following settings:

- `INSTALLED_APPS`
- `TEMPLATE_CONTEXT_PROCESSORS`
- `MIDDLEWARE_CLASSES`
- `PACKAGE_NAME_GRAPPELLI` and `PACKAGE_NAME_FILEBROWSER` (for django-grappelli and django-filebrowser integration)
- The call to `mezzanine.utils.conf.set_dynamic_settings` at the very end of the `settings.py` module.

Back to top

1.13.10 Why are Grappelli and Filebrowser forked?

Grappelli and Filebrowser are fantastic Django apps, and Mezzanine's admin interface would be much poorer without them. When Mezzanine was first created, both of these apps had packaging issues that went unaddressed for quite some time. Development of Mezzanine moved extremely quickly during its early days, and so the forks grappelli_safe and filebrowser_safe were created to allow Mezzanine to be packaged up and installed in a single step.

Over time the packaging issues were resolved, but Grappelli and Filebrowser took paths that weren't desired in Mezzanine. They're only used in Mezzanine for skinning the admin, and providing a generic media library. Extra features that have been added to Grappelli and Filebrowser along the way, haven't been necessary for Mezzanine.

Over time, small changes have also been made to the `grappelli_safe` and `filebrowser_safe` forks, in order to integrate them more closely with Mezzanine. So to this day, the forks are still used as dependencies. They're stable, and have relatively low activity.

Back to top

1.13.11 What is this Pillow dependency?

Mezzanine makes use of Python Imaging Library (PIL) for generating thumbnails. Having PIL as a dependency that gets automatically installed with Mezzanine has caused issues for some people, due to certain issues with PIL's own packaging setup.

Pillow is simply a packaging wrapper around PIL that addresses these issues, and ensures PIL is automatically installed correctly when installing Mezzanine. Pillow is only used when PIL is not already installed.

Back to top

1.13.12 Why doesn't Mezzanine have [FEATURE]?

The best answer to this might be found by searching the mailing list, where many features that aren't currently in Mezzanine have been thoroughly discussed.

Sometimes the conclusion is that certain features aren't within the scope of what Mezzanine aims to be. Sometimes they're great ideas, yet no one has had the time to implement them yet. In the case of the latter, the quickest way to get your feature added is to get working on it yourself.

Communication via the mailing list is key though. Features have been developed and rejected before, simply because they were relatively large in size, and developed in a silo without any feedback from the community. Unfortunately these types of contributions are difficult to accept, since they have the greatest resource requirements in understanding everything involved, without any previous communication.

Back to top

1.13.13 Can I use Cartridge without Mezzanine?

No. Cartridge (an ecommerce app) heavily leverages Mezzanine, and in fact it is implemented as an advanced example of a Mezzanine content type, where each shop category is a page in Mezzanine's navigation tree. This allows for a very flexible shop structure, where hierarchical categories can be set up to create your shop.

You could very well use Cartridge and Mezzanine to build a pure Cartridge site, without using any of Mezzanine's features that aren't relevant to Cartridge. However more often than not, you'll find that general content pages and forms, will be required to some extent anyway.

Back to top

1.13.14 I don't know how to code, how can I contribute?

You're in luck! Programming is by far the most abundant skill contributed to Mezzanine, and subsequently the least needed. There are many ways to contribute without writing any code:

- Answering questions on the mailing list
- Triaging issues on GitHub
- Improving the documentation
- Promoting Mezzanine via blogs, Twitter, etc.

If you don't have time for any of these things, and still want to contribute back to Mezzanine, donations are always welcome and can be made via Flattr or PayPal on the Mezzanine homepage. Donations help to support the continued development of Mezzanine, and go towards paying for infrastructure, such as hosting for the demo site.

Back to top

1.14 Public User Accounts

Mezzanine provides the ability for public users to create their own accounts for logging into your Mezzanine powered site. Features that can be restricted to logged-in users include the ability to post comments, make purchases (using Cartridge), view restricted pages, and anything else you'd like to implement. You can also define what a user's profile consists of, allowing users to create their own profile page for their account.

The accounts functionality is provided by the app *mezzanine.accounts*. Adding it to your INSTALLED_APPS setting will enable signup, login, account updating, and password retrieval features for the public site.

1.14.1 Profiles

Profiles are implemented via the *ACCOUNTS_PROFILE_MODEL* setting. With *mezzanine.accounts* installed, you can create a profile model in one of your apps, with each of the profile fields defined, as well as a related field to Django's user model. For example suppose we wanted to capture bios and dates of birth for each user:

```
# In myapp/models.py

from django.db import models

class MyProfile(models.Model):
    user = models.OneToOneField("auth.User")
    date_of_birth = models.DateField(null=True)
    bio = models.TextField()


# In settings.py

INSTALLED_APPS = (
    "myapp",
    "mezzanine.accounts",
    # Many more
)

ACCOUNTS_PROFILE_MODEL = "myapp.MyProfile"
```

The bio and date of birth fields will be available in the signup and update profile forms, as well as in the user's public profile page.

Note: Profile pages are automatically made available when a profile model is configured.

1.14.2 Restricting Account Fields

By default, Mezzanine will expose all relevant user and profile fields available in the signup and update profile forms, and the user's profile page. However you may want to store extra fields in user profiles, but not expose these fields to the user. You may also want to have no profile model at all, and strip the signup and update profile forms down to only the minimum required fields on the user model, such as username and password.

Mezzanine defines the setting *ACCOUNTS_PROFILE_FORM_EXCLUDE_FIELDS* which allows you to define a sequence of field names, for both the user and profile models, that won't be exposed to the user in any way. Suppose we define a `DateTimeField` on the profile model called `signup_date` which we don't want exposed. We also might not bother asking the user for their first and last name, which are fields defined by Django's user model. In our `settings.py` module we would define:

```
ACCOUNTS_PROFILE_FORM_EXCLUDE_FIELDS = (
    "first_name",
    "last_name",
    "signup_date",
)
```

If you don't want to expose the `username` field to the user, Mezzanine provides the setting *ACCOUNTS_NO_USERNAME*, which when set to `True`, will expose the `email` field as the sole login for the user.

1.14.3 Account Verification

By default, with *mezzanine.accounts* installed, any public visitor to the site can sign up for an account and will be logged in after signup. However you may wish to validate that new accounts are only created by real people with real email addresses. To enable this, Mezzanine provides the setting *ACCOUNTS_VERIFICATION_REQUIRED*, which when set to `True`, will send new user an email with a verification link that they must click on, in order to activate their account.

1.14.4 Account Approval

You may also wish to manually activate newly created public accounts. To enable this, Mezzanine provides the setting *ACCOUNTS_APPROVAL_REQUIRED*, which when set to `True`, will set newly created accounts as inactive, requiring a staff member to activate each account in the admin interface. A list of email addresses can be configured in the admin settings interface, which will then be notified by email each time a new account is created and requires activation. Users are then sent a notification when their accounts are activated by a staff member.

1.15 Search Engine

Mezzanine provides a built-in search engine that allows site visitors to search across different types of content. It includes several tools that enable developers to adjust the scope of the site search. It also includes a Search API to programmatically interact with the search engine, customize the way the search engine accesses different types of content, and perform search queries that are broken down and used to query models for results.

1.15.1 Search Form

Developers can easily customize the scope of the searches via the `{% search_form %}` template tag. A default list of searchable models can be specified in the *SEARCH_MODEL_CHOICES* setting. Only models that subclass *mezzanine.core.models.Displayable* should be used. In addition, the actual HTML form can be customized in the `includes/search_form.html` template.

Note: In `SEARCH_MODEL_CHOICES` and `{% search_form %}`, all model names must be strings in the format `app_label.model_name`. These models can be part of Mezzanine's core, or part of third party applications. However, all these model must subclass *Page* or *Displayable*.

Using `{% search_form "all" %}` will render a search form with a dropdown menu, letting the user choose on what type of content the search will be performed. The dropdown will be populated with all of the models found in *SEARCH_MODEL_CHOICES* (default: pages and blog posts, with products added if Cartridge is installed).

By passing a sequence of space-separated models to the tag, only those models will be made available as choices to the user. For example, to offer search for only the *Page* and `Product` models (provided Cartridge is installed), you can use: `{% search_form "pages.Page shop.Product" %}`.

If you don't want to provide users with a dropdown menu, you can limit the search scope to a single model, by passing the model name as a parameter. For example, to create a blog-only search form, you can use `{% search_form "blog.BlogPost" %}`.

If no parameter is passed to `{% search_form %}`, no drop-down will be provided, and the search will be performed on all models defined in the *SEARCH_MODEL_CHOICES* setting.

Finally, by setting *SEARCH_MODEL_CHOICES* to `None`, the search form will not contain a drop-down, but in this case all models that subclass *Displayable* will be automatically searched.

1.15.2 Search API

The main search API is provided by *mezzanine.core.managers.SearchableManager*. This is a Django model manager that provides a custom *SearchableManager.search()* method. Adding search functionality to any model is as simple as using the *SearchableManager* as a manager for your model.

Note: By following the previous example outlined in *Creating Custom Content Types* no extra work is required to have your custom content included in search queries, as the default search functionality in Mezzanine (defined in *mezzanine.core.views.search()*) automatically covers any models that inherit from *mezzanine.pages.models.Page* or *mezzanine.core.models.Displayable*.

In its most simple form, the *SearchableManager.search()* method takes a single string argument containing a search query and returns a Django queryset representing the results. For example, to search for all pages using the term **plans prices projects**:

```
from mezzanine.pages.models import Page

results = Page.objects.search("plans prices projects")
```

It's also possible to explicitly control which fields will be used for the search. For example to search `Page.title` and `Page.content` only:

```
from mezzanine.pages.models import Page

query = "plans prices projects"
```

```
search_fields = ("title", "content")
results = Page.objects.search(query, search_fields=search_fields)
```

If `search_fields` is not provided in the call to `search`, the fields used will be the default fields specified for the model. These are specified by providing a `search_fields` attribute on any model that uses the *`SearchableManager`*. For example, if we wanted to add search capabilities to our *`GalleryImage`* model from the previous example in *Creating Custom Content Types*:

```
from django.db import models
from mezzanine.pages.models import Page
from mezzanine.core.managers import SearchableManager

class Gallery(Page):
    pass

# Added the title and description fields here for the search example.
class GalleryImage(models.Model):
    gallery = models.ForeignKey("Gallery")
    title = models.CharField("Title", max_length=100)
    description = models.CharField("Description", max_length=1000)
    image = models.ImageField(upload_to="galleries")

    objects = SearchableManager()
    search_fields = ("title", "description")
```

Note: If `search_fields` are not specified using any of the approaches above, then all `CharField` and `TextField` fields defined on the model are used. This isn't the case for *`Page`* subclasses though, since the *`Page`* model defines a `search_fields` attribute which your subclass will also contain, so you'll need to explicitly define `search_fields` yourself.

1.15.3 Ordering Results

By default, results are ordered by the number of matches found within the fields searched. It is possible to control the relative weight of a match found within one field over a match found in another field. Given the first example of searching *`Page`* instances, you might decide that a match within the `title` field is worth 5 times as much as a match in the `description` field. These relative weights can be defined in the same fashion as outlined above for defining the fields to be used in a search by using a slightly different format for the `search_fields` argument:

```
from mezzanine.pages.models import Page

query = "plans prices projects"
search_fields = {"title": 5, "content": 1}
results = Page.objects.search(query, search_fields=search_fields)
```

As shown, a dictionary or mapping sequence can be used to associate weights to fields in any of the cases described above where `search_fields` can be defined.

1.15.4 Searching Heterogeneous Models

So far we've looked at how to search across a single model, but what if we want to search across different types of models at once? This is possible through the use of abstract models. *`SearchableManager`* is designed so that if it is accessed directly through an abstract model, it will search across every model that subclasses the abstract model. This makes it possible to group together different types of models for the purpose of combined search. Continuing on

from our *GalleryImage* example, suppose we also have a `Document` model containing files uploaded and that we wanted a combined search across these models which could both be conceptually defined as assets. We would then go ahead and create an abstract model called `Asset` for the sake of grouping these together for search:

```
class Asset(models.Model):
    title = models.CharField("Title", max_length=100)
    description = models.CharField("Title", max_length=1000)

    objects = SearchableManager()
    search_fields = ("title", "description")

    class Meta:
        abstract = True

class GalleryImage(Asset):
    gallery = models.ForeignKey("Gallery")
    image = models.ImageField(upload_to="galleries")

class Document(Asset):
    image = models.FileField(upload_to="documents")
```

By accessing *SearchableManager* directly via the `Asset` abstract model we can search across the `GalleryImage` and `Document` models at once:

```
>>> Asset.objects.search("My")
[<GalleryImage: My Image 1>, <Document: My Doc>, <GalleryImage: My Image 2>]
```

Note: It was mentioned earlier that the *SearchableManager.search()* method returns a Django queryset meaning that you can then chain together further queryset methods onto the result. However when searching across heterogeneous models via an abstract model, this is not the case and the result is a list of model instances.

Also of importance is the `SEARCH_MODEL_CHOICES` setting mentioned above. When searching across heterogeneous models via an abstract model, the models searched will only be used if they are defined within the `SEARCH_MODEL_CHOICES` setting, either explicitly, or implicitly by a model's parent existing in `SEARCH_MODEL_CHOICES`.

1.15.5 Query Behaviour

When a call to *SearchableManager.search()* is performed, the query entered is processed through several steps until it is translated into a Django queryset. By default the query is broken up into keywords, so the query **plans prices projects** would return results that contain any of the words **plans** or **prices** or **projects**.

The query can contain several special operators which allow for this behaviour to be controlled further. Quotes around exact phrases will ensure that the phrase is searched for specifically, for example the query **"plans prices" projects** will return results matching the exact phrase **plans prices** or the word **projects**, in contrast to the previous example.

You can also prefix both words and phrases with + or - symbols. The + symbol will ensure the word or phrase is contained in all results, and the - symbol will ensure that no results will be returned containing the word or phrase. For example the query **+"plans prices" -projects** would return results that must contain the phrase **plans prices** and must not contain the word **projects**.

Once the query has been parsed into words and phrases to be included or excluded, a second step is performed where the query is stripped of common words know as **stop words**. These are common words such as **and**, **the** or **like** that are generally not meaningful and cause irrelevant results to be returned. The list of stop words is stored in the setting *STOP_WORDS* as described in the Configuration section.

1.16 Configuration

Mezzanine provides a central system for defining settings within your project and applications that can then be edited by admin users. The package *mezzanine.conf* contains the models for storing editable settings in the database as well as the functions for registering and loading these settings throughout your project.

1.16.1 Registering Settings

Settings are defined by creating a module named `defaults.py` inside one or more of the applications defined in your project's `INSTALLED_APPS` setting. Inside your `defaults.py` module you then call the function *mezzanine.conf.register_setting()* for each setting you want to define which takes several keyword arguments:

- `name`: The name of the setting.
- `label`: The verbose name of the setting for the admin.
- `description`: The description of the setting.
- `editable`: If `True`, the setting will be editable via the admin.
- `default`: The default value of the setting.
- `choices`: A list of choices the user can select from when the setting is editable.
- `append`: If registering an existing setting, the default value given will be appended to the current.
- `translatable`: If `django-modeltranslation` is activated, this setting will store and display values on a per-language basis.

Note: For settings registered with `editable` as `True`, currently only strings, integers/floats and boolean values are supported for the `default` value.

For example suppose we had a `authors` application and we wanted to create a setting that controls the number of books displayed per author page, we would define the following in `authors.defaults`:

```
from mezzanine.conf import register_setting

register_setting(
    name="AUTHORS_BOOKS_PER_PAGE",
    label="Authors books per page",
    description="The number of books to show per author page.",
    editable=True,
    default=10,
)
```

Note: If you are using Django 1.7 or greater and your app is included in your INSTALLED_APPS as an AppConfig (eg authors.apps.MyCrazyConfig), Mezzanine won't import your defaults.py automatically. Instead you must import it manually in your AppConfig's ready() method.

1.16.2 Reading Settings

Mezzanine provides a settings object via `mezzanine.conf.settings()` in a similar way to Django's `django.conf.settings()`. This settings object contains each of the settings registered above using their names

as attributes. Continuing on from our previous example, suppose we have a view for photos:

```
from django.shortcuts import render
from mezzanine.conf import settings
from .models import Book

def books_view(request):
    books = Book.objects.all()[:settings.AUTHORS_BOOKS_PER_PAGE]
    return render(request, "books.html", {"books": books})
```

When defining editable settings, care should be taken when considering where in your project the setting will be used. For example if a setting is used in a `urlpattern` or the creation of a `model` class it would only be read when your site is first loaded, and therefore having it change at a later point by an admin user would not have any effect without reloading your entire project. In the snippet above, since the settings is being read within a view, the value of the setting being accessed is loaded each time the view is run. This ensures that if the value of the setting has been changed by an admin user it will be reflected on the website.

Note: It's also important to realize that with any settings flagged as editable, defining a value for these in your project's `settings.py` will only serve to provide their default values. Once editable settings are modified via the admin, their values stored in the database will always be used.

1.16.3 Django Settings

Mezzanine's settings object integrates with Django's settings object in a couple of ways.

Firstly it's possible to override the default value for any setting defined using *`mezzanine.conf.register_setting()`* by adding its name and value as a regular setting to your project's settings module. This is especially useful when any of your project's `INSTALLED_APPS` (including Mezzanine itself) register settings that aren't editable and you want to override these settings without modifying the application that registered them.

Secondly it's possible to access any of the settings defined by Django or your project's settings module via Mezzanine's settings object in the same way you would use Django's settings object. This allows for a single access point for all settings regardless of how they are defined.

1.16.4 Default Settings

Mezzanine defines the following settings:

ACCOUNTS_APPROVAL_EMAILS

A comma separated list of email addresses that will receive an email notification each time a new account is created that requires approval.

Default: `''`

ACCOUNTS_APPROVAL_REQUIRED

If `True`, when users create an account, they will not be enabled by default and a staff member will need to activate their account in the admin interface.

Default: `False`

ACCOUNTS_MIN_PASSWORD_LENGTH

Minimum length for passwords

Default: `6`

ACCOUNTS_NO_USERNAME

If `True`, the username field will be excluded from sign up and account update forms.

Default: `False`

ACCOUNTS_PROFILE_FORM_CLASS

Dotted package path and class name of profile form to use for users signing up and updating their profile, when `mezzanine.accounts` is installed.

Default: `'mezzanine.accounts.forms.ProfileForm'`

ACCOUNTS_PROFILE_FORM_EXCLUDE_FIELDS

List of fields to exclude from the profile form.

Default: `()`

ACCOUNTS_PROFILE_MODEL

String in the form *app_label.model_name* for the model used for account profiles.

Default: `None`

ACCOUNTS_PROFILE_VIEWS_ENABLED

If `True`, users will have their own public profile pages.

Default: `False`

ACCOUNTS_VERIFICATION_REQUIRED

If `True`, when users create an account, they will be sent an email with a verification link, which they must click to enable their account.

Default: `False`

ADD_PAGE_ORDER

A sequence of `Page` subclasses in the format `app_label.model_name`, that controls the ordering of items in the select drop-down for adding new pages within the admin page tree interface.

Default: `('pages.RichTextPage',)`

ADMIN_MENU_COLLAPSED

Controls whether or not the left-hand admin menu is collapsed by default.

Default: `False`

ADMIN_MENU_ORDER

Controls the ordering and grouping of the admin menu.

Default: `(('Content', ('pages.Page', 'blog.BlogPost', 'generic.ThreadedComment', ('Media Library', 'media-library'))), ('Site', ('sites.Site', 'redirects.Redirect', 'conf.Setting')), ('Users', ('auth.User', 'auth.Group')))`

ADMIN_REMOVAL

A sequence of Python dotted paths to models (eg: `['mezzanine.blog.models.BlogPost',]`) that should be removed from the admin.

Default: `()`

ADMIN_THUMB_SIZE

Size of thumbnail previews for image fields in the admin interface.

Default: `'24x24'`

AKISMET_API_KEY

Key for http://akismet.com spam filtering service. Used for filtering comments and forms.

Default: `''`

BITLY_ACCESS_TOKEN

Access token for http://bit.ly URL shortening service.

Default: `''`

BLOG_POST_PER_PAGE

Number of blog posts shown on a blog listing page.

Default: `5`

BLOG_RSS_LIMIT

Number of most recent blog posts shown in the RSS feed. Set to `None` to display all blog posts in the RSS feed.

Default: `20`

BLOG_SLUG

Slug of the page object for the blog.

Default: `'blog'`

BLOG_URLS_DATE_FORMAT

A string containing the value `year`, `month`, or `day`, which controls the granularity of the date portion in the URL for each blog post. Eg: `year` will define URLs in the format /blog/yyyy/slug/, while `day` will define URLs with the format /blog/yyyy/mm/dd/slug/. An empty string means the URLs will only use the slug, and not contain any portion of the date at all.

Default: `''`

BLOG_USE_FEATURED_IMAGE

Enable featured images in blog posts

Default: `False`

CACHE_SET_DELAY_SECONDS

Mezzanine's caching uses a technique know as mint caching. This is where the requested expiry for a cache entry is stored with the cache entry in cache, and the real expiry used has the `CACHE_SET_DELAY` added to it. Then on a cache get, the store expiry is checked, and if it has passed, the cache entry is set again, and no entry is returned. This tries to ensure that cache misses never occur, and if many clients were to get a cache miss at once, only one would actually need to re-generated the cache entry.

Default: `30`

COMMENTS_ACCOUNT_REQUIRED

If `True`, users must log in to comment.

Default: `False`

COMMENTS_DEFAULT_APPROVED

If `True`, built-in comments are approved by default.

Default: `True`

COMMENTS_DISQUS_API_PUBLIC_KEY

Public key for http://disqus.com developer API

Default: `''`

COMMENTS_DISQUS_API_SECRET_KEY

Secret key for http://disqus.com developer API

Default: `''`

COMMENTS_DISQUS_SHORTNAME

Shortname for the http://disqus.com comments service.

Default: `''`

COMMENTS_NOTIFICATION_EMAILS

A comma separated list of email addresses that will receive an email notification each time a new comment is posted on the site.

Default: `''`

COMMENTS_NUM_LATEST

Number of latest comments shown in the admin dashboard.

Default: `5`

COMMENTS_REMOVED_VISIBLE

If `True`, comments that have `removed` checked will still be displayed, but replaced with a `removed` message.

Default: `True`

COMMENTS_UNAPPROVED_VISIBLE

If `True`, comments that have `is_public` unchecked will still be displayed, but replaced with a `waiting to be approved` message.

Default: `True`

COMMENTS_USE_RATINGS

If `True`, comments can be rated.

Default: `True`

COMMENT_FILTER

Dotted path to the function to call on a comment's value before it is rendered to the template.

Default: `None`

COMMENT_FORM_CLASS

The form class to use for adding new comments.

Default: `'mezzanine.generic.forms.ThreadedCommentForm'`

DASHBOARD_TAGS

A three item sequence, each containing a sequence of template tags used to render the admin dashboard.

Default: `(('blog_tags.quick_blog', 'mezzanine_tags.app_list'), ('comment_tags.recent_comments',), ('mezzanine_tags.recent_actions',))`

DEVICE_DEFAULT

Device specific template sub-directory to use as the default device.

Default: `''`

DEVICE_USER_AGENTS

Mapping of device specific template sub-directory names to the sequence of strings that may be found in their user agents.

Default: `(('mobile', ('2.0 MMP', '240x320', '400X240', 'AvantGo', 'BlackBerry', 'Blazer', 'Cellphone', 'Danger', 'DoCoMo', 'Elaine/3.0', 'EudoraWeb', 'Googlebot-Mobile', 'hiptop', 'IEMobile', 'KYOCERA/WX310K', 'LG/U990', 'MIDP-2.', 'MMEF20', 'MOT-V', 'NetFront', 'Newt', 'Nintendo Wii', 'Nitro', 'Nokia', 'Opera Mini', 'Palm', 'PlayStation Portable', 'portalmmm', 'Proxinet', 'ProxiNet', 'SHARP-TQ-GX10', 'SHG-i900', 'Small', 'SonyEricsson', 'Symbian OS', 'SymbianOS', 'TS21i-10', 'UP.Browser', 'UP.Link', 'webOS', 'Windows CE', 'WinWAP', 'YahooSeeker/M1A1-R2D2', 'iPhone', 'iPod', 'Android', 'BlackBerry9530', 'LG-TU915 Obigo', 'LGE VX', 'webOS', 'Nokia5800')),)`

EMAIL_FAIL_SILENTLY

If `True`, failures to send email will happen silently, otherwise an exception is raised. Defaults to `settings.DEBUG`.

Default: `False`

EXTRA_MODEL_FIELDS

A sequence of fields that will be injected into Mezzanine's (or any library's) models. Each item in the sequence is a four item sequence. The first two items are the dotted path to the model and its field name to be added, and the dotted path to the field class to use for the field. The third and fourth items are a sequence of positional args and a dictionary of keyword args, to use when creating the field instance. When specifying the field class, the path `django.models.db.` can be omitted for regular Django model fields.

Default: `()`

FORMS_CSV_DELIMITER

Char to use as a field delimiter when exporting form responses as CSV.

Default: `','`

FORMS_EXTRA_FIELDS

Extra field types for the forms app. Should contain a sequence of three-item sequences, each containing the ID, dotted import path for the field class, and field name, for each custom field type. The ID is simply a numeric constant for the field, but cannot be a value already used, so choose a high number such as 100 or greater to avoid conflicts.

Default: `()`

FORMS_EXTRA_WIDGETS

Extra field widgets for the forms app. Should contain a sequence of two-item sequences, each containing an existing ID for a form field, and a dotted import path for the widget class.

Default: `()`

FORMS_FIELD_MAX_LENGTH

Max length allowed for field values in the forms app.

Default: `2000`

FORMS_LABEL_MAX_LENGTH

Max length allowed for field labels in the forms app.

Default: `200`

FORMS_UPLOAD_ROOT

Absolute path for storing file uploads for the forms app.

Default: `''`

FORMS_USE_HTML5

If `True`, website forms will use HTML5 features.

Default: `False`

GOOGLE_ANALYTICS_ID

Google Analytics ID (http://www.google.com/analytics/)

Default: `''`

HOST_THEMES

A sequence mapping host names to themes, allowing different templates to be served per HTTP host. Each item in the sequence is a two item sequence, containing a host such as `othersite.example.com`, and the name of an importable Python package for the theme. If the host is matched for a request, the templates directory inside the theme package will be first searched when loading templates.

Default: `()`

INLINE_EDITING_ENABLED

If `True`, front-end inline editing will be enabled.

Default: `True`

JQUERY_FILENAME

Name of the jQuery file found in mezzanine/core/static/mezzanine/js/

Default: `'jquery-1.8.3.min.js'`

JQUERY_UI_FILENAME

Name of the jQuery UI file found in mezzanine/core/static/mezzanine/js/

Default: `'jquery-ui-1.8.24.min.js'`

MAX_PAGING_LINKS

Max number of paging links to display when paginating.

Default: `10`

MEDIA_LIBRARY_PER_SITE

If `True`, each site will use its own directory within the filebrowser media library.

Default: `False`

NEVERCACHE_KEY

Unique random string like `SECRET_KEY`, but used for two-phased cache responses. Like `SECRET_KEY`, should be automatically generated by the `mezzanine-project` command.

Default: `''`

OWNABLE_MODELS_ALL_EDITABLE

Models that subclass `Ownable` and use the `OwnableAdmin` have their admin change-list records filtered down to records owned by the current user. This setting contains a sequence of models in the format `app_label.object_name`, that when subclassing `Ownable`, will still show all records in the admin change-list interface, regardless of the current user.

Default: `()`

PAGES_PUBLISHED_INCLUDE_LOGIN_REQUIRED

If `True`, pages with `login_required` checked will still be listed in menus and search results, for unauthenticated users. Regardless of this setting, when an unauthenticated user accesses a page with `login_required` checked, they'll be redirected to the login page.

Default: `False`

PAGE_MENU_TEMPLATES

A sequence of templates used by the `page_menu` template tag. Each item in the sequence is a three item sequence, containing a unique ID for the template, a label for the template, and the template path. These templates are then available for selection when editing which menus a page should appear in. Note that if a menu template is used that doesn't appear in this setting, all pages will appear in it.

Default: `((1, 'Top navigation bar', 'pages/menus/dropdown.html'), (2, 'Left-hand tree', 'pages/menus/tree.html'), (3, 'Footer', 'pages/menus/footer.html'))`

PAGE_MENU_TEMPLATES_DEFAULT

A sequence of IDs from the `PAGE_MENU_TEMPLATES` setting that defines the default menu templates selected when creating new pages. By default all menu templates are selected. Set this setting to an empty sequence to have no templates selected by default.

Default: `None`

RATINGS_ACCOUNT_REQUIRED

If `True`, users must log in to rate content such as blog posts and comments.

Default: `False`

RATINGS_RANGE

A sequence of integers that are valid ratings.

Default: `[1, 2, 3, 4, 5]`

RICHTEXT_ALLOWED_ATTRIBUTES

List of HTML attributes that won't be stripped from `RichTextField` instances.

Default: `('abbr', 'accept', 'accept-charset', 'accesskey', 'action', 'align', 'alt', 'axis', 'border', 'cellpadding', 'cellspacing', 'char', 'charoff',`

'charset', 'checked', 'cite', 'class', 'clear', 'cols', 'colspan', 'color', 'compact', 'coords', 'datetime', 'dir', 'disabled', 'enctype', 'for', 'frame', 'headers', 'height', 'href', 'hreflang', 'hspace', 'id', 'ismap', 'label', 'lang', 'longdesc', 'maxlength', 'media', 'method', 'multiple', 'name', 'nohref', 'noshade', 'nowrap', 'prompt', 'readonly', 'rel', 'rev', 'rows', 'rowspan', 'rules', 'scope', 'selected', 'shape', 'size', 'span', 'src', 'start', 'style', 'summary', 'tabindex', 'target', 'title', 'type', 'usemap', 'valign', 'value', 'vspace', 'width', 'xml:lang')

RICHTEXT_ALLOWED_STYLES

List of inline CSS styles that won't be stripped from `RichTextField` instances.

Default: ('border', 'display', 'float', 'list-style-type', 'margin', 'margin-bottom', 'margin-left', 'margin-right', 'margin-top', 'padding-left', 'text-align', 'text-decoration', 'vertical-align')

RICHTEXT_ALLOWED_TAGS

List of HTML tags that won't be stripped from `RichTextField` instances.

Default: ('a', 'abbr', 'acronym', 'address', 'area', 'article', 'aside', 'b', 'bdo', 'big', 'blockquote', 'br', 'button', 'caption', 'center', 'cite', 'code', 'col', 'colgroup', 'dd', 'del', 'dfn', 'dir', 'div', 'dl', 'dt', 'em', 'fieldset', 'figure', 'font', 'footer', 'form', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'header', 'hr', 'i', 'img', 'input', 'ins', 'kbd', 'label', 'legend', 'li', 'map', 'men', 'nav', 'ol', 'optgroup', 'option', 'p', 'pre', 'q', 's', 'samp', 'section', 'select', 'small', 'span', 'strike', 'strong', 'sub', 'sup', 'table', 'tbody', 'td', 'textarea', 'tfoot', 'th', 'thead', 'tr', 'tt', '', 'ul', 'var', 'wbr')

RICHTEXT_FILTERS

List of dotted paths to functions, called in order, on a `RichTextField` value before it is rendered to the template.

Default: `('mezzanine.utils.html.thumbnails',)`

RICHTEXT_FILTER_LEVEL

Do not change this setting unless you know what you're doing.

When content is saved in a Rich Text (WYSIWYG) field, unsafe HTML tags and attributes are stripped from the content to protect against staff members intentionally adding code that could be used to cause problems, such as changing their account to a super-user with full access to the system.

This setting allows you to change the level of filtering that occurs. Setting it to low will allow certain extra tags to be permitted, such as those required for embedding video. While these tags are not the main candidates for users adding malicious code, they are still considered dangerous and could potentially be mis-used by a particularly technical user, and so are filtered out when the filtering level is set to high.

Setting the filtering level to no filtering, will disable all filtering, and allow any code to be entered by staff members, including script tags.

Choices: High: `1`, Low (allows video, iframe, Flash, etc): `2`, No filtering: `3`

Default: `1`

RICHTEXT_WIDGET_CLASS

Dotted package path and class name of the widget to use for the `RichTextField`.

Default: `'mezzanine.core.forms.TinyMceWidget'`

SEARCH_MODEL_CHOICES

Sequence of models that will be provided by default as choices in the search form. Each model should be in the format `app_label.model_name`. Only models that subclass `mezzanine.core.models.Displayable` should be used.

Default: `('pages.Page', 'blog.BlogPost')`

SEARCH_PER_PAGE

Number of results shown in the search results page.

Default: `10`

SITE_PREFIX

A URL prefix for mounting all of Mezzanine's urlpatterns under. When using this, you'll also need to manually apply it to your project's root `urls.py` module. The root `urls.py` module provided by Mezzanine's `mezzanine-project` command contains an example of this towards its end.

Default: `''`

SITE_TAGLINE

A tag line that will appear at the top of all pages.

Default: `'An open source content management platform.'`

SITE_TITLE

Title that will display at the top of the site, and be appended to the content of the HTML title tags on every page.

Default: `'Mezzanine'`

SLUGIFY

Dotted Python path to the callable for converting strings into URL slugs. Defaults to `mezzanine.utils.urls.slugify_unicode` which allows for non-ascii URLs. Change to `django.template.defaultfilters.slugify` to use Django's slugify function, or something of your own if required.

Default: `'mezzanine.utils.urls.slugify_unicode'`

SPAM_FILTERS

Sequence of dotted Python paths to callable functions used for checking posted content (such as forms or comments) is spam. Each function should accept three arguments: the request object, the form object, and the URL that was posted from. Defaults to `mezzanine.utils.views.is_spam_akismet` which will use the http://akismet.com spam filtering service when the `AKISMET_API_KEY` setting is configured.

Default: `('mezzanine.utils.views.is_spam_akismet',)`

SSL_ENABLED

If `True`, users will be automatically redirected to HTTPS for the URLs specified by the `SSL_FORCE_URL_PREFIXES` setting.

Default: `False`

SSL_FORCED_PREFIXES_ONLY

If `True`, only URLs specified by the `SSL_FORCE_URL_PREFIXES` setting will be accessible over SSL, and all other URLs will be redirected back to HTTP if accessed over HTTPS.

Default: `True`

SSL_FORCE_HOST

Host name that the site should always be accessed via that matches the SSL certificate.

Default: `''`

SSL_FORCE_URL_PREFIXES

Sequence of URL prefixes that will be forced to run over SSL when `SSL_ENABLED` is `True`. i.e. ('/admin', '/example') would force all URLs beginning with /admin or /example to run over SSL.

Default: `('/admin', '/account')`

STOP_WORDS

List of words which will be stripped from search queries.

Default:
```
('a', 'about', 'above', 'above', 'across', 'after', 'afterwards',
'again', 'against', 'all', 'almost', 'alone', 'along', 'already', 'also',
'although', 'always', 'am', 'among', 'amongst', 'amoungst', 'amount',
'an', 'and', 'another', 'any', 'anyhow', 'anyone', 'anything', 'anyway',
'anywhere', 'are', 'around', 'as', 'at', 'back', 'be', 'became', 'because',
'become', 'becomes', 'becoming', 'been', 'before', 'beforehand', 'behind',
'being', 'below', 'beside', 'besides', 'between', 'beyond', 'bill', 'both',
'bottom', 'but', 'by', 'call', 'can', 'cannot', 'cant', 'co', 'con', 'could',
'couldnt', 'cry', 'de', 'describe', 'detail', 'do', 'done', 'down', 'due',
'during', 'each', 'eg', 'eight', 'either', 'eleven', 'else', 'elsewhere',
'empty', 'enough', 'etc', 'even', 'ever', 'every', 'everyone', 'everything',
'everywhere', 'except', 'few', 'fifteen', 'fifty', 'fill', 'find', 'fire',
'first', 'five', 'for', 'former', 'formerly', 'forty', 'found', 'four',
```

```
'from', 'front', 'full', 'further', 'get', 'give', 'go', 'had', 'has',
'hasnt', 'have', 'he', 'hence', 'her', 'here', 'hereafter', 'hereby',
'herein', 'hereupon', 'hers', 'herself', 'him', 'himself', 'his', 'how',
'however', 'hundred', 'ie', 'if', 'in', 'inc', 'indeed', 'interest', 'into',
'is', 'it', 'its', 'itself', 'keep', 'last', 'latter', 'latterly', 'least',
'less', 'ltd', 'made', 'many', 'may', 'me', 'meanwhile', 'might', 'mill',
'mine', 'more', 'moreover', 'most', 'mostly', 'move', 'much', 'must', 'my',
'myself', 'name', 'namely', 'neither', 'never', 'nevertheless', 'next',
'nine', 'no', 'nobody', 'none', 'noone', 'nor', 'not', 'nothing', 'now',
'nowhere', 'of', 'off', 'often', 'on', 'once', 'one', 'only', 'onto', 'or',
'other', 'others', 'otherwise', 'our', 'ours', 'ourselves', 'out', 'over',
'own', 'part', 'per', 'perhaps', 'please', 'put', 'rather', 're', 'same',
'see', 'seem', 'seemed', 'seeming', 'seems', 'serious', 'several', 'she',
'should', 'show', 'side', 'since', 'sincere', 'six', 'sixty', 'so', 'some',
'somehow', 'someone', 'something', 'sometime', 'sometimes', 'somewhere',
'still', 'such', 'system', 'take', 'ten', 'than', 'that', 'the', 'their',
'them', 'themselves', 'then', 'thence', 'there', 'thereafter', 'thereby',
'therefore', 'therein', 'thereupon', 'these', 'they', 'thickv', 'thin',
'third', 'this', 'those', 'though', 'three', 'through', 'throughout', 'thr',
'thus', 'to', 'together', 'too', 'top', 'toward', 'towards', 'twelve',
'twenty', 'two', 'un', 'under', 'until', 'up', 'upon', 'us', 'very', 'via',
'was', 'we', 'well', 'were', 'what', 'whatever', 'when', 'whence', 'whenever',
'where', 'whereafter', 'whereas', 'whereby', 'wherein', 'whereupon',
'wherever', 'whether', 'which', 'while', 'whither', 'who', 'whoever', 'whole',
'whom', 'whose', 'why', 'will', 'with', 'within', 'without', 'would', 'yet',
'you', 'your', 'yours', 'yourself', 'yourselves', 'the')
```

TAG_CLOUD_SIZES

Number of different sizes for tags when shown as a cloud.

Default: `4`

TEMPLATE_ACCESSIBLE_SETTINGS

Sequence of setting names available within templates.

Default:
```
('ACCOUNTS_APPROVAL_REQUIRED', 'ACCOUNTS_VERIFICATION_REQUIRED',
'ADMIN_MENU_COLLAPSED', 'BITLY_ACCESS_TOKEN', 'BLOG_USE_FEATURED_IMAGE',
'COMMENTS_DISQUS_SHORTNAME', 'COMMENTS_NUM_LATEST', 'COMMENTS_DISQUS_API_PUBLIC_KEY',
'COMMENTS_DISQUS_API_SECRET_KEY', 'COMMENTS_USE_RATINGS', 'DEV_SERVER',
'FORMS_USE_HTML5', 'GRAPPELLI_INSTALLED', 'GOOGLE_ANALYTICS_ID',
'JQUERY_FILENAME', 'JQUERY_UI_FILENAME', 'LOGIN_URL', 'LOGOUT_URL',
'SITE_TITLE', 'SITE_TAGLINE', 'USE_L10N', 'USE_MODELTRANSLATION')
```

THUMBNAILS_DIR_NAME

Directory name to store thumbnails in, that will be created relative to the original image's directory.

Default: `'.thumbnails'`

TINYMCE_SETUP_JS

URL for the JavaScript file (relative to `STATIC_URL`) that handles configuring TinyMCE when the default `RICHTEXT_WIDGET_CLASS` is used.

Default: `'mezzanine/js/tinymce_setup.js'`

TWITTER_ACCESS_TOKEN_KEY

Default: `''`

TWITTER_ACCESS_TOKEN_SECRET

Default: `''`

TWITTER_CONSUMER_KEY

Default: `''`

TWITTER_CONSUMER_SECRET

Default: `''`

TWITTER_DEFAULT_NUM_TWEETS

Number of tweets to display in the default Twitter feed.

Default: `3`

TWITTER_DEFAULT_QUERY

Twitter query to use for the default query type.

Note: Once you change this from the default, you'll need to configure each of the oAuth consumer/access key/secret settings. Please refer to http://dev.twitter.com for more information on creating an application and acquiring these settings.

Default: `'from:stephen_mcd mezzanine'`

TWITTER_DEFAULT_QUERY_TYPE

Type of query that will be used to retrieve tweets for the default Twitter feed.

Choices: User: `user`, List: `list`, Search: `search`

Default: `'search'`

UPLOAD_TO_HANDLERS

Dict mapping file field names in the format `app_label.model_name.field_name` to the Python dotted path to function names that will be used for the file field's `upload_to` argument.

Default: `{}`

USE_MODELTRANSLATION

If `True`, the django-modeltranslation application will be automatically added to the `INSTALLED_APPS` setting.

Default: `False`

1.17 Importing External Blogs

Mezzanine has the ability to import blog posts from other blogging platforms using a Django management command. These are the currently supported formats and their commands:

- WordPress: `import_wordpress`
- Blogger: `import_blogger`
- Tumblr: `import_tumblr`
- Posterous: `import_posterous`
- RSS: `import_rss`

Each command takes a Mezzanine username to assign the blog posts to as well as certain arguments specific to the blog platform. For example to import an existing Wordpress blog:

```
$ python manage.py import_wordpress --mezzanine-user=username [options]
```

Use the `--help` argument to learn more about the arguments specific to each blog platform's command. For example you can see all options for Wordpress by running:

```
$ python manage.py import_wordpress --help
```

1.17.1 Considerations

There are some known issues with HTML formatting loss - specifically where a heading tag is followed by a paragraph tag or another block HTML element that is not typically enclosed with a `<p>` tag is followed by a paragraph. This depends heavily on the originating platform and how it encodes the blog post's copy. The import processor gets this about 90% correct but you may need to do some quick clean up afterwards.

Generally speaking you shouldn't be able to import your data twice. There is a check in place to either create or update for both comments and posts as they are processed, so even if you run the importer multiple times you should only end up with data imported once. However if you have changed any data this will be overwritten.

1.17.2 Importing from Wordpress

Dependencies

- Mark Pilgrim's feedparser

The first step is to export your Wordpress data. Login to Wordpress and go to `Tools -> Export`. Here you can select your filters, otherwise only published posts will be exported. Once you have saved your export file make a note of the location you saved it to.

Note: It is faster to import directly from your filesystem if you can, especially if you have a large blog with lots of comments.

The next step is to run the `import_wordpress` command where the `url` argument contains the path or URL to your export file:

```
$ python manage.py import_wordpress --mezzanine-user=.. --url=[path|URL]
```

1.17.3 Importing from Blogger

The Blogger import currently has one known limitation which is a maximum of 500 blogs or 500 comments per blog that can be imported. If you have more than this the import will still work but end up being truncated.

Dependencies

- Google's gdata Library

The first step is to obtain your Blogger ID. Login to Blogger and go to `Settings`. You'll see that the address in your browser end with `BlogID=XXX` where `XXX` is your Blogger ID. Make a note of this and while you're in settings, go to `Site Feed` then set `Allow Blog Feeds` to be `Full` - this will give you all your data when you run the import.

The next step is to run the `import_blogger` command where the `blogger-id` argument contains the Blogger ID you retrieved:

```
$ python manage.py import_blogger --mezzanine-user=.. --blogger-id=XXX
```

1.17.4 Importing from Tumblr

Simply run the `import_tumblr` command where the `tumblr-user` argument contains your Tumblr username:

```
$ python manage.py import_blogger --mezzanine-user=.. --tumblr-user=username
```

1.17.5 Importing RSS

Dependencies

- Mark Pilgrim's feedparser

Simply run the `import_rss` command where the `rss-url` argument contains the URL for your RSS feed:

```
$ python manage.py import_rss --mezzanine-user=.. --rss-url=url
```

1.17.6 Importing from Posterous

Dependencies

- Kenneth Reitz's requests

Simply run `import_posterous` command with the right params. You need to get your API key from the Posterous API Reference:

```
$ python manage.py import_posterous --mezzanine-user=.. --api-token=.. --posterous-user=your_posterou
```

If you have more than one blog on your posterous account check out the `-posterous-host` option. Be aware that like the tumblr importer, this leaves your media assets on the Posterous servers. If you're worried about posterous being shut down you may want want to have a closer look at the API to actually export your media.

1.17.7 Importer API - Adding New Importers

The importer system has been designed to be extensible so that import commands can easily be added for other blogging platforms.

Each importer's management command is located in the `mezzanine.blog.management.commands` package, and should have its module named `import_type` where `type` represents the type of import the command is for. This module will then contain a class named `Command` which subclasses *mezzanine.blog.management.base.BaseImporterCommand*.

The first step is to define any custom arguments the command will require using Python's optparse handling.

The main responsbility of the `Command` class is then to implement a *handle_import()* method which handles retrieving blog posts and comments from the particular blogging platform. The *handle_import()* method is passed a dictionary of options for the command. The *add_post()* and *add_comment()* methods should be called inside the *handle_import()* method, adding posts and comments respectively. The *add_post()* method returns a post to be used with the *add_comment()* method. For example:

```
from optparse import make_option
from django.core.management.base import CommandError
from mezzanine.blog.management.base import BaseImporterCommand

class Command(BaseImporterCommand):

    def add_arguments(self, parser):
        super(Command, self).add_arguments(parser)
        parser.add_argument(
            "-s", "--some-arg-name", dest="some_arg_var",
            help="Description of some-arg-name")

    def handle_import(self, options):
        # Perform the tasks that need to occur to retrieve blog posts.
        # We'll use an imaginary "posts" variable that contains a list of
        # post dicts with keys: title, author, pub_date, tags and content.
        # In this example we have access to the command line argument
        # "some-arg-name" via "options["some_arg_var"]".
        for retrieved_post in posts:
            added_post = self.add_post(**retrieved_post)
            # Another imaginary variable to demo the API.
            for retrieved_comment in comments:
                self.add_comment(post=added_post, **retrieved_comment)
```

1.18 Twitter Integration

The *mezzanine.twitter* application exposes the ability to consume, store, and display your own tweets on your site in an efficient manner, as well as the ability to send tweets when publishing new content to the site.

1.18.1 Twitter Feeds

If Twitter feeds are implemented in your templates, a cron job is required that will run the following management command. For example, if we want the tweets to be updated every 10 minutes:

```
*/10 * * * * python path/to/your/site/manage.py poll_twitter
```

This ensures that the data is always available in the site's database when accessed, and allows you to control how often the Twitter API is queried. Note that the Fabric script described earlier includes features for deploying templates for cron jobs, which includes the job for polling Twitter by default.

As of June 2013, Twitter also requires that all API access is authenticated. For this you'll need to configure a Twitter application with OAuth credentials for your site to access the Twitter API. These credentials are configurable as Mezzanine settings. See the Configuration section for more information on these, as well as the Twitter developer site for info on creating an application and configuring your OAuth credentials.

1.18.2 Sending Tweets

When setting up a Twitter application, you'll also be able to configure the permissions your OAuth credentials have against the Twitter API. To consume Twitter feeds, only read permissions are needed, but you may also choose to allow both read and write permissions. With write permissions enabled, you'll then also be able to expose the option in Mezzanine's admin interface for automatically tweeting new blog posts (or your own custom content) when they're published.

To enable this, simply install the *python-twitter <https://pypi.python.org/pypi/python-twitter>* library. With the library installed and credential setttings set, blog posts in the admin will have a "Send to Twitter" checkbox, which when checked, will send a tweet with the post's title and URL.

You can also add this functionality to your own admin classes by making use of `mezzanine.twitter.admin.TweetableAdminMixin`. See *mezzanine.blog.admin.BlogPostAdmin* for an example.

1.19 Packages

Below are auto-generated docs mostly covering each of the packages contained within Mezzanine that are added to `settings.INSTALLED_APPS`.

1.19.1 mezzanine.boot

An app that is forced to the top of the list in `INSTALLED_APPS` for the purpose of hooking into Django's `class_prepared` signal and adding custom fields as defined by the `EXTRA_MODEL_FIELDS` setting. Also patches `django.contrib.admin.site` to use `LazyAdminSite` that defers certains register/unregister calls until `admin.autodiscover` to avoid some timing issues around custom fields not being available when custom admin classes are registered.

mezzanine.boot.**add_extra_model_fields**(*sender*, ***kwargs*)
Injects custom fields onto the given sender model as defined by the EXTRA_MODEL_FIELDS setting. This is a closure over the "fields" variable.

mezzanine.boot.**autodiscover**(**args*, ***kwargs*)
Replaces django's original autodiscover to add a call to LazyAdminSite's lazy_registration.

mezzanine.boot.**import_field**(*field_classpath*)
Imports a field by its dotted class path, prepending "django.db.models" to raw class names and raising an exception if the import fails.

mezzanine.boot.**parse_extra_model_fields**(*extra_model_fields*)
Parses the value of EXTRA_MODEL_FIELDS, grouping the entries by model and instantiating the extra fields. Returns a sequence of tuples of the form (model_key, fields) where model_key is a pair of app_label, model_name and fields is a list of (field_name, field_instance) pairs.

mezzanine.boot.**parse_field_path**(*field_path*)
Take a path to a field like "mezzanine.pages.models.Page.feature_image" and return a model key, which is a tuple of the form ('pages', 'page'), and a field name, e.g. "feature_image".

1.19.2 mezzanine.core

Provides abstract models and admin features used throughout the various Mezzanine apps.

mezzanine.core.models

class mezzanine.core.models.**ContentTyped**(**args*, ***kwargs*)
Mixin for models that can be subclassed to create custom types. In order to use them:

- Inherit model from ContentTyped.
- Call the set_content_model() method in the model's save() method.
- Inherit that model's ModelAdmin from ContentTypesAdmin.
- Include "admin/includes/content_typed_change_list.html" in the

change_list.html template.

get_content_model()
Return content model, or if this is the base class return it.

classmethod get_content_model_name()
Return the name of the OneToOneField django automatically creates for child classes in multi-table inheritance.

classmethod get_content_models()
Return all subclasses of the concrete model.

set_content_model()
Set content_model to the child class's related name, or None if this is the base class.

class mezzanine.core.models.**Displayable**(**args*, ***kwargs*)
Abstract model that provides features of a visible page on the website such as publishing fields. Basis of Mezzanine pages, blog posts, and Cartridge products.

generate_short_url()
Returns a new short URL generated using bit.ly if credentials for the service have been specified.

get_absolute_url()
Raise an error if called on a subclass without get_absolute_url defined, to ensure all search results contains a URL.

get_absolute_url_with_host()
Returns host + get_absolute_url - used by the various short_url mechanics below.

Technically we should use self.site.domain, here, however if we were to invoke the short_url mechanics on a list of data (eg blog post list view), we'd trigger a db query per item. Using current_request should provide the same result, since site related data should only be loaded based on the current host anyway.

get_next_by_publish_date(***kwargs*)
Retrieves next object by publish date.

get_previous_by_publish_date(***kwargs*)
Retrieves previous object by publish date.

publish_date_since()
Returns the time since publish_date.

save(**args*, ***kwargs*)
Set default for publish_date. We can't use auto_now_add on the field as it will be blank when a blog post is created from the quick blog form in the admin dashboard.

set_short_url()
Generates the short_url attribute if the model does not already have one. Used by the set_short_url_for template tag and TweetableAdmin.

If no sharing service is defined (bitly is the one implemented, but others could be by overriding generate_short_url), the SHORT_URL_UNSET marker gets stored in the DB. In this case, short_url is temporarily (eg not persisted) set to host + get_absolute_url - this is so that we don't permanently store get_absolute_url, since it may change over time.

class mezzanine.core.models.**MetaData**(**args*, ***kwargs*)
Abstract model that provides meta data for content.

description_from_content()
Returns the first block or sentence of the first content-like field.

meta_title()
Accessor for the optional _meta_title field, which returns the string version of the instance if not provided.

save(**args*, ***kwargs*)
Set the description field on save.

class mezzanine.core.models.**Orderable**(**args*, ***kwargs*)
Abstract model that provides a custom ordering integer field similar to using Meta's order_with_respect_to, since to date (Django 1.2) this doesn't work with ForeignKey("self"), or with Generic Relations. We may also want this feature for models that aren't ordered with respect to a particular field.

delete(**args*, ***kwargs*)
Update the ordering values for siblings.

get_next_by_order(***kwargs*)
Retrieves next object by order.

get_previous_by_order(***kwargs*)
Retrieves previous object by order.

save(**args*, ***kwargs*)
Set the initial ordering value.

with_respect_to()
Returns a dict to use as a filter for ordering operations containing the original `Meta.order_with_respect_to` value if provided. If the field is a Generic Relation, the dict returned contains names and values for looking up the relation's `ct_field` and `fk_field` attributes.

class mezzanine.core.models.**OrderableBase**
Checks for `order_with_respect_to` on the model's inner `Meta` class and if found, copies it to a custom attribute and deletes it since it will cause errors when used with `ForeignKey("self")`. Also creates the `ordering` attribute on the `Meta` class if not yet provided.

class mezzanine.core.models.**Ownable**(**args*, ***kwargs*)
Abstract model that provides ownership of an object for a user.

is_editable(*request*)
Restrict in-line editing to the objects's owner and superusers.

class mezzanine.core.models.**RichText**(**args*, ***kwargs*)
Provides a Rich Text field for managing general content and making it searchable.

class mezzanine.core.models.**SitePermission**(**args*, ***kwargs*)
Permission relationship between a user and a site that's used instead of `User.is_staff`, for admin and inline-editing access.

class mezzanine.core.models.**SiteRelated**(**args*, ***kwargs*)
Abstract model for all things site-related. Adds a foreignkey to Django's `Site` model, and filters by site with all querysets. See `mezzanine.utils.sites.current_site_id` for implementation details.

save(*update_site=False*, **args*, ***kwargs*)
Set the site to the current site when the record is first created, or the `update_site` argument is explicitly set to `True`.

class mezzanine.core.models.**Slugged**(**args*, ***kwargs*)
Abstract model that handles auto-generating slugs. Each slugged object is also affiliated with a specific site object.

generate_unique_slug()
Create a unique slug by passing the result of get_slug() to utils.urls.unique_slug, which appends an index if necessary.

get_slug()
Allows subclasses to implement their own slug creation logic.

save(**args*, ***kwargs*)
If no slug is provided, generates one before saving.

class mezzanine.core.models.**TimeStamped**(**args*, ***kwargs*)
Provides created and updated timestamps on models.

mezzanine.core.managers

class mezzanine.core.managers.**CurrentSiteManager**(*field_name=None*, **args*, ***kwargs*)
Extends Django's site manager to first look up site by ID stored in the request, the session, then domain for the current request (accessible via threadlocals in `mezzanine.core.request`), the environment variable `MEZZANINE_SITE_ID` (which can be used by management commands with the `--site` arg, finally falling back to `settings.SITE_ID` if none of those match a site.

class mezzanine.core.managers.**DisplayableManager**(*field_name=None*, **args*, ***kwargs*)
Manually combines CurrentSiteManager, PublishedManager and SearchableManager for the Displayable model.

url_map(*for_user=None*, ***kwargs*)
Returns a dictionary of urls mapped to Displayable subclass instances, including a fake homepage instance if none exists. Used in mezzanine.core.sitemaps.

class mezzanine.core.managers.**ManagerDescriptor**(*manager*)
This class exists purely to skip the abstract model check in the __get__ method of Django's ManagerDescriptor.

class mezzanine.core.managers.**PublishedManager**
Provides filter for restricting items returned by status and publish date when the given user is not a staff member.

published(*for_user=None*)
For non-staff users, return items with a published status and whose publish and expiry dates fall before and after the current date when specified.

class mezzanine.core.managers.**SearchableManager**(**args*, ***kwargs*)
Manager providing a chainable queryset. Adapted from http://www.djangosnippets.org/snippets/562/ search method supports spanning across models that subclass the model being used to search.

contribute_to_class(*model*, *name*)
Newer versions of Django explicitly prevent managers being accessed from abstract classes, which is behaviour the search API has always relied on. Here we reinstate it.

get_search_fields()
Returns the search field names mapped to weights as a dict. Used in get_queryset below to tell SearchableQuerySet which search fields to use. Also used by DisplayableAdmin to populate Django admin's search_fields attribute.

Search fields can be populated via SearchableManager.__init__, which then get stored in SearchableManager._search_fields, which serves as an approach for defining an explicit set of fields to be used.

Alternatively and more commonly, search_fields can be defined on models themselves. In this case, we look at the model and all its base classes, and build up the search fields from all of those, so the search fields are implicitly built up from the inheritence chain.

Finally if no search fields have been defined at all, we fall back to any fields that are CharField or TextField instances.

search(**args*, ***kwargs*)
Proxy to queryset's search method for the manager's model and any models that subclass from this manager's model if the model is abstract.

class mezzanine.core.managers.**SearchableQuerySet**(**args*, ***kwargs*)
QuerySet providing main search functionality for SearchableManager.

iterator()
If search has occurred and no ordering has occurred, decorate each result with the number of search terms so that it can be sorted by the number of occurrence of terms.

In the case of search fields that span model relationships, we cannot accurately match occurrences without some very complicated traversal code, which we won't attempt. So in this case, namely when there are no matches for a result (count=0), and search fields contain relationships (double underscores), we assume one match for one of the fields, and use the average weight of all search fields with relationships.

order_by(**field_names*)
Mark the filter as being ordered if search has occurred.

search(*query*, *search_fields=None*)
Build a queryset matching words in the given search query, treating quoted terms as exact phrases and taking into account + and - symbols as modifiers controlling which terms to require and exclude.

mezzanine.core.managers.**search_fields_to_dict**(*fields*)
In SearchableQuerySet and SearchableManager, search fields can either be a sequence, or a dict of fields mapped to weights. This function converts sequences to a dict mapped to even weights, so that we're consistently dealing with a dict of fields mapped to weights, eg: ("title", "content") -> {"title": 1, "content": 1}

mezzanine.core.views

mezzanine.core.views.**direct_to_template**(*request*, *template*, *extra_context=None*, ***kwargs*)
Replacement for Django's direct_to_template that uses TemplateResponse via mezzanine.utils.views.render.

mezzanine.core.views.**displayable_links_js**(*request*)
Renders a list of url/title pairs for all Displayable subclass instances into JSON that's used to populate a list of links in TinyMCE.

mezzanine.core.views.**edit**(*request*, **args*, ***kwargs*)
Process the inline editing form.

mezzanine.core.views.**page_not_found**(*request*, **args*, ***kwargs*)
Mimics Django's 404 handler but with a different template path.

mezzanine.core.views.**search**(*request*, *template=u'search_results.html'*, *extra_context=None*)
Display search results. Takes an optional "contenttype" GET parameter in the form "app-name.ModelName" to limit search results to a single model.

mezzanine.core.views.**server_error**(*request*, **args*, ***kwargs*)
Mimics Django's error handler but adds STATIC_URL to the context.

mezzanine.core.views.**set_device**(*request*, *device=u''*)
Sets a device name in a cookie when a user explicitly wants to go to the site for a particular device (eg mobile).

mezzanine.core.views.**set_site**(*request*, **args*, ***kwargs*)
Put the selected site ID into the session - posted to from the "Select site" drop-down in the header of the admin. The site ID is then used in favour of the current request's domain in mezzanine.core.managers.CurrentSiteManager.

mezzanine.core.views.**static_proxy**(*request*, **args*, ***kwargs*)
Serves TinyMCE plugins inside the inline popups and the uploadify SWF, as these are normally static files, and will break with cross-domain JavaScript errors if STATIC_URL is an external host. URL for the file is passed in via querystring in the inline popup plugin template, and we then attempt to pull out the relative path to the file, so that we can serve it locally via Django.

mezzanine.core.forms

class mezzanine.core.forms.**CheckboxSelectMultiple**(**args*, ***kwargs*)
Wraps render with a CSS class for styling.

class mezzanine.core.forms.**DynamicInlineAdminForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*)
Form for DynamicInlineAdmin that can be collapsed and sorted with drag and drop using OrderWidget.

class mezzanine.core.forms.**Html5Mixin**(**args*, ***kwargs*)
Mixin for form classes. Adds HTML5 features to forms for client side validation by the browser, like a "required" attribute and "email" and "url" input types.

class mezzanine.core.forms.**OrderWidget**(*attrs=None*)
Add up and down arrows for ordering controls next to a hidden form field.

class mezzanine.core.forms.**SplitSelectDateTimeWidget**(*attrs=None*, *date_format=None*, *time_format=None*)
Combines Django's SelectDateTimeWidget and SelectDateWidget.

class mezzanine.core.forms.**TinyMceWidget**(**args*, ***kwargs*)
Setup the JS files and targetting CSS class for a textarea to use TinyMCE.

mezzanine.core.forms.**get_edit_form**(*obj*, *field_names*, *data=None*, *files=None*)
Returns the in-line editing form for editing a single model field.

mezzanine.core.admin

class mezzanine.core.admin.**BaseDynamicInlineAdmin**
Admin inline that uses JS to inject an "Add another" link which when clicked, dynamically reveals another fieldset. Also handles adding the _order field and its widget for models that subclass Orderable.

form
alias of DynamicInlineAdminForm

get_fields(*request*, *obj=None*)
For subclasses of Orderable, the _order field must always be present and be the last field.

get_fieldsets(*request*, *obj=None*)
Same as above, but for fieldsets.

class mezzanine.core.admin.**BaseTranslationModelAdmin**(*model*, *admin_site*)
Abstract class used to handle the switch between translation and no-translation class logic. We define the basic structure for the Media class so we can extend it consistently regardless of whether or not modeltranslation is used.

class mezzanine.core.admin.**DisplayableAdmin**(**args*, ***kwargs*)
Admin class for subclasses of the abstract Displayable model.

check_permission(*request*, *page*, *permission*)
Subclasses can define a custom permission check and raise an exception if False.

save_model(*request*, *obj*, *form*, *change*)
Save model for every language so that field auto-population is done for every each of it.

class mezzanine.core.admin.**OwnableAdmin**(*model*, *admin_site*)
Admin class for models that subclass the abstract Ownable model. Handles limiting the change list to objects owned by the logged in user, as well as setting the owner of newly created objects to the logged in user.

Remember that this will include the `user` field in the required fields for the admin change form which may not be desirable. The best approach to solve this is to define a `fieldsets` attribute that excludes the `user` field or simple add `user` to your admin excludes: `exclude = ('user',)`

get_queryset(*request*)
Filter the change list by currently logged in user if not a superuser. We also skip filtering if the model for this admin class has been added to the sequence in the setting `OWNABLE_MODELS_ALL_EDITABLE`, which contains models in the format `app_label.object_name`, and allows models subclassing `Ownable` to be excluded from filtering, eg: ownership should not imply permission to edit.

save_form(*request*, *form*, *change*)
Set the object's owner as the logged in user.

mezzanine.core.middleware

class mezzanine.core.middleware.**AdminLoginInterfaceSelectorMiddleware**(*get_response=None*)
Checks for a POST from the admin login view and if authentication is successful and the "site" interface is selected, redirect to the site.

class mezzanine.core.middleware.**FetchFromCacheMiddleware**(*get_response=None*)
Request phase for Mezzanine cache middleware. Return a response from cache if found, othwerwise mark the request for updating the cache in `UpdateCacheMiddleware`.

class mezzanine.core.middleware.**RedirectFallbackMiddleware**(**args*, ***kwargs*)
Port of Django's `RedirectFallbackMiddleware` that uses Mezzanine's approach for determining the current site.

class mezzanine.core.middleware.**SSLRedirectMiddleware**(**args*)
Handles redirections required for SSL when `SSL_ENABLED` is `True`.

If `SSL_FORCE_HOST` is `True`, and is not the current host, redirect to it.

Also ensure URLs defined by `SSL_FORCE_URL_PREFIXES` are redirect to HTTPS, and redirect all other URLs to HTTP if on HTTPS.

class mezzanine.core.middleware.**SitePermissionMiddleware**(*get_response=None*)
Marks the current user with a `has_site_permission` which is used in place of `user.is_staff` to achieve per-site staff access.

class mezzanine.core.middleware.**TemplateForDeviceMiddleware**(*get_response=None*)
Inserts device-specific templates to the template list.

class mezzanine.core.middleware.**TemplateForHostMiddleware**(*get_response=None*)
Inserts host-specific templates to the template list.

class mezzanine.core.middleware.**UpdateCacheMiddleware**(*get_response=None*)
Response phase for Mezzanine's cache middleware. Handles caching the response, and then performing the second phase of rendering, for content enclosed by the `nevercache` tag.

mezzanine.core.templatetags.mezzanine_tags

mezzanine.core.templatetags.mezzanine_tags.**admin_app_list**(*request*)
Adopted from `django.contrib.admin.sites.AdminSite.index`. Returns a list of lists of models grouped and ordered according to `mezzanine.conf.ADMIN_MENU_ORDER`. Called from the `admin_dropdown_menu` template tag as well as the `app_list` dashboard widget.

mezzanine.core.templatetags.mezzanine_tags.**admin_dropdown_menu**(*parser*, *token*)
Renders the app list for the admin dropdown menu navigation.

mezzanine.core.templatetags.mezzanine_tags.**app_list**(*parser*, *token*)
Renders the app list for the admin dashboard widget.

mezzanine.core.templatetags.mezzanine_tags.**compress**(*parser*, *token*)
Dummy tag for fallback when django-compressor isn't installed.

mezzanine.core.templatetags.mezzanine_tags.**dashboard_column**(*parser*, *token*)
Takes an index for retrieving the sequence of template tags from `mezzanine.conf.DASHBOARD_TAGS` to render into the admin dashboard.

mezzanine.core.templatetags.mezzanine_tags.**editable**(*parser*, *token*)
Add the required HTML to the parsed content for in-line editing, such as the icon and edit form if the object is deemed to be editable - either it has an `editable` method which returns `True`, or the logged in user has change permissions for the model.

mezzanine.core.templatetags.mezzanine_tags.**editable_loader**(*parser*, *token*)
Set up the required JS/CSS for the in-line editing toolbar and controls.

mezzanine.core.templatetags.mezzanine_tags.**errors_for**(*parser*, *token*)
Renders an alert if the form has any errors.

mezzanine.core.templatetags.mezzanine_tags.**fields_for**(*context*, *form*, *template=u'includes/form_fields.html'*)
Renders fields for a form with an optional template choice.

mezzanine.core.templatetags.mezzanine_tags.**gravatar_url**(*email*, *size=32*)
Return the full URL for a Gravatar given an email hash.

mezzanine.core.templatetags.mezzanine_tags.**ifinstalled**(*parser*, *token*)
Old-style `if` tag that renders contents if the given app is installed. The main use case is:

{% ifinstalled app_name %} {% include "app_name/template.html" %} {% endifinstalled %}

so we need to manually pull out all tokens if the app isn't installed, since if we used a normal `if` tag with a False arg, the include tag will still try and find the template to include.

mezzanine.core.templatetags.mezzanine_tags.**is_installed**(*app_name*)
Returns `True` if the given app name is in the `INSTALLED_APPS` setting.

mezzanine.core.templatetags.mezzanine_tags.**metablock**(*parser*, *token*)
Remove HTML tags, entities and superfluous characters from meta blocks.

mezzanine.core.templatetags.mezzanine_tags.**pagination_for**(*parser*, *token*)
Include the pagination template and data for persisting querystring in pagination links. Can also contain a comma separated string of var names in the current querystring to exclude from the pagination links, via the `exclude_vars` arg.

mezzanine.core.templatetags.mezzanine_tags.**recent_actions**(*parser*, *token*)
Renders the recent actions list for the admin dashboard widget.

mezzanine.core.templatetags.mezzanine_tags.**richtext_filters**(*content*)
Takes a value edited via the WYSIWYG editor, and passes it through each of the functions specified by the RICHTEXT_FILTERS setting.

mezzanine.core.templatetags.mezzanine_tags.**search_form**(*parser*, *token*)
Includes the search form with a list of models to use as choices for filtering the search by. Models should be a string with models in the format `app_label.model_name` separated by spaces. The string `all` can also be used, in which case the models defined by the `SEARCH_MODEL_CHOICES` setting will be used.

mezzanine.core.templatetags.mezzanine_tags.**set_short_url_for**(*parser*, *token*)
Sets the `short_url` attribute of the given model for share links in the template.

mezzanine.core.templatetags.mezzanine_tags.**sort_by**(*items*, *attr*)
General sort filter - sorts by either attribute or key.

mezzanine.core.templatetags.mezzanine_tags.**thumbnail**(*image_url*, *width*, *height*, *upscale=True*, *quality=95*, *left=0.5*, *top=0.5*, *padding=False*, *padding_color=u'#fff'*)
Given the URL to an image, resizes the image using the given width and height on the first time it is requested, and returns the URL to the new resized image. If width or height are zero then original ratio is maintained. When `upscale` is False, images smaller than the given size will not be grown to fill that size. The given width and height thus act as maximum dimensions.

mezzanine.core.templatetags.mezzanine_tags.**translate_url**(*context*, *language*)
Translates the current URL for the given language code, eg:

{% translate_url de %}

mezzanine.core.templatetags.mezzanine_tags.**try_url**(*url_name*)
Mimics Django's `url` template tag but fails silently. Used for url names in admin templates as these won't resolve when admin tests are running.

mezzanine.core.management.commands

mezzanine.core.request

class mezzanine.core.request.**CurrentRequestMiddleware**(*get_response=None*)
Stores the request in the current thread for global access.

mezzanine.core.request.**current_request**()
Retrieves the request from the current thread.

mezzanine.core.tests

1.19.3 mezzanine.pages

Provides the main structure of a Mezzanine site with a hierarchical tree of pages, each subclassing the Page model to create a content structure.

mezzanine.pages.models

class mezzanine.pages.models.**BasePage**(**args*, ***kwargs*)
Exists solely to store `PageManager` as the main manager. If it's defined on `Page`, a concrete model, then each `Page` subclass loses the custom manager.

class mezzanine.pages.models.**Link**(**args*, ***kwargs*)
A general content type for creating external links in the page menu.

class mezzanine.pages.models.**Page**(**args*, ***kwargs*)
A page in the page tree. This is the base class that custom content types need to subclass.

can_add(*request*)
Dynamic `add` permission for content types to override.

can_change(*request*)
Dynamic `change` permission for content types to override.

can_delete(*request*)
Dynamic `delete` permission for content types to override.

can_move(*request*, *new_parent*)
Dynamic `move` permission for content types to override. Controls whether a given page move in the page tree is permitted. When the permission is denied, raises a `PageMoveException` with a single argument (message explaining the reason).

description_from_content()
Override `Displayable.description_from_content` to load the content type subclass for when `save` is called directly on a `Page` instance, so that all fields defined on the subclass are available for generating the description.

get_absolute_url()
URL for a page - for `Link` page types, simply return its slug since these don't have an actual URL pattern. Also handle the special case of the homepage being a page object.

get_ascendants(*for_user=None*)
Returns the ascendants for the page. Ascendants are cached in the `_ascendants` attribute, which is populated when the page is loaded via `Page.objects.with_ascendants_for_slug`.

get_slug()
Recursively build the slug from the chain of parents.

get_template_name()
Subclasses can implement this to provide a template to use in `mezzanine.pages.views.page`.

overridden()
Returns `True` if the page's slug has an explicitly defined urlpattern and is therefore considered to be overridden.

save(**args*, ***kwargs*)
Create the titles field using the titles up the parent chain and set the initial value for ordering.

set_helpers(*context*)
Called from the `page_menu` template tag and assigns a handful of properties based on the current page, that are used within the various types of menus.

set_parent(*new_parent*)
Change the parent of this page, changing this page's slug to match the new parent if necessary.

set_slug(*new_slug*)
Changes this page's slug, and all other pages whose slugs start with this page's slug.

exception `mezzanine.pages.models.`**PageMoveException**(*msg=None*)
Raised by `can_move()` when the move permission is denied. Takes an optinal single argument: a message explaining the denial.

class `mezzanine.pages.models.`**RichTextPage**(**args*, ***kwargs*)
Implements the default type of page with a single Rich Text content field.

mezzanine.pages.views

`mezzanine.pages.views.`**admin_page_ordering**(*request*, **args*, ***kwargs*)
Updates the ordering of pages via AJAX from within the admin.

`mezzanine.pages.views.`**page**(*request*, *slug*, *template=u'pages/page.html'*, *extra_context=None*)
Select a template for a page and render it. The request object should have a `page` attribute that's added via

mezzanine.pages.middleware.PageMiddleware. The page is loaded earlier via middleware to perform various other functions. The urlpattern that maps to this view is a catch-all pattern, in which case the page attribute won't exist, so raise a 404 then.

For template selection, a list of possible templates is built up based on the current page. This list is order from most granular match, starting with a custom template for the exact page, then adding templates based on the page's parent page, that could be used for sections of a site (eg all children of the parent). Finally at the broadest level, a template for the page's content type (it's model class) is checked for, and then if none of these templates match, the default pages/page.html is used.

mezzanine.pages.admin

class mezzanine.pages.admin.**PageAdmin**(**args*, ***kwargs*)

Admin class for the Page model and all subclasses of Page. Handles redirections between admin interfaces for the Page model and its subclasses.

add_view(*request*, ***kwargs*)

For the Page model, redirect to the add view for the first page model, based on the ADD_PAGE_ORDER setting.

change_view(*request*, *object_id*, ***kwargs*)

Enforce custom permissions for the page instance.

check_permission(*request*, *page*, *permission*)

Runs the custom permission check and raises an exception if False.

delete_view(*request*, *object_id*, ***kwargs*)

Enforce custom delete permissions for the page instance.

get_content_models()

Return all Page subclasses that are admin registered, ordered based on the ADD_PAGE_ORDER setting.

response_add(*request*, *obj*)

Enforce page permissions and maintain the parent ID in the querystring.

response_change(*request*, *obj*)

Enforce page permissions and maintain the parent ID in the querystring.

save_model(*request*, *obj*, *form*, *change*)

Set the ID of the parent page if passed in via querystring, and make sure the new slug propagates to all descendant pages.

mezzanine.pages.middleware

class mezzanine.pages.middleware.**PageMiddleware**(**args*, ***kwargs*)

Adds a page to the template context for the current response.

If no page matches the URL, and the view function is not the fall-back page view, we try and find the page with the deepest URL that matches within the current URL, as in this situation, the app's urlpattern is considered to sit "under" a given page, for example the blog page will be used when individual blog posts are viewed. We want the page for things like breadcrumb nav, and page processors, but most importantly so the page's login_required flag can be honoured.

If a page is matched, and the fall-back page view is called, we add the page to the extra_context arg of the page view, which it can then use to choose which template to use.

In either case, we add the page to the response's template context, so that the current page is always available.

classmethod installed()
Used in mezzanine.pages.views.page to ensure PageMiddleware or a subclass has been installed. We cache the result on the PageMiddleware._installed to only run this once. Short path is to just check for the dotted path to PageMiddleware in MIDDLEWARE_CLASSES - if not found, we need to load each middleware class to match a subclass.

process_view(*request*, *view_func*, *view_args*, *view_kwargs*)
Per-request mechanics for the current page object.

mezzanine.pages.templatetags.pages_tags

mezzanine.pages.templatetags.pages_tags.**models_for_pages**(*parser*, *token*)
Create a select list containing each of the models that subclass the Page model.

mezzanine.pages.templatetags.pages_tags.**page_menu**(*parser*, *token*)
Return a list of child pages for the given parent, storing all pages in a dict in the context when first called using parents as keys for retrieval on subsequent recursive calls from the menu template.

mezzanine.pages.templatetags.pages_tags.**set_model_permissions**(*parser*, *token*)
Assigns a permissions dict to the given model, much like Django does with its dashboard app list.

Used within the change list for pages, to implement permission checks for the navigation tree.

mezzanine.pages.templatetags.pages_tags.**set_page_permissions**(*parser*, *token*)
Assigns a permissions dict to the given page instance, combining Django's permission for the page's model and a permission check against the instance itself calling the page's can_add, can_change and can_delete custom methods.

Used within the change list for pages, to implement permission checks for the navigation tree.

mezzanine.pages.page_processors

mezzanine.pages.page_processors.**autodiscover**()
Taken from django.contrib.admin.autodiscover and used to run any calls to the processor_for decorator.

mezzanine.pages.page_processors.**processor_for**(*content_model_or_slug*, *exact_page=False*)
Decorator that registers the decorated function as a page processor for the given content model or slug.

When a page exists that forms the prefix of custom urlpatterns in a project (eg: the blog page and app), the page will be added to the template context. Passing in True for the exact_page arg, will ensure that the page processor is not run in this situation, requiring that the loaded page object is for the exact URL currently being viewed.

1.19.4 mezzanine.generic

Provides various models and associated functionality, that can be related to any other model using generic relationshipswith Django's contenttypes framework, such as comments, keywords/tags and voting.

mezzanine.generic.models

class mezzanine.generic.models.**AssignedKeyword**(**args*, ***kwargs*)
A Keyword assigned to a model instance.

class mezzanine.generic.models.**Keyword**(**args*, ***kwargs*)
Keywords/tags which are managed via a custom JavaScript based widget in the admin.

class mezzanine.generic.models.**Rating**(**args*, ***kwargs*)
A rating that can be given to a piece of content.

save(**args*, ***kwargs*)
Validate that the rating falls between the min and max values.

class mezzanine.generic.models.**ThreadedComment**(**args*, ***kwargs*)
Extend the Comment model from django_comments to add comment threading. Comment provides its own site foreign key, so we can't inherit from SiteRelated in mezzanine.core, and therefore need to set the site on save. CommentManager inherits from Mezzanine's CurrentSiteManager, so everything else site related is already provided.

get_absolute_url()
Use the URL for the comment's content object, with a URL hash appended that references the individual comment.

save(**args*, ***kwargs*)
Set the current site ID, and is_public based on the setting COMMENTS_DEFAULT_APPROVED.

mezzanine.generic.managers

class mezzanine.generic.managers.**CommentManager**(*field_name=None*, **args*, ***kwargs*)
Provides filter for restricting comments that are not approved if COMMENTS_UNAPPROVED_VISIBLE is set to False.

count_queryset()
Called from CommentsField.related_items_changed to store the comment count against an item each time a comment is saved.

visible()
Return the comments that are visible based on the COMMENTS_XXX_VISIBLE settings. When these settings are set to True, the relevant comments are returned that shouldn't be shown, and are given placeholders in the template generic/includes/comment.html.

mezzanine.generic.fields

class mezzanine.generic.fields.**BaseGenericRelation**(**args*, ***kwargs*)
Extends GenericRelation to:

- Add a consistent default value for object_id_field and check for a default_related_model attribute which can be defined on subclasses as a default for the to argument.
- Add one or more custom fields to the model that the relation field is applied to, and then call a related_items_changed method each time related items are saved or deleted, so that a calculated value can be stored against the custom fields since aggregates aren't available for GenericRelation instances.

contribute_to_class(*cls*, *name*)
Add each of the names and fields in the fields attribute to the model the relationship field is applied to, and set up the related item save and delete signals for calling related_items_changed.

related_items_changed(*instance*, *related_manager*)
Can be implemented by subclasses - called whenever the state of related items change, eg they're saved or deleted. The instance for this field and the related manager for the field are passed as arguments.

value_from_object(*obj*)
Returns the value of this field in the given model instance. See: https://code.djangoproject.com/ticket/22552

class mezzanine.generic.fields.**CommentsField**(**args*, ***kwargs*)
Stores the number of comments against the COMMENTS_FIELD_NAME_count field when a comment is saved or deleted.

related_items_changed(*instance*, *related_manager*)
Stores the number of comments. A custom count_filter queryset gets checked for, allowing managers to implement custom count logic.

class mezzanine.generic.fields.**KeywordsField**(**args*, ***kwargs*)
Stores the keywords as a single string into the KEYWORDS_FIELD_NAME_string field for convenient access when searching.

contribute_to_class(*cls*, *name*)
Swap out any reference to KeywordsField with the KEYWORDS_FIELD_string field in search_fields.

formfield(***kwargs*)
Provide the custom form widget for the admin, since there isn't a form field mapped to GenericRelation model fields.

related_items_changed(*instance*, *related_manager*)
Stores the keywords as a single string for searching.

save_form_data(*instance*, *data*)
The KeywordsWidget field will return data as a string of comma separated IDs for the Keyword model - convert these into actual AssignedKeyword instances. Also delete Keyword instances if their last related AssignedKeyword instance is being removed.

class mezzanine.generic.fields.**RatingField**(**args*, ***kwargs*)
Stores the rating count and average against the RATING_FIELD_NAME_count and RATING_FIELD_NAME_average fields when a rating is saved or deleted.

related_items_changed(*instance*, *related_manager*)
Calculates and saves the average rating.

mezzanine.generic.views

mezzanine.generic.views.**admin_keywords_submit**(*request*, **args*, ***kwargs*)
Adds any new given keywords from the custom keywords field in the admin, and returns their IDs for use when saving a model with a keywords field.

mezzanine.generic.views.**comment**(*request*, *template=u'generic/comments.html'*, *extra_context=None*)
Handle a ThreadedCommentForm submission and redirect back to its related object.

mezzanine.generic.views.**initial_validation**(*request*, *prefix*)
Returns the related model instance and post data to use in the comment/rating views below.

Both comments and ratings have a prefix_ACCOUNT_REQUIRED setting. If this is True and the user is unauthenticated, we store their post data in their session, and redirect to login with the view's url (also defined by the prefix arg) as the next param. We can then check the session data once they log in, and complete the action authenticated.

On successful post, we pass the related object and post data back, which may have come from the session, for each of the comments and ratings view functions to deal with as needed.

mezzanine.generic.views.**rating**(*request*)
Handle a RatingForm submission and redirect back to its related object.

mezzanine.generic.forms

class mezzanine.generic.forms.**KeywordsWidget**(*attrs=None*)
Form field for the KeywordsField generic relation field. Since the admin with model forms has no form field for generic relations, this form field provides a single field for managing the keywords. It contains two actual widgets, a text input for entering keywords, and a hidden input that stores the ID of each Keyword instance.

The attached JavaScript adds behaviour so that when the form is submitted, an AJAX post is made that passes the list of keywords in the text input, and returns a list of keyword IDs which are then entered into the hidden input before the form submits. The list of IDs in the hidden input is what is used when retrieving an actual value from the field for the form.

decompress(*value*)
Takes the sequence of AssignedKeyword instances and splits them into lists of keyword IDs and titles each mapping to one of the form field widgets.

format_output(*rendered_widgets*)
Wraps the output HTML with a list of all available Keyword instances that can be clicked on to toggle a keyword.

value_from_datadict(*data*, *files*, *name*)
Return the comma separated list of keyword IDs for use in KeywordsField.save_form_data().

class mezzanine.generic.forms.**RatingForm**(*request*, **args*, ***kwargs*)
Form for a rating. Subclasses CommentSecurityForm to make use of its easy setup for generic relations.

clean()
Check unauthenticated user's cookie as a light check to prevent duplicate votes.

save()
Saves a new rating - authenticated users can update the value if they've previously rated.

mezzanine.generic.admin

class mezzanine.generic.admin.**ThreadedCommentAdmin**(*model*, *admin_site*)
Admin class for comments.

mezzanine.generic.templatetags.comment_tags

mezzanine.generic.templatetags.comment_tags.**comment_filter**(*comment_text*)
Passed comment text to be rendered through the function defined by the COMMENT_FILTER setting. If no function is defined (the default), Django's linebreaksbr and urlize filters are used.

mezzanine.generic.templatetags.comment_tags.**comment_thread**(*parser*, *token*)
Return a list of child comments for the given parent, storing all comments in a dict in the context when first called, using parents as keys for retrieval on subsequent recursive calls from the comments template.

mezzanine.generic.templatetags.comment_tags.**comments_for**(*parser*, *token*)
Provides a generic context variable name for the object that comments are being rendered for.

mezzanine.generic.templatetags.comment_tags.**recent_comments**(*parser*, *token*)
Dashboard widget for displaying recent comments.

mezzanine.generic.templatetags.disqus_tags

mezzanine.generic.templatetags.disqus_tags.**disqus_id_for**(*obj*)
Returns a unique identifier for the object to be used in DISQUS JavaScript.

mezzanine.generic.templatetags.disqus_tags.**disqus_sso_script**(*parser*, *token*)
Provides a generic context variable which adds single-sign-on support to DISQUS if `COMMENTS_DISQUS_API_PUBLIC_KEY` and `COMMENTS_DISQUS_API_SECRET_KEY` are specified.

mezzanine.generic.templatetags.keyword_tags

mezzanine.generic.templatetags.keyword_tags.**keywords_for**(*parser*, *token*)
Return a list of `Keyword` objects for the given model instance or a model class. In the case of a model class, retrieve all keywords for all instances of the model and apply a `weight` attribute that can be used to create a tag cloud.

mezzanine.generic.templatetags.rating_tags

mezzanine.generic.templatetags.rating_tags.**rating_for**(*parser*, *token*)
Provides a generic context variable name for the object that ratings are being rendered for, and the rating form.

1.19.5 mezzanine.blog

Provides a blogging app with posts, keywords, categories and comments. Posts can be listed by month, keyword, category or author.

mezzanine.blog.models

class mezzanine.blog.models.**BlogCategory**(**args*, ***kwargs*)
A category for grouping blog posts into a series.

class mezzanine.blog.models.**BlogPost**(**args*, ***kwargs*)
A blog post.

get_absolute_url()
URLs for blog posts can either be just their slug, or prefixed with a portion of the post's publish date, controlled by the setting `BLOG_URLS_DATE_FORMAT`, which can contain the value `year`, `month`, or `day`. Each of these maps to the name of the corresponding urlpattern, and if defined, we loop through each of these and build up the kwargs for the correct urlpattern. The order which we loop through them is important, since the order goes from least granular (just year) to most granular (year/month/day).

mezzanine.blog.views

mezzanine.blog.views.**blog_post_detail**(*request*, *slug*, *year=None*, *month=None*, *day=None*, *template=u'blog/blog_post_detail.html'*, *extra_context=None*)
. Custom templates are checked for using the name `blog/blog_post_detail_XXX.html` where `XXX` is the blog posts's slug.

mezzanine.blog.views.**blog_post_feed**(*request*, *format*, ***kwargs*)
Blog posts feeds - maps format to the correct feed view.

mezzanine.blog.views.**blog_post_list**(*request*, *tag=None*, *year=None*, *month=None*, *username=None*, *category=None*, *template=u'blog/blog_post_list.html'*, *extra_context=None*)
Display a list of blog posts that are filtered by tag, year, month, author or category. Custom templates are checked for using the name `blog/blog_post_list_XXX.html` where `XXX` is either the category slug or author's username if given.

mezzanine.blog.forms

class mezzanine.blog.forms.**BlogPostForm**
Model form for `BlogPost` that provides the quick blog panel in the admin dashboard.

mezzanine.blog.admin

class mezzanine.blog.admin.**BlogCategoryAdmin**(*model*, *admin_site*)
Admin class for blog categories. Hides itself from the admin menu unless explicitly specified.

has_module_permission(*request*)
Hide from the admin menu unless explicitly set in `ADMIN_MENU_ORDER`.

class mezzanine.blog.admin.**BlogPostAdmin**(**args*, ***kwargs*)
Admin class for blog posts.

save_form(*request*, *form*, *change*)
Super class ordering is important here - user must get saved first.

mezzanine.blog.feeds

class mezzanine.blog.feeds.**PostsAtom**(**args*, ***kwargs*)
Atom feed for all blog posts.

class mezzanine.blog.feeds.**PostsRSS**(**args*, ***kwargs*)
RSS feed for all blog posts.

mezzanine.blog.templatetags.blog_tags

mezzanine.blog.templatetags.blog_tags.**blog_authors**(*parser*, *token*)
Put a list of authors (users) for blog posts into the template context.

mezzanine.blog.templatetags.blog_tags.**blog_categories**(*parser*, *token*)
Put a list of categories for blog posts into the template context.

mezzanine.blog.templatetags.blog_tags.**blog_months**(*parser*, *token*)
Put a list of dates for blog posts into the template context.

mezzanine.blog.templatetags.blog_tags.**blog_recent_posts**(*parser*, *token*)
Put a list of recently published blog posts into the template context. A tag title or slug, category title or slug or author's username can also be specified to filter the recent posts returned.

Usage:

```
{% blog_recent_posts 5 as recent_posts %}
{% blog_recent_posts limit=5 tag="django" as recent_posts %}
{% blog_recent_posts limit=5 category="python" as recent_posts %}
{% blog_recent_posts 5 username=admin as recent_posts %}
```

mezzanine.blog.templatetags.blog_tags.**quick_blog**(*parser*, *token*)
Admin dashboard tag for the quick blog form.

mezzanine.blog.management.base

class mezzanine.blog.management.base.**BaseImporterCommand**(***kwargs*)
Base importer command for blogging platform specific management commands to subclass when importing blog posts into Mezzanine. The `handle_import` method should be overridden to provide the import mechanism specific to the blogging platform being dealt with.

add_comment(*post=None*, *name=None*, *email=None*, *pub_date=None*, *website=None*, *body=None*)
Adds a comment to the post provided.

add_meta(*obj*, *tags*, *prompt*, *verbosity*, *old_url=None*)
Adds tags and a redirect for the given obj, which is a blog post or a page.

add_page(*title=None*, *content=None*, *old_url=None*, *tags=None*, *old_id=None*, *old_parent_id=None*)
Adds a page to the list of pages to be imported - used by the Wordpress importer.

add_post(*title=None*, *content=None*, *old_url=None*, *pub_date=None*, *tags=None*, *categories=None*, *comments=None*)
Adds a post to the post list for processing.

- `title` and `content` are strings for the post.
- `old_url` is a string that a redirect will be created for.
- `pub_date` is assumed to be a `datetime` object.
- `tags` and `categories` are sequences of strings.
- `comments` is a sequence of dicts - each dict should be the return value of `add_comment`.

handle(**args*, ***options*)
Processes the converted data into the Mezzanine database correctly.

Attributes: mezzanine_user: the user to put this data in against date_format: the format the dates are in for posts and comments

handle_import(*options*)
Should be overridden by subclasses - performs the conversion from the originating data source into the lists of posts and comments ready for processing.

trunc(*model*, *prompt*, ***fields*)
Truncates fields values for the given model. Prompts for a new value if truncation occurs.

mezzanine.blog.management.commands

class mezzanine.blog.management.commands.import_rss.**Command**(***kwargs*)
Import an RSS feed into the blog app.

class mezzanine.blog.management.commands.import_blogger.**Command**(***kwargs*)
Implements a Blogger importer. Takes a Blogger ID in order to be able to determine which blog it should point to and harvest the XML from.

handle_import(*options*)
Gets posts from Blogger.

class mezzanine.blog.management.commands.import_wordpress.**Command**(***kwargs*)
Implements a Wordpress importer. Takes a file path or a URL for the Wordpress Extended RSS file.

get_text(*xml*, *name*)
Gets the element's text value from the XML object provided.

handle_import(*options*)
Gets the posts from either the provided URL or the path if it is local.

wp_caption(*post*)
Filters a Wordpress Post for Image Captions and renders to match HTML.

class mezzanine.blog.management.commands.import_tumblr.**Command**(***kwargs*)
Import Tumblr blog posts into the blog app.

mezzanine.blog.management.commands.import_tumblr.**title_from_content**(*content*)
Try and extract the first sentence from a block of test to use as a title.

1.19.6 mezzanine.accounts

Provides features for non-staff user accounts, such as login, signup with optional email verification, password reset, and integration with user profiles models defined by the ACCOUNTS_PROFILE_MODEL setting. Some utility functions for probing the profile model are included below.

mezzanine.accounts.**get_profile_for_user**(*user*)
Returns site-specific profile for this user. Raises ProfileNotConfigured if settings.ACCOUNTS_PROFILE_MODEL is not set, and ImproperlyConfigured if the corresponding model can't be found.

mezzanine.accounts.**get_profile_form**()
Returns the profile form defined by settings.ACCOUNTS_PROFILE_FORM_CLASS.

mezzanine.accounts.**get_profile_model**()
Returns the Mezzanine profile model, defined in settings.ACCOUNTS_PROFILE_MODEL, or None if no profile model is configured.

mezzanine.accounts.**get_profile_user_fieldname**(*profile_model=None*, *user_model=None*)
Returns the name of the first field on the profile model that points to the auth.User model.

mezzanine.accounts.views

mezzanine.accounts.views.**account_redirect**(*request*, **args*, ***kwargs*)
Just gives the URL prefix for accounts an action - redirect to the profile update form.

mezzanine.accounts.views.**login**(*request*, *template=u'accounts/account_login.html'*, *form_class=<class 'mezzanine.accounts.forms.LoginForm'>*, *extra_context=None*)
Login form.

mezzanine.accounts.views.**logout**(*request*)
Log the user out.

mezzanine.accounts.views.**profile**(*request*, *username*, *template=u'accounts/account_profile.html'*, *extra_context=None*)
Display a profile.

mezzanine.accounts.views.**profile_redirect**(*request*, **args*, ***kwargs*)
Just gives the URL prefix for profiles an action - redirect to the logged in user's profile.

mezzanine.accounts.views.**profile_update**(*request*, **args*, ***kwargs*)
Profile update form.

mezzanine.accounts.views.**signup**(*request*, *template=u'accounts/account_signup.html'*, *extra_context=None*)

Signup form.

mezzanine.accounts.views.**signup_verify**(*request*, *uidb36=None*, *token=None*)

View for the link in the verification email sent to a new user when they create an account and ACCOUNTS_VERIFICATION_REQUIRED is set to True. Activates the user and logs them in, redirecting to the URL they tried to access when signing up.

mezzanine.accounts.forms

class mezzanine.accounts.forms.**LoginForm**(**args*, ***kwargs*)

Fields for login.

clean()

Authenticate the given username/email and password. If the fields are valid, store the authenticated user for returning via save().

save()

Just return the authenticated user - used for logging in.

class mezzanine.accounts.forms.**PasswordResetForm**(**args*, ***kwargs*)

Validates the user's username or email for sending a login token for authenticating to change their password.

save()

Just return the authenticated user - used for sending login email.

class mezzanine.accounts.forms.**ProfileForm**(**args*, ***kwargs*)

ModelForm for auth.User - used for signup and profile update. If a Profile model is defined via ACCOUNTS_PROFILE_MODEL, its fields are injected into the form.

clean_email()

Ensure the email address is not already registered.

clean_password2()

Ensure the password fields are equal, and match the minimum length defined by ACCOUNTS_MIN_PASSWORD_LENGTH.

clean_username()

Ensure the username doesn't exist or contain invalid chars. We limit it to slugifiable chars since it's used as the slug for the user's profile view.

save(**args*, ***kwargs*)

Create the new user. If no username is supplied (may be hidden via ACCOUNTS_PROFILE_FORM_EXCLUDE_FIELDS or ACCOUNTS_NO_USERNAME), we generate a unique username, so that if profile pages are enabled, we still have something to use as the profile's slug.

mezzanine.accounts.templatetags.accounts_tags

mezzanine.accounts.templatetags.accounts_tags.**login_form**(*parser*, *token*)

Returns the login form:

{% login_form as form %} {{ form }}

mezzanine.accounts.templatetags.accounts_tags.**profile_fields**(*user*)

Returns profile fields as a dict for the given user. Used in the profile view template when the ACCOUNTS_PROFILE_VIEWS_ENABLED setting is set to True, and also in the account approval emails sent to administrators when the ACCOUNTS_APPROVAL_REQUIRED setting is set to True.

mezzanine.accounts.templatetags.accounts_tags.**profile_form**(*parser*, *token*)
Returns the profile form for a user:

{% if request.user.is_authenticated %} {% profile_form request.user as form %} {{ form }} {% endif %}

mezzanine.accounts.templatetags.accounts_tags.**signup_form**(*parser*, *token*)
Returns the signup form:

{% signup_form as form %} {{ form }}

mezzanine.accounts.templatetags.accounts_tags.**username_or**(*user*, *attr*)
Returns the user's username for display, or an alternate attribute if ACCOUNTS_NO_USERNAME is set to True.

mezzanine.accounts.admin

1.19.7 mezzanine.forms

A port of django-forms-builder for Mezzanine. Allows admin users to create their own HTML5 forms and export form submissions as CSV.

mezzanine.forms.models

class mezzanine.forms.models.**AbstractBaseField**(**args*, ***kwargs*)
A field for a user-built form.

get_choices()
Parse a comma separated choice string into a list of choices taking into account quoted choices.

is_a(**args*)
Helper that returns True if the field's type is given in any arg.

class mezzanine.forms.models.**Field**(*id*, *_order*, *label*, *field_type*, *required*, *visible*, *choices*, *default*, *placeholder_text*, *help_text*, *form*)

class mezzanine.forms.models.**FieldEntry**(**args*, ***kwargs*)
A single field value for a form entry submitted via a user-built form.

class mezzanine.forms.models.**FieldManager**
Only show visible fields when displaying actual form..

class mezzanine.forms.models.**Form**(**args*, ***kwargs*)
A user-built form.

class mezzanine.forms.models.**FormEntry**(**args*, ***kwargs*)
An entry submitted via a user-built form.

mezzanine.forms.forms

class mezzanine.forms.forms.**EntriesForm**(*form*, *request*, **args*, ***kwargs*)
Form with a set of fields dynamically assigned that can be used to filter entries for the given forms.models.Form instance.

columns()
Returns the list of selected column names.

rows(*csv=False*)
Returns each row based on the selected criteria.

class mezzanine.forms.forms.**FormForForm**(*form*, *context*, **args*, ***kwargs*)
Form with a set of fields dynamically assigned, directly based on the given forms.models.Form instance.

email_to()
Return the value entered for the first field of type forms.EmailField.

save(***kwargs*)
Create a FormEntry instance and related FieldEntry instances for each form field.

mezzanine.forms.page_processors

mezzanine.forms.page_processors.**form_processor**(*request*, *page*)
Display a built form and handle submission.

mezzanine.forms.page_processors.**format_value**(*value*)
Convert a list into a comma separated string, for displaying select multiple values in emails.

mezzanine.forms.admin

class mezzanine.forms.admin.**FieldAdmin**(*parent_model*, *admin_site*)
Admin class for the form field. Inherits from TabularDynamicInlineAdmin to add dynamic "Add another" link and drag/drop ordering.

model
alias of Field

class mezzanine.forms.admin.**FormAdmin**(**args*, ***kwargs*)
Admin class for the Form model. Includes the urls & views for exporting form entries as CSV and downloading files uploaded via the forms app.

entries_view(*request*, *form_id*)
Displays the form entries in a HTML table with option to export as CSV file.

file_view(*request*, *field_entry_id*)
Output the file for the requested field entry.

get_urls()
Add the entries view to urls.

1.19.8 mezzanine.galleries

Implements a photo gallery content type.

mezzanine.galleries.models

class mezzanine.galleries.models.**BaseGallery**(**args*, ***kwargs*)
Base gallery functionality.

save(*delete_zip_import=True*, **args*, ***kwargs*)
If a zip file is uploaded, extract any images from it and add them to the gallery, before removing the zip file.

class mezzanine.galleries.models.**Gallery**(**args*, ***kwargs*)
Page bucket for gallery photos.

class mezzanine.galleries.models.**GalleryImage**(*id*, *_order*, *gallery*, *file*, *description*)

save(**args*, ***kwargs*)
If no description is given when created, create one from the file name.

mezzanine.galleries.admin

1.19.9 mezzanine.conf

Drop-in replacement for `django.conf.settings` that provides a consistent access method for settings defined in applications, the project or Django itself. Settings can also be made editable via the admin.

class `mezzanine.conf.`**Settings**
An object that provides settings via dynamic attribute access.

Settings that are registered as editable will be stored in the database once the site settings form in the admin is first saved. When these values are accessed via this settings object, *all* database stored settings get retrieved from the database.

When accessing uneditable settings their default values are used, unless they've been given a value in the project's settings.py module.

The settings object also provides access to Django settings via `django.conf.settings`, in order to provide a consistent method of access for all settings.

class **Placeholder**
A Weakly-referable wrapper of `object`.

`Settings.`**clear_cache**()
Clear the settings cache for the current request.

`Settings.`**use_editable**()
Clear the cache for the current request so that editable settings are fetched from the database on next access. Using editable settings is the default, so this is deprecated in favour of `clear_cache()`.

`mezzanine.conf.`**register_setting**(*name=None*, *label=None*, *editable=False*, *description=None*, *default=None*, *choices=None*, *append=False*, *translatable=False*)
Registers a setting that can be edited via the admin. This mostly equates to storing the given args as a dict in the `registry` dict by name.

mezzanine.conf.models

class `mezzanine.conf.models.`**Setting**(**args*, ***kwargs*)
Stores values for `mezzanine.conf` that can be edited via the admin.

mezzanine.conf.forms

class `mezzanine.conf.forms.`**SettingsForm**(**args*, ***kwargs*)
Form for settings - creates a field for each setting in `mezzanine.conf` that is marked as editable.

format_help(*description*)
Format the setting's description into HTML.

save()
Save each of the settings to the DB.

mezzanine.conf.admin

class mezzanine.conf.admin.**SettingsAdmin**(*model*, *admin_site*)
Admin class for settings model. Redirect add/change views to the list view where a single form is rendered for editing all settings.

mezzanine.conf.context_processors

class mezzanine.conf.context_processors.**TemplateSettings**(*settings*, *allowed_settings*, **args*, ***kwargs*)
Dict wrapper for template settings. This exists to enforce the restriction of settings in templates to those named in TEMPLATE_ACCESSIBLE_SETTINGS, and to warn about deprecated settings.

Django's template system attempts a dict-style index lookup before an attribute lookup when resolving dot notation in template variables, so we use `__getitem__()` this as the primary way of getting at the settings.

mezzanine.conf.context_processors.**settings**(*request=None*)
Add the settings object to the template context.

1.19.10 mezzanine.template

class mezzanine.template.**Library**
Extends `django.template.Library` providing several shortcuts that attempt to take the leg-work out of creating different types of template tags.

as_tag(*tag_func*)
Creates a tag expecting the format: `{% tag_name as var_name %}` The decorated func returns the value that is given to `var_name` in the template.

inclusion_tag(*name*, *context_class=<class 'django.template.context.Context'>*, *takes_context=False*)
Replacement for Django's `inclusion_tag` which looks up device specific templates at render time.

render_tag(*tag_func*)
Creates a tag using the decorated func as the render function for the template tag node. The render function takes two arguments - the template context and the tag token.

to_end_tag(*tag_func*)
Creates a tag that parses until it finds the corresponding end tag, eg: for a tag named `mytag` it will parse until `endmytag`. The decorated func's return value is used to render the parsed content and takes three arguments - the parsed content between the start and end tags, the template context and the tag token.

1.19.11 mezzanine.template.loader_tags

class mezzanine.template.loader_tags.**OverExtendsNode**(*nodelist*, *parent_name*, *template_dirs=None*)
Allows the template `foo/bar.html` to extend `foo/bar.html`, given that there is another version of it that can be loaded. This allows templates to be created in a project that extend their app template counterparts, or even app templates that extend other app templates with the same relative name/path.

We use our own version of `find_template`, that uses an explict list of template directories to search for the template, based on the directories that the known template loaders (`app_directories` and `filesystem`) use. This list gets stored in the template context, and each time a template is found, its absolute path gets removed from the list, so that subsequent searches for the same relative name/path can find parent templates in other directories, which allows circular inheritance to occur.

Django's `app_directories`, `filesystem`, and `cached` loaders are supported. The `eggs` loader, and any loader that implements `load_template_source` with a source string returned, should also theoretically work.

find_template(*name*, *context*, *peeking=False*)
Replacement for Django's `find_template` that uses the current template context to keep track of which template directories it has used when finding a template. This allows multiple templates with the same relative name/path to be discovered, so that circular template inheritance can occur.

get_parent(*context*)
Load the parent template using our own `find_template`, which will cause its absolute path to not be used again. Then peek at the first node, and if its parent arg is the same as the current parent arg, we know circular inheritance is going to occur, in which case we try and find the template again, with the absolute directory removed from the search list.

`mezzanine.template.loader_tags.`**overextends**(*parser*, *token*)
Extended version of Django's `extends` tag that allows circular inheritance to occur, eg a template can both be overridden and extended at once.

1.19.12 mezzanine.twitter

Provides models and utilities for displaying different types of Twitter feeds.

mezzanine.twitter.models

class `mezzanine.twitter.models.`**Query**(*id*, *type*, *value*, *interested*)

run()
Request new tweets from the Twitter API.

class `mezzanine.twitter.models.`**Tweet**(*id*, *remote_id*, *created_at*, *text*, *profile_image_url*, *user_name*, *full_name*, *retweeter_profile_image_url*, *retweeter_user_name*, *retweeter_full_name*, *query*)

mezzanine.twitter.managers

class `mezzanine.twitter.managers.`**TweetManager**
Manager that handles generating the initial `Query` instance for a user, list or search term.

get_for(*query_type*, *value*)
Create a query and run it for the given arg if it doesn't exist, and return the tweets for the query.

mezzanine.twitter.templatetags.twitter_tags

`mezzanine.twitter.templatetags.twitter_tags.`**tweets_default**(*parser*, *token*)
Tweets for the default settings.

`mezzanine.twitter.templatetags.twitter_tags.`**tweets_for**(*query_type*, *args*, *per_user=None*)
Retrieve tweets for a user, list or search term. The optional `per_user` arg limits the number of tweets per user, for example to allow a fair spread of tweets per user for a list.

`mezzanine.twitter.templatetags.twitter_tags.`**tweets_for_list**(*parser*, *token*)
Tweets for a user's list.

mezzanine.twitter.templatetags.twitter_tags.**tweets_for_search**(*parser*, *token*)
Tweets for a search query.

mezzanine.twitter.templatetags.twitter_tags.**tweets_for_user**(*parser*, *token*)
Tweets for a user.

mezzanine.twitter.management.commands

class mezzanine.twitter.management.commands.poll_twitter.**Command**(*stdout=None*, *stderr=None*, *no_color=False*)
Polls the Twitter API for tweets associated to the queries in templates.

1.19.13 mezzanine.utils

Various utility functions used throughout the different Mezzanine apps.

mezzanine.utils.cache.**add_cache_bypass**(*url*)
Adds the current time to the querystring of the URL to force a cache reload. Used for when a form post redirects back to a page that should display updated content, such as new comments or ratings.

mezzanine.utils.cache.**cache_get**(*key*)
Wrapper for `cache.get`. The expiry time for the cache entry is stored with the entry. If the expiry time has past, put the stale entry back into cache, and don't return it to trigger a fake cache miss.

mezzanine.utils.cache.**cache_installed**(**args*, ***kwds*)
Returns `True` if a cache backend is configured, and the cache middleware classes or subclasses thereof are present. This will be evaluated once per run, and then cached.

mezzanine.utils.cache.**cache_key_prefix**(*request*)
Cache key for Mezzanine's cache middleware. Adds the current device and site ID.

mezzanine.utils.cache.**cache_set**(*key*, *value*, *timeout=None*, *refreshed=False*)
Wrapper for `cache.set`. Stores the cache entry packed with the desired cache expiry time. When the entry is retrieved from cache, the packed expiry time is also checked, and if past, the stale cache entry is stored again with an expiry that has `CACHE_SET_DELAY_SECONDS` added to it. In this case the entry is not returned, so that a cache miss occurs and the entry should be set by the caller, but all other callers will still get the stale entry, so no real cache misses ever occur.

mezzanine.utils.cache.**nevercache_token**()
Returns the secret token that delimits content wrapped in the `nevercache` template tag.

class mezzanine.utils.conf.**SitesAllowedHosts**
This is a fallback for Django's ALLOWED_HOSTS setting which is required when DEBUG is False. It looks up the `Site` model and uses any domains added to it, the first time the setting is accessed.

mezzanine.utils.conf.**real_project_name**(*project_name*)
Used to let Mezzanine run from its project template directory, in which case "{{ project_name }}" won't have been replaced by a real project name.

mezzanine.utils.conf.**set_dynamic_settings**(*s*)
Called at the end of the project's settings module, and is passed its globals dict for updating with some final tweaks for settings that generally aren't specified, but can be given some better defaults based on other settings that have been specified. Broken out into its own function so that the code need not be replicated in the settings modules of other project-based apps that leverage Mezzanine's settings module.

mezzanine.utils.device.**device_from_request**(*request*)
Determine's the device name from the request by first looking for an overridding cookie, and if not found then

matching the user agent. Used at both the template level for choosing the template to load and also at the cache level as a cache key prefix.

mezzanine.utils.device.**templates_for_device**(*request*, *templates*)
Given a template name (or list of them), returns the template names as a list, with each name prefixed with the device directory inserted before it's associate default in the list.

Utils called from project_root/docs/conf.py when Sphinx documentation is generated.

mezzanine.utils.docs.**build_changelog**(*docs_path*, *package_name=u'mezzanine'*)
Converts Mercurial commits into a changelog in RST format.

mezzanine.utils.docs.**build_modelgraph**(*docs_path*, *package_name=u'mezzanine'*)
Creates a diagram of all the models for mezzanine and the given package name, generates a smaller version and add it to the docs directory for use in model-graph.rst

mezzanine.utils.docs.**build_requirements**(*docs_path*, *package_name=u'mezzanine'*)
Updates the requirements file with Mezzanine's version number.

mezzanine.utils.docs.**build_settings_docs**(*docs_path*, *prefix=None*)
Converts names, descriptions and defaults for settings in `mezzanine.conf.registry` into RST format for use in docs, optionally filtered by setting names with the given prefix.

mezzanine.utils.docs.**deep_force_unicode**(*value*)
Recursively call force_text on value.

mezzanine.utils.email.**send_approve_mail**(*request*, *user*)
Sends an email to staff in listed in the setting `ACCOUNTS_APPROVAL_EMAILS`, when a new user signs up and the `ACCOUNTS_APPROVAL_REQUIRED` setting is `True`.

mezzanine.utils.email.**send_approved_mail**(*request*, *user*)
Sends an email to a user once their `is_active` status goes from `False` to `True` when the `ACCOUNTS_APPROVAL_REQUIRED` setting is `True`.

mezzanine.utils.email.**send_mail_template**(*subject*, *template*, *addr_from*, *addr_to*, *context=None*, *attachments=None*, *fail_silently=None*, *addr_bcc=None*, *headers=None*)
Send email rendering text and html versions for the specified template name using the context dictionary passed in.

mezzanine.utils.email.**send_verification_mail**(*request*, *user*, *verification_type*)
Sends an email with a verification link to users when `ACCOUNTS_VERIFICATION_REQUIRED` is `'True` and they're signing up, or when they reset a lost password. The `verification_type` arg is both the name of the urlpattern for the verification link, as well as the names of the email templates to use.

mezzanine.utils.email.**split_addresses**(*email_string_list*)
Converts a string containing comma separated email addresses into a list of email addresses.

mezzanine.utils.email.**subject_template**(*template*, *context*)
Loads and renders an email subject template, returning the subject string.

class mezzanine.utils.html.**TagCloser**(*html*)
HTMLParser that closes open tags. Takes a HTML string as its first arg, and populate a `html` attribute on the parser with the original HTML arg and any required closing tags.

mezzanine.utils.html.**absolute_urls**(*html*)
Converts relative URLs into absolute URLs. Used for RSS feeds to provide more complete HTML for item descriptions, but could also be used as a general richtext filter.

mezzanine.utils.html.**decode_entities**(*html*)
Remove HTML entities from a string. Adapted from http://effbot.org/zone/re-sub.htm#unescape-html

mezzanine.utils.html.**escape**(*html*)
Escapes HTML according to the rules defined by the settings RICHTEXT_FILTER_LEVEL, RICHTEXT_ALLOWED_TAGS, RICHTEXT_ALLOWED_ATTRIBUTES, RICHTEXT_ALLOWED_STYLES.

mezzanine.utils.html.**thumbnails**(*html*)
Given a HTML string, converts paths in img tags to thumbnail paths, using Mezzanine's thumbnail template tag. Used as one of the default values in the RICHTEXT_FILTERS setting.

mezzanine.utils.importing.**import_dotted_path**(*path*)
Takes a dotted path to a member name in a module, and returns the member after importing it.

mezzanine.utils.importing.**path_for_import**(*name*)
Returns the directory path for the given package or module.

class mezzanine.utils.models.**AdminThumbMixin**
Provides a thumbnail method on models for admin classes to reference in the list_display definition.

class mezzanine.utils.models.**ModelMixin**
Used as a subclass for mixin models that inject their behaviour onto models defined outside of a project. The subclass should define an inner Meta class with a mixin_for attribute containing the model that will be mixed into.

class mezzanine.utils.models.**ModelMixinBase**
Metaclass for ModelMixin which is used for injecting model fields and methods into models defined outside of a project. This currently isn't used anywhere.

mezzanine.utils.models.**base_concrete_model**(*abstract*, *model*)
Used in methods of abstract models to find the super-most concrete (non abstract) model in the inheritance chain that inherits from the given abstract model. This is so the methods in the abstract model can query data consistently across the correct concrete model.

Consider the following:

```
class Abstract(models.Model)

    class Meta:
        abstract = True

    def concrete(self):
        return base_concrete_model(Abstract, self)

class Super(Abstract):
    pass

class Sub(Super):
    pass

sub = Sub.objects.create()
sub.concrete() # returns Super
```

In actual Mezzanine usage, this allows methods in the Displayable and Orderable abstract models to access the Page instance when instances of custom content types, (eg: models that inherit from Page) need to query the Page model to determine correct values for slug and _order which are only relevant in the context of the Page model and not the model of the custom content type.

mezzanine.utils.models.**get_user_model_name**()
Returns the app_label.object_name string for the user model.

mezzanine.utils.models.**upload_to**(*field_path*, *default*)
Used as the upload_to arg for file fields - allows for custom handlers to be implemented on a per field basis defined by the UPLOAD_TO_HANDLERS setting.

mezzanine.utils.sites.**current_site_id**()
Responsible for determining the current Site instance to use when retrieving data for any SiteRelated models. If we're inside an override_current_site_id context manager, return the overriding site ID. Otherwise, try to determine the site using the following methods in order:

- site_id in session. Used in the admin so that admin users can switch sites and stay on the same domain for the admin.
- The id of the Site object corresponding to the hostname in the current request. This result is cached.
- MEZZANINE_SITE_ID environment variable, so management commands or anything else outside of a request can specify a site.
- SITE_ID setting.

If a current request exists and the current site is not overridden, the site ID is stored on the request object to speed up subsequent calls.

mezzanine.utils.sites.**has_site_permission**(*user*)
Checks if a staff user has staff-level access for the current site. The actual permission lookup occurs in SitePermissionMiddleware which then marks the request with the has_site_permission flag, so that we only query the db once per request, so this function serves as the entry point for everything else to check access. We also fall back to an is_staff check if the middleware is not installed, to ease migration.

mezzanine.utils.sites.**host_theme_path**()
Returns the directory of the theme associated with the given host.

mezzanine.utils.sites.**override_current_site_id**(**args*, ***kwds*)
Context manager that overrides the current site id for code executed within it. Used to access SiteRelated objects outside the current site.

mezzanine.utils.sites.**templates_for_host**(*templates*)
Given a template name (or list of them), returns the template names as a list, with each name prefixed with the device directory inserted into the front of the list.

class mezzanine.utils.tests.**TestCase**(*methodName='runTest'*)
This is the base test case providing common features for all tests across the different apps in Mezzanine.

create_recursive_objects(*model*, *parent_field*, ***kwargs*)
Create multiple levels of recursive objects.

queries_used_for_template(*template*, ***context*)
Return the number of queries used when rendering a template string.

setUp()
Creates an admin user, sets up the debug cursor, so that we can track the number of queries used in various places, and creates a request factory for views testing.

tearDown()
Clean up the admin user created and debug cursor.

mezzanine.utils.tests.**copy_test_to_media**(*module*, *name*)
Copies a file from Mezzanine's test data path to MEDIA_ROOT. Used in tests and demo fixtures.

mezzanine.utils.tests.**run_pep8_for_package**(*package_name*, *extra_ignore=None*)
If pep8 is installed, run it across the given package name returning any warnings or errors found.

mezzanine.utils.tests.**run_pyflakes_for_package**(*package_name*, *extra_ignore=None*)
If pyflakes is installed, run it across the given package name returning any warnings found.

mezzanine.utils.timezone.**get_best_local_timezone**()
Compares local timezone offset to pytz's timezone db, to determine a matching timezone name to use when TIME_ZONE is not set.

mezzanine.utils.urls.**admin_url**(*model*, *url*, *object_id=None*)
Returns the URL for the given model and admin url name.

mezzanine.utils.urls.**clean_slashes**(*path*)
Canonicalize path by removing leading slashes and conditionally removing trailing slashes.

mezzanine.utils.urls.**home_slug**()
Returns the slug arg defined for the `home` urlpattern, which is the definitive source of the `url` field defined for an editable homepage object.

mezzanine.utils.urls.**login_redirect**(*request*)
Returns the redirect response for login/signup. Favors: - next param - LOGIN_REDIRECT_URL setting - homepage

mezzanine.utils.urls.**next_url**(*request*)
Returns URL to redirect to from the `next` param in the request.

mezzanine.utils.urls.**path_to_slug**(*path*)
Removes everything from the given URL path, including language code and `PAGES_SLUG` if any is set, returning a slug that would match a `Page` instance's slug.

mezzanine.utils.urls.**slugify**(*s*)
Loads the callable defined by the `SLUGIFY` setting, which defaults to the `slugify_unicode` function.

mezzanine.utils.urls.**slugify_unicode**(*s*)
Replacement for Django's slugify which allows unicode chars in slugs, for URLs in Chinese, Russian, etc. Adopted from https://github.com/mozilla/unicode-slugify/

mezzanine.utils.urls.**unique_slug**(*queryset*, *slug_field*, *slug*)
Ensures a slug is unique for the given queryset, appending an integer to its end until the slug is unique.

mezzanine.utils.views.**ip_for_request**(*request*)
Returns ip address for request - first checks `HTTP_X_FORWARDED_FOR` header, since app will generally be behind a public web server.

mezzanine.utils.views.**is_editable**(*obj*, *request*)
Returns `True` if the object is editable for the request. First check for a custom `editable` handler on the object, otherwise use the logged in user and check change permissions for the object's model.

mezzanine.utils.views.**is_spam**(*request*, *form*, *url*)
Main entry point for spam handling - called from the comment view and page processor for `mezzanine.forms`, to check if posted content is spam. Spam filters are configured via the `SPAM_FILTERS` setting.

mezzanine.utils.views.**is_spam_akismet**(*request*, *form*, *url*)
Identifies form data as being spam, using the http://akismet.com service. The Akismet API key should be specified in the `AKISMET_API_KEY` setting. This function is the default spam handler defined in the `SPAM_FILTERS` setting.

The name, email, url and comment fields are all guessed from the form fields:

- name: First field labelled "Name", also taking i18n into account.
- email: First `EmailField` field.
- url: First `URLField` field.
- comment: First field with a `Textarea` widget.

If the actual comment can't be extracted, spam checking is passed.

The referrer field expects a hidden form field to pass the referrer through, since the HTTP_REFERER will be the URL the form is posted from. The hidden referrer field is made available by default with the `{% fields_for %}` templatetag used for rendering form fields.

`mezzanine.utils.views.`**`paginate`**(*objects*, *page_num*, *per_page*, *max_paging_links*)
Return a paginated page for the given objects, giving it a custom `visible_page_range` attribute calculated from `max_paging_links`.

`mezzanine.utils.views.`**`render`**(*request*, *templates*, *dictionary=None*, *context_instance=None*, ***kwargs*)
Mimics `django.shortcuts.render` but uses a TemplateResponse for `mezzanine.core.middleware.TemplateForDeviceMiddleware`

`mezzanine.utils.views.`**`set_cookie`**(*response*, *name*, *value*, *expiry_seconds=None*, *secure=False*)
Set cookie wrapper that allows number of seconds to be given as the expiry time, and ensures values are correctly encoded.

1.20 Colophon

1.20.1 Authors

- Stephen McDonald
- Lex Hider
- Van Lindberg
- Timur Bobrus
- Toby White
- Eric Floehr
- Tom von Schwerdtner
- Brad Montgomery
- Andrew Fisher
- Carlos David Marrero
- Lee Matos
- Josh de Blank
- Dominique Guardiola Falco
- Michał Oleniec
- John Campbell
- Andrew Grigorev
- Audrey Roy
- Josh Cartmell
- Osiloke Emoekpere
- Eduardo Gutierrez
- Rich Atkinson
- Brett Clouser

- Brent Hoover
- Owen Nelson
- Zeke Harris
- Ken Bolton
- Eli Spizzichino
- Michael Delaney
- David Prusaczyk
- Alexey Makarenya
- Sebastián Ramírez Magrí
- Kevin Levenstein
- Josh Batchelor
- John Barham
- Luke Plant
- Zdeněk Softič
- Alvin Mites
- Jason Kowaleski
- Nicola Larosa
- Anders Hofstee
- Tommy Wolber
- Chris Trengove
- Chris Smith
- Patrick Taylor
- Paolo Dinay
- Nicolas Perriault
- Aleksandr Vladimirskiy
- Thomas Wajs
- Arsenio Santos
- Dmitry Falk
- Brian Schott
- Gary Reynolds
- Maxim Sukharev
- Anton Sutton
- Kent Hauser
- Renyi Khor
- Van Nguyen
- Thomas Lockhart

- Pavel Ponomarev
- Ross Laird
- Alex Hill
- Zachary Gohr
- Edita Menclová
- Jaffa McNeill
- Kristjan Schmidt
- Yong Choi
- Milorad Pop-Tosic
- Rivo Zängov
- Vincent Rialland
- Martin Jahn
- Olav Lindekleiv
- Christopher R. Parr
- Hilton Medeiros
- Yassine Elassad
- Armadillo Integración Tecnológica C.A
- Sergi Almacellas Abellana
- Enrico Tröger
- Sanjay B
- Adam Brenecki
- James Page
- Hakan Bakkalbasi
- Isaac Bythewood
- Lorin Hochstein
- Aaron Merriam
- Pedro Miguel Correia Araújo
- Kevin London
- David Novakovic
- Mark Mukherjee
- Eduardo Rivas
- Kenneth Falck
- Zean Tsoi
- Robert Moggach
- Artem Ploujnikov
- Sean Voss

- Stefan Hummert
- Penny Leach
- Andrey Shipilov
- Andre Graf
- Per Andersson
- Ulrich Wagner
- Ahmad Khayyat
- Ivan Teoh
- Thomas Jetzinger
- Grant Warren-Robertson
- Doug Evenhouse
- Matt Stevenson
- Olivier Harris
- Churkin Oleg
- Chris F Ravenscroft
- Kenneth Love
- Gavin Wahl
- Rocky Meza
- Jonathan Potter
- David K. Hess
- skooch
- Li Yinhui
- Jackson Gothe-Snape
- Stian Prestholdt
- Wojtek Ruszczewski
- Ben Wilson
- Mahdi Bornazadeh
- Travis Nickles
- Bryden Frizzell
- Jesus Armando Anaya Orozco
- Pahaz Blinov
- Mahdi Bornazadeh
- David Lawrence
- Basil Mironenko
- Dmitry Belaventcev
- Thejaswi Puthraya

- Sachin Shende
- Sam Kingston
- José Aliste
- Marcos Scriven
- Gabe Smedresman
- Kim Tore Jensen
- Mike Wakerly
- Jeff Fein-Worton
- Petr Papoušek
- Andrey Zhukov
- Alexandre Hajjar
- Breno Uchoa
- Nar Kumar
- Tim Slot
- Andromeda Yelton
- John Groszko
- Jeremie Ferry
- Eduardo S. Klein
- Jason Wong
- Romain Hardouin
- Ling Thio
- Tim Valenta
- Artem Gluvchynsky
- Dheeraj Sayala
- Antoine Catton
- Marek Wywiał
- Vinod Kurup
- Ethan Goldstine
- Henri Koivuneva
- Mehmet Özgür Bayhan
- Thomas Rega
- Deric Crago
- Cristian Ciupitu
- Danny Sag
- Troy Harvey
- Ahmet Bakan

- Ben Ledbury
- Nicole Harris
- David Winterbottom
- David Higgins
- hanchen
- John Henry
- Cornel K
- Tuk Bredsdorff
- Simon Griffee
- Markus Törnqvist
- Alyssa Welles
- Tulio Nobrega
- Ed Schofield
- Sebastian Clemens
- Alexandre Hajjar
- Zachery Metcalf
- Tim Harton
- Daniel Lawrence
- Leo Zhu
- Hervé Cauwelier
- Adrian Carpenter
- Tye Scott
- David Tomaschik
- Denis Cornehl
- Luiz Felipe Schleder
- Neum Schmickrath
- David Sanders
- Sylvain Fankhauser
- Laurent Prodon
- Simone Federici
- Roberto Macho
- Alejandro Peralta
- Venelin Stoykov
- Samir Shah
- Arnold Krille
- Federico Bruni

- Jeff Pittman
- Philip Mateescu
- Tzu-ping Chung
- Ziwei Zhou
- Dominique Bischof
- Bryan Clement
- Mario Rosa
- Sergey Maranchuk
- Dovydas Stepanavicius
- Luc Milland
- Melvyn Sopacua
- Tiantian Gao
- Eric Valadas
- Douglas Fraser
- Mike Cornwell
- Simone Dalla
- Eino Mäkitalo
- River Satya
- Maciej Szulik
- Andreas Fleig
- Rivo Laks
- Baylee Feore
- Dustin Broderick
- Robert Zywucki
- Kelvin Wong
- William Scales
- Sean Hussey
- Stone Lasley
- Krzysztof Szumny
- Avery Laird
- Christian Abbott
- Hagan Franks
- Souren Araya
- Ryan Sadwick
- Fábio C. Barrionuevo da Luz
- Mathias Ettinger

- Scott Clark
- Frankie Robertson
- Alex Bendig
- Pavan Rikhi
- Paul Hunt
- Jan Varho
- Christian Wiegand
- Simen Heggestøyl
- Alex Tsai
- Darius Daftary
- Graham Oliver
- Johannes Ammon
- Michael Best
- Google, Inc.
- Nik Nyby
- Stuart Dines
- Gerald Hien
- Alexander Bliskovsky
- Ryne Everett
- Firsov Dmytriy
- Fygul Hether
- Anna Wiggins
- Ha Pham
- Pirave Eahalaivan
- Julian Andrews
- Christian Hill
- Henri Hulski
- Tomas Chmelevskij
- Daniel Blasco
- Florent D'halluin
- Douglas Kastle
- Kristiyan Kostadinov
- Jura Studenkov
- Jeff Cook
- Andrew Cassidy
- Vladir Parrado Cruz

- Adam Venturella
- Wim Feijen
- Martín Gaitán
- Alisson Patricio
- Geoffrey Royer

1.20.2 License

1.20.3 Change Log

Version 4.2.2 (Sep 25, 2016)

- Add an option for turning off the runserver banner. Add an option to Mezzanine's runserver command for not showing the. banner at startup - Simen Heggestøyl
- Only set `content_model` first time. Fix #1676 - ryneeverett

Version 4.2.1 (Sep 19, 2016)

- Stop using deprecated template settings. - `TEMPLATE_DIRS`. - `TEMPLATE_LOADERS`. I think these must have been actually removed in `django-1.10.0` because. they're throwing AttributeError's now - ryneeverett
- Remove deprecated `richtext_filter` tag. I don't see how the fallback could even work without the user defining. `RICHTEXT_FILTERS` as empty in their own settings, because it defaults to. *("mezzanine.utils.html.thumbnails",)* so the *if not filternames* path. would never be taken. Give that the fallback does nothing, I think. printing a warning is deceptive so better just to remove it entirely - ryneeverett
- Fixed crash in `Page.get_ascendants` with non-current site - Alex Hill
- Add `override_current_site_id` context manager - Alex Hill

- Make sure a thread-local request is set in PagesTests. We rely in these tests on a "current request" being stored as a. thread-local, but without explicitly assigning one we were dependent on. other tests that make requests being executed before these - Alex Hill
- Exclude `.pyc`, `.pyo` and `__pycache__` files from release - Edward Betts
- Correctly handle custom email fields in `mezzanine.forms`. The *is_a* method checks whether a field is one of Mezzanine's built-in. form fields. As far as I can tell, every other usage is for the purpose. of initializing the built-in fields and widgets, so it makes sense to. exclude user-defined fields. However in this instance, we only want to know whether the field is an. EmailField, not that it is *the* built-in Mezzanine email field. Therefore, check the class rather than identity. The effective change here is that user-defined fields (in. `FORMS_EXTRA_FIELDS)` which subclass django's EmailField will be able to. receive confirmation emails - ryneeverett
- Remove redundant assignment of `request.site_id` - Alex Hill
- Add compatibility with the Django 1.10 MIDDLEWARE setting - Samir Shah
- Add more docstrings to ContentTyped - ryneeverett
- Set base ContentTyped `content_model` to None. Per discussion with @AlexHill in cartridge#293, `content_model` is the. name of django's automatic OneToOneField's linking the concrete base. class inheriting from ContentTyped to it's subclasses. There is no such. field when the base class itself is instantiated, so the `content_model`. should be None - ryneeverett
- `ContentTyped.get_content_model()` should return instance, not class - Alex Hill
- Test admin redirects for ContentTyped models - Alex Hill
- Small error documentation. I think there are a small error : fields (plural) not field - Anthony
- Add some MIDDLEWARE compatibility fixes that were missing from #1660. Also change the way in which the middleware setting is determined. A statis variable will fail when settings are modified on the fly, `e.g.`, when running tests - Samir Shah
- Fix positional argument count on `get_db_prep_value` - Jeff Cook
- Restore context vars in `admin_dropdown_menu` templatetag. Closes #1669 - Stephen McDonald
- Fix request context for form entries in admin - Stephen McDonald
- Fix template for form entries in admin - Stephen McDonald
- Fix overextends tag with cached template loader - Alex Hill
- Deprecate overextends tag since Django now includes its functionality - Alex Hill
- Remove some obsolete checks for Django < 1.8 - Alex Hill
- Deprecate SSLRedirectMiddleware. Resolve #1600. Also, set default SSL settings to *editable=False* so they do not. display in the settings by default - ryneeverett
- Move most Field functionality into abstract base class - Alex Hill
- Escape comment usernames in admin list - Stephen McDonald
- Move richtext filtering into a util function - Stephen McDonald

Version 4.2.0 (Aug 12, 2016)

- Update collecttemplates command for 4.1. Closes #1512 - Stephen McDonald
- Return correct HTTP error response if static proxy cannot find requested file - Stephen McDonald
- Fix dummy homepage object in sitemap. Closes #1516 - Stephen McDonald

- Fixes #1449 – duplicate profile fields in admin - Alex Hill
- Move template form CSS rule to grappelli-safe - Alex Hill
- Removed nesting of Context objects. Inclusion tags should: * Return dictionaries, not Context objects. * Explicitly pass all needed data to their templates. * Not modify passed in Context object ideally. No code should wrap Context objects in further Context objects, as this. causes errors in some contexts - Luke Plant
- Fix blog pagination due to Django 1.9 change - Stephen McDonald
- Don't load user model at import time in tests - Stephen McDonald
- Move misc admin related urlpatterns under /admin/ for better compatibility with SSLRedirectMiddleware. Closes #1365 - Stephen McDonald
- Fix user creation with required profile fields - Alex Hill
- Removed syncdb from deploy as it is no longer supported in `Django1.9` and has been depricated since 1.7, migrate is used instead - Douglas Kastle
- Use Django's native dynamic inline mechanism - Alex Hill
- Remove custom dynamic inline templates - Alex Hill
- Support with i18n urlpatterns with `SSL_FORCE_URL_PREFIXES`. Closes #1456 - Stephen McDonald
- Restore the ordering-related parts of `dynamic_inline.js`. We still need to use `dynamic_inline.js` for its custom ordering code, but we can remove everything related to dynamically adding and removing inline forms since we're now using Django's built-in mechanisms for that - Alex Hill
- Remove now-unused dynamic inline templates - Alex Hill
- Update Python/Django versions in documentation - Stephen McDonald
- Accept either CDATA or text nodes for WordPress comments. ,. All text output is CDATA in an export from WordPress 4.4.2. It is hoped. that accepting either text or CDATA as node type will be compatible. with exports from all versions of WordPress - Jeff Cook
- Changed proxy to resolve mime types using the mimetypes library for python - Andrew Cassidy
- Google Chrome seems to interpret an empty response as. <html><body></body></html> so forced `content_type` to text/plain to. prevent Chrome handling it in this way - Andrew Cassidy
- Changed free themes url. I have changed the URL for the free themes - thecodinghouse
- Fix help text for Link page URL. The field is required, but the inherited help text says it's optional - Gavin Wahl
- Add Coveralls coverage testing to CI, per #1012 - ryneeverett
- Clear `TEST_COLLATION` deprecation warning - David Sanders
- Add Federation of Egalitarian Communities Website - Pavan Rikhi
- Only autofocus visible fields with Html5Mixin - David Sanders
- Ensure csrftoken cookie is set inside nevercache tag - David Sanders
- Remove check for CSRF token before re-running CSRF middleware - David Sanders
- Add testcase for nevercache and CSRF cookie - David Sanders
- Response should be marked as CSRF processing not done, not request - David Sanders
- Fix unicode errors in user agent check for devices. Closes #1589 - Stephen McDonald
- `ModelAdmin.in_menu -> has_module_permission`. Deprecate *ModelAdmin.in_menu* now that django has an equivalent method,. *ModelAdmin.has_module_permission*. See.

<https://docs.djangoproject.com/en/1.9/ref/contrib/admin/#django.contrib.admin.ModelAd
- ryneeverett

- Replace undocumented `ModelAdmin.get_model_perms`. Instead use the documented methods *ModelAdmin.has_change_permission*. and *ModelAdmin.has_add_permission* - ryneeverett
- Fix tests for pyflakes 1.2.x. The warning message now includes the module name - ryneeverett
- Fix selector of the `_order` field to make stacked inlines sortable - Eduardo Rivas
- Catch TinyMCE errors when trying to save inlines. Previously this used to prevent the new `_order` values from being computed - Eduardo Rivas
- Annotate special handling of the `_order` field in inlines - Eduardo Rivas
- Add CSS to hide the `_order` field in stacked inlines - Eduardo Rivas
- Constrain content images to max width in default templates - Stephen McDonald
- un-urlencode thumbnail filenames. Closes #1580 - Stephen McDonald
- Clarify format of `ADMIN_REMOVAL` setting - Stephen McDonald
- Fix: Add missing space - cspollar
- Fix file name decoding from zip files in python 3. In python3, non-ascii filenames in galleries are incorrectly decoded,. interpreting utf8 code points as box-drawing characters. For example, in. the demo project "Avila Spain" is incorrectly parsed as "Aüvila Spain". CP437 is a superset of ascii and the de facto standard for file names. Obviously not every valid utf-8 character is in this character set, but. a lot of tooling does not support file names with characters outside. this set anyway. If we were to encode them in a broader character set I. suspect we would get into OS-interoperability issues, so better to. forego encoding them and coerce them into valid file names. Note that this changes the behavior such that in python3, file names are. now decoded with a chardet-detected encoding. It's also notable that the latest release of chardet incorrectly. identifies the encoding, so in the demo galleries, "Avila Spain" is. incorrectly parsed as "AĔvila Spain". This is fixed in chardet master - ryneeverett
- `base_concrete_model`: accept model class argument. Previously only model instances were accepted but now model classes can. be passed alternatively - ryneeverett
- Factor custom content types out of Page. Custom content types are now implemented as Model and ModelAdmin mixins. This way they can be reused by Cartridge's Product model - ryneeverett
- Add `content_typed/change_list.html` include. This eliminates template duplication for the content type selector in. cartridge. Note that all jQuery events are propagated unless one of them returns. *false*, so both of our *addlist* change handlers get called. `http://stackoverflow.com/questions/4379403/jquery-event-handlers-return-values` - ryneeverett
- Append default `TEMPLATE_ACCESSIBLE_SETTINGS`. This means users don't have to copy the defaults into their settings and. are protected from future changes to the settings used by internal. mezzanine templates - ryneeverett
- Warn when unallowed template settings are used. Since the exception is suppressed, give a hint that the template. setting isn't allowed - ryneeverett
- dynamically set current rating if it exists - Martín Gaitán
- Remove redundant slash in urlpatterns when homepage is the blog - Stephen McDonald
- Django 1.10 removed `LOGOUT_URL`, so provide a default - Stephen McDonald
- Replace Django's deprecated `AUTH_PROFILE_MODULE` setting with new `ACCOUNTS_PROFILE_MODEL` setting - Stephen McDonald

- Remove use of NoArgsCommand, which Django 1.10 removes - Stephen McDonald
- Remove `content_typed` namespace. - `content_typed.py` -> `models.py` + `admin.py`. - `content_typed/change_list.html` ->. `admin/includes/content_typed_change_list.html` - ryneeverett
- Restore guards against back button in page tree / content typed JS - Stephen McDonald
- Fix encoding in blog feeds. Closes #1461 - Stephen McDonald
- Add form media to Form pages. My use case is adding form assets to a widget used by a field included. in the `FORMS_EXTRA_FIELDS` setting. I don't think one should have to. override this template to do this and overextension doesn't seem to work. on content-typed templates - ryneeverett
- Fail gracefully on third-party admin classes that do odd things. Closes #1628 - Stephen McDonald
- Prevent bleach from stripping 'tel:' hrefs in HTML - Stephen McDonald
- Fix #1438 – allow multiple comment forms on a page - Alex Hill
- Use `call_command` instead calling `Command.execute()` directly - Alex Hill
- Fix search by hacking around Django's abstract manager restriction - Alex Hill
- Allow Django 1.10 in `setup.py` - Alex Hill
- Pass raw context dict to `template.render()` in error views - Alex Hill
- Remove testing of dotted path for `LOGIN_URL` setting, since Django 1.10 doesn't support it - Stephen McDonald
- Prevent Django 1.10 from adding required attribute to admin change list actions dropdown - Stephen McDonald
- Replace usage of optparse with argparse - Alex Hill
- Ensure blog import commands contain base args - Stephen McDonald
- Add Django `stable/1.10.x` to test matrix - Alex Hill
- Upgrade pip and setuptools before test run - Alex Hill
- Restore access to parent template context in comments template tag. Closes #1654 - Stephen McDonald

Version 4.1.0 (Jan 17, 2016)

- Update Python version classifiers in `setup.py` - Stephen McDonald
- Update excluded files in package build - Stephen McDonald
- Force `local_settings.py` into `sys.modules` so it's visible to Django's autoreloader - Stephen McDonald
- Add the ability to use proxy in "Add ," drop down. We sometimes want to use proxy models in the add dropdown, to have. different changeform being backed-up by the same model. See ce02f8afe3d42dda for more information about the "and not. `m._meta.proxy"` part - Antoine Catton
- Improve readability by using list comprehension - Antoine Catton
- Remove extraneous site permissions field. Closes #1366 - Stephen McDonald
- Add to settings this: `RATINGS_ACCOUNT_REQUIRED` = True `COMMENTS_ACCOUNT_REQUIRED` = True `RATINGS_RANGE` = [-1, 1] and then run tests you will get about 5 errors, so i fixed tests for these settings and now they work well - d-first

- Do not throw away next parameter on login. `get_full_path()` throws away the "next" URL parameter, breaking. vanilla Django redirection behavior. The login form should redirect. to the "next" parameter on a successful login - Alexander Bliskovsky
- When the thumbnail richtext filter runs, deal with BeautifulSoup adding closing br tags, by stripping them out. Closes #1377 - Stephen McDonald
- `r_range` deleted replased with settings - d-first
- Fix textarea fields in live-editing - Stephen McDonald
- Upgrade to jQuery 1.8.3 - Nik Nyby
- Upgrade jQuery UI to 1.8.24. This upgrades jQuery UI from 1.8.2 to 1.8.24, fixing. a bunch of bugs. It also changes the CSS file used for jQuery UI - the `v1.9`. CSS was being used, so I've replaced it with the appropriate. CSS for version 1.8.24 - Nik Nyby
- Upgrade `jquery.form.js` to 3.51.0-2014.06.20 - Nik Nyby
- Fix edit overlay vertical positioning - Nik Nyby
- Upgrade Bootstrap from 3.0.3 to 3.1.1 - Nik Nyby
- Test on Python 2.7, Django 1.8. These are the versions I use by default, so we shouldn't need to exclude this combination - Nik Nyby
- Don't use zip files - Nik Nyby
- Fix TinyMCE width in live-editing - Stephen McDonald
- Bootstrap 3.2.0. Continuing the effort of gradually updating bootstrap,. this upgrades from 3.1.1 to 3.2.0. There's not many changes here and everything's looking okay. to me - Nik Nyby
- Fix tag list padding - Stephen McDonald
- Make a note about removing JS hack for site permissions field at some stage - Stephen McDonald
- Update `tinymce_setup.js`: Django language codes. `https://docs.djangoproject.com/en/1.8/topics/i1` Language codes are generally represented in lower-case, but the HTTP Accept-Language header is case-insensitive. The separator is a dash. I also noticed a warning message showing when I run 'python `manage.py` runserver'. it's about 'zh-tw' will be deprecated in Django 1.9, use 'zh-hant' instead. So I also add 'zh-hant'. I guess 'zh-cn' may be deprecated too. I tested 'zh-hant' and 'zh-tw' well, my tinymce editor shows hints in Traditional Chinese now. I only change the content in "var `language_codes` = { }" block, but github marks whole file to be changed. I don't know why - fygul
- Update `frequently-asked-questions.rst`. Update the link of "`urls.py` module" - Fygul Hether
- Checking if page attribute in request is of type `mezzanine.page.models.Page`, before processing it - pirave
- Ken is a core team member - Stephen McDonald
- Add support for blogs with > 500 posts - Anna Wiggins
- Replace timestamp trimming code with a more robust regex; current code broke on some timestamps returned by blogger - Anna Wiggins
- Add several new overridable blocks to the base template - Anna Wiggins
- Fix and test `page_processor's exact_page` argument - David Sanders
- Updating disqus sso tag with working encoder - pirave
- Resolve css/js Media paths using static templatetag - David Sanders
- Clean up page context processor check - Stephen McDonald

- Removing `tox.ini` since it's out of date - and of course it is, we don't use it - Stephen McDonald
- `utils.html`: HTMLParseError disappeared in Python 3.5. The HTMLParser is guaranteed not to choke on HTML soup - Hervé Cauwelier
- Add kwarg for form used in login and password reset views - David Sanders
- Don't titlecase group name for `ADMIN_MENU_ORDER` - David Sanders
- Add `_parent_page_ids` to global context scope. Currently if the first menu loaded is in a nested context. ' parent_pages_ids' can drop out of scope, and then never get reset since. *menu_pages* is set in *context.dicts[0]*. See `https://github.com/stephenmcd/mezzanine/issues/1154` - Julian Andrews
- Upgrade html5shiv to `v3.7.3` - Nik Nyby
- Remove unnecessary triple-quote in comment - Nik Nyby
- Don't overwrite files options in mezzanine-project command. Updating the *files* option here disables the *–name* option, which I need. to render custom template files (`i.e.` a Makefile) when making a custom. mezzanine template for use with *mezzanine-project*. This change adds *local_settings.py.template* to the list of files to render instead. of overwriting the list completely. This allows users to specify their own template. files if necessary. Relevant django code is here: `https://github.com/django/django/blob/master/django/core/management/templates.py#L56`. And my mezzanine template I'm getting this to work with is here: `https://github.com/nikolas/ctlmezzanine` - Nik Nyby
- Ensure global context variables set in `page_menu` template tag are actually global - Stephen McDonald
- Switch to a lazy static for Media statics - David Sanders
- In the admin dropdown menu, show add links to users without change permissions. This is a patch to fix the following problem: 1. Create a user with add permissions but not change permissions on a model. that is managed through the admin (`e.g.` blog posts). 2. Log in as that user in the admin site. 3. Click on the name of the model in the dropdown menu on the left. 4. Error. The problem arises because the template uses the `admin_url` instead of the. `add_url` for users without appropriate permissions to change instances of the. model. It then tries to remedy the situation by manually appending 'add/' to. the url. However, the url it receives from the template tag `admin_url` is set. to 'None' - Pieter
- Fix for `is_spam_akismet` to handle Python 3's bytestring response properly - Christian Hill
- Handle None as `content_type` subclass without breaking - Sam Kingston
- Restore login redirects on ratings/comments. Closes #1440 - Stephen McDonald
- Check for `FORMS_USE_HTML5` on the admin instead of the model. Fixes #1399 - Eduardo Rivas
- Removing duplicate profile fields from admin. Issue #1449 - Danny Sag
- Document `errors_for` and update `fields_for` docs. - The fields for template path and type are out of date. - Document `errors_for` since it adds a lot of value to `fields_for` - ryneeverett
- Add references to api-docs. This will add links to documentation generated from the source - Tomas Chmelevskij
- Added mezzanine-shortcodes to third-party apps - Stephen McDonald
- Don't strip HTML in TinyMCE. Because filtering is handled by bleach. Reimplement 0f6ab7c - ryneeverett
- Recommend using `includes/footer_scripts`. This is how it's done in the default templates but docs-readers probably. aren't going to know that they're already calling *editable_loader* via. an include - ryneeverett
- Add support for Python 3.5 in trove classifiers and travis build - Sam Kingston

- Add a setter for property `MenusField.choices`. Django 1.9's `CharField.choices` is just a regular attribute, no longer a. property. Its constructor tries to set a value, and since we weren't. providing a setter, we'd hit an exception - Alex Hill
- Update project template to use TEMPLATES - Alex Hill
- Suggest using TEMPLATES in `set_dynamic_settings` - Alex Hill
- Replace SortedDict with OrderedDict - Alex Hill
- Drop support for Django 1.7 - Alex Hill
- Drop Python 3.3 support - Alex Hill
- Don't use OrderableBase in 1.9. Django 1.9 supports ordering with respect to generic foreign keys,. which makes OrderableBase no longer necessary - Alex Hill
- Update Django & Python versions in `setup.py` - Alex Hill
- Import skipUnless from Python's unites - Alex Hill
- Fix saving related objects in tests - Alex Hill
- Don't use removed `Field.get_flatchoices()` - Alex Hill
- Remove check for page context processor in PageMiddleware - Alex Hill
- Remove LazyModelOperations. Django now provides lazy model signals which accept model strings and. render this class redundant - Alex Hill
- Update flake8 configuration to match Django's - Alex Hill
- Refactor `EXTRA_MODEL_FIELD` code. Break into a few smaller functions, and use `Apps.lazy_model_operation`. instead of `class_prepared` signal - Alex Hill
- Remove usage of SubfieldBase - Alex Hill
- Revert "Don't use OrderableBase in 1.9". This reverts commit 54d900776a2c7412cdacd7b5a6a4af44affac869 - Alex Hill
- Remove complexity check from flake8 config. This wasn't being honoured by the lint test before, so causes several. failures when enabled. We can add it again later and refactor those. functions if necessary - Alex Hill
- Bring `docs/conf.py` into line with flake8 - Alex Hill
- Restore but deprecate Mezzanine's `get_user_model()`. This wasn't actually defined before, simply imported from Django. We. define it here in order to provide a deprecation warning for anybody. importing it from this file - Alex Hill
- Make `EXTRA_MODEL_FIELDS` work in 1.8 again - Alex Hill
- Add a `footer_js` block in templates - Alex Hill
- Fix usage of TemplateResponse - Alex Hill
- Account for middleware subclasses in `cache_installed()` - Alex Hill
- Remove "builtins" from TEMPLATE options in Django < 1.9 - Alex Hill
- Can't use add() with bulk=False in Django 1.8 - Alex Hill
- Formalise template tests with checks framework - Alex Hill
- Fixed #1483 ValueError while parsing dates of imported Blogger posts - Daniel Blasco
- missing french messages for accounts, compiling `.mo` file - flo@mymedecine.fr
- Move SingletonAdmin to utils. Resolve #1480 - ryneeverett

- Add classes to `app_list` in admin. This provides a hook for model specific styling, and mirrors what Django. itself does. See: `https://github.com/django/django/blob/5399ccc0f4257676981ef7937ea84be36f7058a6/django/` - Julian Andrews
- Catch error if user sitepermissions don't exist. For the most part, if not using *SitePermissionMiddleware* Mezzanine. falls back on the *is_staff* attribute seamlessly. But since. d5d21ba527bd4 it's possible to have users without sitepermissions. This. breaks the admin dropdown, but not much else. This fix should allow. single site projects to continue to leave *SitePermissionMiddleware*. uninstalled - Julian Andrews
- Fall back to title field first before using string rep for meta title - Stephen McDonald
- Give the media library a root URL so it can be correctly highlighted in the admin nav. Closes #1505 - Stephen McDonald
- Only add debug toolbar urlpatterns if installed - Stephen McDonald
- Support admin classes registered via decorator. Closes #1462 - Stephen McDonald
- Fix some more cases of related managers requiring saved data in Django 1.9 - Stephen McDonald
- Remove redundant slashes in blog urlpatterns - Stephen McDonald
- Update from deprecated features of urlpatterns - Stephen McDonald
- Remove use of deprecated `get_all_field_names()` - Stephen McDonald
- Remove deprecated use of `i18n_patterns()` - Stephen McDonald
- Remove deprecated middleware names - Stephen McDonald
- Fix admin login interface selector - Stephen McDonald
- Fix slashes in blog urlpatterns - Stephen McDonald
- Create links to settings mentioned in the comments. Adds labels to the `settings.rst` files which can be used accross the. documentation to to link to their description - Tomas Chmelevskij
- Remove deprecated category/keyword helpers on blog posts, which existed for Django 1.3 - Stephen McDonald
- Remove Django 1.7 support from overextends templatetag - Stephen McDonald
- Remove a bunch of Django 1.7 handling from tests - Stephen McDonald
- TEMPLATES is a list of configurations - ryneeverett
- `django.core.context_processors->django.template,`. RemovedInDjango110Warning: `django.core.context_processors` is deprecated in favor of `django.template.context_processors` - ryneeverett
- string url views -> callable url views. RemovedInDjango110Warning: Support for string view arguments to url() is deprecated and will be removed in Django 1.10, Pass the callable instead - ryneeverett
- Fix user profile signal handler - Stephen McDonald
- Restore profile form fields - Stephen McDonald

Version 4.0.1 (Jul 26, 2015)

- Prompt for user creation in createdb command if interactive - Stephen McDonald
- Fix #1351 - exception in `displayable_links_js` - Alex Hill
- Use Django's createsuperuser command when createdb is run in interactive mode - Stephen McDonald
- Fix some Django 1.9 warnings - Stephen McDonald

- Remove references to long deprecated `PAGES_MENU_SHOW_ALL` setting - Stephen McDonald
- Locale middleware should fall after session middleware - Stephen McDonald
- Set up debug toolbar explicitly when installed. Closes #1358 - Stephen McDonald
- Restore support for alternate package option in mezzanine-project command - Stephen McDonald
- Fix for issue #1361 - backup command fails when called from deploy - Luke Plant
- Use 'exec' instead of import to add `local_settings`. This allows `local_settings` to reference and modify existing settings. Refs issue #1360 - Luke Plant

Version 4.0.0 (Jul 09, 2015)

- Strip punctuation from keywords instead of non-alphanumeric chars, since languages like Hindi use characters that Python doesn't consider to be alphanumeric - Stephen McDonald
- Unpin tzlocal version 1.0 - Alex Hill
- Update tinymce setup to use browsers' built in spell checking. Tested and works in latest FireFox and Chrome and Safari - joshcartme
- Support custom user models in admin password change view. Previously it was assumed that the user's pw change view is at "auth/user/(d+)/password/", which caused NoReverseMatch with custom models - Rivo Laks
- Fix admin password change for Django 1.5 - Stephen McDonald
- Support Django 1.7 migrations - Baylee Feore
- Don't remove south when `USE_SOUTH` isn't defined - Stephen McDonald
- Use `is_staff` to check for logged in user in base admin template. Closes #1114 - Stephen McDonald
- beautifulsoup version should not be pinned exactly. There's no need to require exactly 4.1.3 - Gavin Wahl
- Added brackets for Python 3 print() - Tuk Bredsdorff
- Handle malformed user agent strings. Closes #1116 - Stephen McDonald
- Configure `SOUTH_MIGRATION_MODULES` setting to check for custom south migration packages - Stephen McDonald
- Move `south_migrations` -> migrations/south - Stephen McDonald
- Added *The Entrepreneurial School <http://theentrepreneurialschool.com/>* to site using Mezzanine - Renyi Khor
- Fix DoesNotExist when non admin visits /admin. SitePermission objects are only added when staff users are created. If a non admin user (with no manually assigned site permissions) visits the admin a DoesNotExist is raised since no site permissions exist for the user. Therefore the templatetags logic should only run if the user is staff - joshcartme
- Exposed `JQUERY_UI_FILENAME` for templates - wrwrwr
- wordpress export is under Tools, not Settings - Gavin Wahl
- Renamed all `get_query_set` methods to `get_queryset`. Django 1.6 normalized the naming, providing a metaclass that handles. previous naming schemes, while printing a warning. See: `https://code.djangoproject.com/ticket/15363`. This is probably incompatible with 1.5, but limits the amount of. Django 1.8 deprecation warnings - wrwrwr
- Explicitly list fields when defining form from model. This only changes one test (`` test_richtext_widget``), but here's a. short article that may explain why `fields` or `exclude` becomes. mandatory argument to `modelform_factory` and why you actually may. want to explicitly list fields to be available in forms:

http://blog.mhartl.com/2008/09/21/mass-assignment-in-rails-applications/ - wrwrwr

- Unfrozen future, pep and flakes. If the reason to keep these at some fixed old versions still exists,. please add a comment - wrwrwr
- Let sqlite configuration not contain a NAME at all (as with some settings environments) - wrwrwr
- Resolve race condition in conf.settings - Alex Hill
- Fixed a couple of block comments not starting with hash and space - wrwrwr
- List form fields (as recommended) to preserve compatibility with 1.5. The "__all__" shortcut must be past-1.6 only - wrwrwr
- Fixed test_login_required when run without mezzanine.accounts. Note 1: Always testing both scenerios no matter what's in the settings. would be thorough, but without something like the 1.7+ modify_settings. it turns out ugly. It would be better to run the whole suite with and. without some optional apps. Note 2: This test passes when run through runtests, but it cheats. by forcing mezzanine.accounts into installed apps ;-) - wrwrwr
- Allow Mezzanine's static files handling to support MEDIA_ROOT outside of STATIC_ROOT during development - Stephen McDonald
- Don't colorize the terminal banner if color sequences aren't supported - Stephen McDonald
- Cleaned up in_menus defaults test. Making use of the runtime settings changes support - wrwrwr
- Added failing settings race condition test - Alex Hill
- Simpler fix for race condition in settings - Alex Hill
- Reorder and add comments to settings test - Alex Hill
- Overwrite settings cache when loading from DB - Alex Hill
- Clear DB settings after test run - Alex Hill
- Made forms tests use Mezzanine's TestCase. Not currently necessary, but potentially surprising if you add something. to the TestCase - wrwrwr
- Added a simple decorator defining get_query_set or get_queryset. allowing to use the latter one while preserving compatibility with 1.5. Django (1.6+) uses a metaclass to allow usage of the former name after. renaming a method (django.utils.deprecation.RenameMethodsBase), but. for the 2 cases in Mezzanine a decorator seems sufficient and less. intrusive - wrwrwr
- Replaced usage of Options.get_(add|change|delete)_permission with. auth.get_permission_codename. Just a single case in utils/views.py. The former one is deprecated and will be removed in Django 1.8; see. https://code.djangoproject.com/ticket/20642 - wrwrwr
- Added an 1.5-compatible implementation of get_permission_codename - wrwrwr
- Made MenusField respect the current value of PAGE_MENU_TEMPLATES_DEFAULT - wrwrwr
- Small semantic change in MenusField behavior, now in_menus returns an. empty tuple instead of None for a page not in any menu. This seems more consistent with PAGE_MENU_TEMPLATES_DEFAULT = tuple() - wrwrwr
- Made MenusField also support dynamic changes to PAGE_MENU_TEMPLATES. This costs a call to a private Field._get_choices(), because. Field.choices is already a property - wrwrwr
- Extended the test_login_required to check if LOGIN_URL set to view or. pattern name still allows the decorator to work as expected. The new cases are only checked if mezzanine.accounts is installed. These

additional LOGIN_URL possibilities were introduced in 1.5; view. objects should also work, but don't seem to be documented - wrwrwr

- Don't test the new options with 1.4, it's not supposed to support them - wrwrwr
- Prevent duplicate app names when testing - Stephen McDonald
- Extend renamed_get_queryset to also support admin classes, and apply to OwnableAdmin - Stephen McDonald
- set_slug now save itself - Dustin Broderick
- set_slug now saves itself - Dustin Broderick
- Remove the requirement for searchable models to subclass Displayable - Stephen McDonald
- Handle installing initial data with Django 1.7's syncdb signals. Closes #1123 - Stephen McDonald
- Remove redundant page save in slug tests - Stephen McDonald
- Slightly faster settings loader - Stephen McDonald
- Fix some docstrings and messages - Stephen McDonald
- Actually skip the threading test as expected - Stephen McDonald
- Fix version check in post_syncdb signals - Stephen McDonald
- Added description of "label" keyword argument of register_settings. function to docs - eyeinthebrick
- Cache site ID on request object, even when fallback setting is used. Closes #1144 - Stephen McDonald
- Fix site ID fallback when testing - Stephen McDonald
- separate basic gallery functionality. to be more flexible with creating galleries we can separate gallery. functionality so it can be reused if developer wants to create a. gallery but not necessarily a gallery page - Robert Zywucki
- Added a test for DynamicInlineAdmin - wrwrwr
- Allow DynamicInlineAdmin fields to be a tuple. Previously, if fields was defined as a tuple, a very confusing TypeError. would be raised - Rocky Meza
- Travis test the latest versions of Django. This way you don't have to update every time a security release comes. out - Rocky Meza
- Re-instate conf test as per recent threading fixes. Closes #858 - Stephen McDonald
- Return HTTP 405 on comment/rating URLs for GET requests. Closes #1159 - Stephen McDonald
- Fix Travis Django installation - Rocky Meza
- Shamelessly modified the dynamic admin fields tuple test, so it checks. get_fieldsets() instead of directly using the fields attribute. Also made BaseDynamicInlineAdmin work when fields are listed without. the _order field or fieldsets are declared (with or without it) - wrwrwr
- SiteRelated changed to allow explicit site assignment on creation - Kelvin Wong
- Made "flake8 ." output less warnings. Excluded docs/conf.py as its autogenerated by Sphinx and increased. acceptable complexity from 10 to 20 - wrwrwr
- Allowed optional apps to be loaded for testing - wrwrwr
- Separated generation of short_urls from their saving - wrwrwr
- include menu pages in broader context. issue #1154 - Eduardo S. Klein
- Refactored confirmation prompts in createdb, create_pages and collecttemplates - wrwrwr

- Factored out deleting unused keywords as a manager method - wrwrwr
- Removed `django_extensions` from `INSTALLED_APPS` for testing - wrwrwr
- Fixed `pages.test_login_required` with I18N / LocaleMiddleware - wrwrwr
- Fixed `core.test_password_reset` with `i18n_patterns` - wrwrwr
- Use the response context during two-phase rendering. If a response has *context_data* use that instead of creating a new. context and running context processors again - David Sanders
- Fix for using response `context_data` - David Sanders
- Remove call to `set_model_permissions` in page admin - Alex Hill
- Backout context processor changes to cache middleware for now, re #1174 - Stephen McDonald
- Prevent order setting vs. form submission race condition. If the js runs slow for any reason the admin form can submit before the order of inlines has been set - joshcartme
- Create new model field OrderField - Alex Hill
- `OrderWidget.is_hidden` should evaluate False - Alex Hill
- Update orderable ordering to use the submit event. It seems that the click event may not be guaranteed to run in FireFox before the form actually submits. This may be due to a variety of factors including the version of jQuery but for now switching to the submit event solves the problem - joshcartme
- Fixed spelling of 'collapsed' - Stone C. Lasley
- Refactored all the initial data setup to only occur when the createdb command is used, since Django 1.7's migrations no longer provide a signal that can reliably trap when tables are created - Stephen McDonald
- Added migrations for changed `_order` field - Stephen McDonald
- Host the dashboard screentshot in the docs so it can be correctly referenced on github - Stephen McDonald
- Fix docs build for Django 1.7 - Stephen McDonald
- Fix for brocken link - Krzysztof Szumny
- Fix optional data installation for non-interactive installs - Stephen McDonald
- Don't store absolute urls as short urls in the db, since the column length is shorted, and these aren't necessairly permanent either. Closes #1178 - Stephen McDonald
- Refactored the `fields_for` templatetag to allow for custom field template - Avery Laird
- nginx conf: use Mozilla recommended ciphers. For the included `nginx.conf`, use the ciphersuite recommended by the Operations. Security team at the Mozilla project: `https://wiki.mozilla.org/Security/Server_Side_TLS`. The ones included here are the ones labelled "Intermediate compatibility". They. also document a ciphersuite with a higher level of security on that page. labelled "Modern compatibility", though it excludes more older browsers - Lorin Hochstein
- Updated the js in `footer_scripts.html` to handle universal analytics - Avery Laird
- Convert relative URLs to absolute in RSS feed item descriptions. Closes #1193 - Stephen McDonald
- Fix 500 error when the 'comment' view function doesn't receive expected form data - Christian Abbott
- Filenames in ZIP not always UTF-8. Fixing this with chardet - Souren Araya
- Don't assume `INSTALLED_APPS` is all modules as Django 1.7 adds AppConfig - Frankie Robertson
- Replace `django.contrib.comments` with `django_comments`. Django has removed the comments app from `django.contrib`, and. moved it into an external django-contrib-comments package hosted. under the Django GitHub organisation - Alex Hill

- Test up to Python 3.4 and Django up to 1.8 - Alex Hill
- Test with Py 3.4, Django 1.7 and default runners. Update `.travis.yml` config to run tests under Python 3.4 and Django 1.7. Future-proof the test script to work with both default test runners: DjangoTestSuiteRunner `pre-1.6`, and DiscoverRunner in 1.6 and later. Under DiscoverRunner, the `django.contrib` tests won't be run - Alex Hill
- Enable SHA1 hasher for happy Django 1.4 tests - Alex Hill
- Miscellaneous backwards-compatible 1.8 fixes. Includes changes to Meta, some moved functions and various internal API. changes - Alex Hill
- Accommodate Django 1.8's template system changes. Django 1.8 introduces the possibility of using multiple template. engines, which means that some components of Django's template system. have been encapsulated in an Engine class instead of being available. globally, and other parts refactored and moved around - Alex Hill
- Remove Django 1.4 and 1.5 from `.travis.yml` - Alex Hill
- No need to enable SHA1 hasher in tests > 1.4 - Alex Hill
- Remove `mezzanine.utils.models.get_user_model`. This was introduced for compatibility with Django 1.4 after Django 1.5. included support for custom User models. Now that we no longer support. Django 1.4, we can just use the builtin version. Still need to remove references in filebrowser, so the import remains. in `mezzanine.utils.models` - Alex Hill
- Remove shims for `force_text` and `smart_text`. Not necessary since dropping support for Django 1.4 - Alex Hill
- Deleted `mezzanine/utils/deprecation.py`. This contained compatibility fixes for Django < 1.6, which we no longer. support - Alex Hill
- Remove miscellaneous BC fixes for Django < 1.6 - Alex Hill
- Use `get_models` from `django.db.models.loading` - Alex Hill
- Bump required Django version to > 1.6.0 - Alex Hill
- Import `local_settings.py` before test run - Alex Hill
- Restore previous behaviour in runtests script - use the `local_settings` template to create `test_settings`, not an actual `local_settings` module, which may already exist during development - Stephen McDonald
- Use south's test command, which is needed to pick up our custom values for `SOUTH_MIGRATION_MODULES` - Stephen McDonald
- Some envs don't create pyc files - Stephen McDonald
- Allow `local_settings` import when project isn't a package - Stephen McDonald
- Some path hacks during development to allow tests to be picked up when calling the test command via `manage.py` - Stephen McDonald
- KeyError when excluding `first_name`, `last_name`, and username fields using `ACCOUNTS_PROFILE_FORM_EXCLUDE_FIELDS` and `ACCOUNTS_NO_USERNAME` settings - rsadwick
- Fix handling blank first/last names in username creation - Stephen McDonald
- Fix bug with missing scope in atexit registered function in test runner - Stephen McDonald
- Update `views.py`. clean the cookie rating record after an auth user undoing his/her rating - Yuheng QIU
- Exclude commit messages with 3 words or less from the changelog - Stephen McDonald
- Using Tabbed admin instead of showing all fields at once - Mathias

- Delete cached context settings on settings update. With editable settings it is best to invalidate the context settings. cache key when settings have been changed via the admin panel so that. they can take effect immediately - David Sanders
- Added site ID back to settings cache key prefix - David Sanders
- pip can't seem to install Django 1.8 from a zip file on Python 2.7 - Stephen McDonald
- Fallback lookup for current page should exclude link pages, since they may contain duplicate slugs. Closes #1220 - Stephen McDonald
- Update docs and add mini-tutorial in Deployment section - Eduardo Rivas
- Handle new app-loading mechanism for Django 1.7 in fabfile. See `https://docs.djangoproject.com/en/1.7/releases/1.7/#standalone-scripts` - Eduardo Rivas
- Add documentation about using AppConfigs in `INSTALLED_APPS` and `defaults.py` - Frankie Robertson
- Added exception handler for `make_dirs()` function call in thumbnail() template tag - Alex Bendig
- Fix error raised when twitter lib is installed, but `mezzanine.twitter` is removed from `INSTALLED_APPS` - Stephen McDonald
- Update `multi-lingual.rst` for spelling/grammar - Ryan Sadwick
- Show Links to Code in Package Documentation. Refs #1148 - Pavan Rikhi
- Account for `cartridge.shop` being after `mezzanine.pages` in the template rendering pipeline. Also enable nested {% ifinstalled %} tags - Mathias
- Twitter: Fix parsing of email addresses as Twitter usernames - Eduardo Rivas
- Added setting to allow custom forms for submitting comments. Handy for common requests from people such as adding captchas. Easiest way is to create a new form class that inherits from ThreadedCommentForm then specify that class in the new setting - Paul Hunt
- Import WordPress draft posts correctly - Jan Varho
- Don't consider proxy models a content model. Otherwise we'll try to use them with `select_related`, which is not. supported - Gavin Wahl
- Remove all support for South in favour of Django >= 1.7 migrations - Stephen McDonald
- Remove all support for Django < 1.7 - Stephen McDonald
- Add option for thumbnails to not grow in size. Add the *no_grow* option to the thumbnail template tag, specifying. whether a thumbnail is allowed to grow in size when resizing to a given. width or height - Simen Heggestøyl
- Fix createdb to work with Django 1.7 and 1.8. Django 1.8 changed the way optional args are added to management commands. This patch adds optional args "the old way" for Django 1.7 in *Command.__init__*, and then uses "the new way" for Djagno 1.8, which is via a call to the new class method *add_arguments()* - Eduardo Rivas
- Check for valid Python package name format in mezzanine-project command. Closes #1248 - Stephen McDonald
- Add `SECRET_KEY` to docs config which fixes broken rtd builds - Sam Kingston
- Attempt to fix path issue in rtd environment - Sam Kingston
- Set `CurrentSiteManager.use_in_migrations` = False - Alex Hill
- Migrations for Django 1.8 - Alex Hill

- Fix overextends tag for Django 1.7 & 1.8 - Alex Hill
- Compile all new Spanish locale - Eduardo Rivas
- Initial project layout update - Alex Hill
- Allow project template's `local_settings.py` into repo - Alex Hill
- Explicitly set email field `max_length` to 254 - Alex Hill
- Make updated project template work with `local_settings.py.template` - Alex Hill
- Restore `".template"` suffix in test script - Alex Hill
- Django changed `Model._meta.get_parent_list()` to return a list instead of a set! This fixes it - Stephen McDonald
- Change field on site permission model as per Django warning - Stephen McDonald
- Add mezzanine-modal-announcements to third party modules - Josh Cartmell
- change redirect to be permanent, which is how Django does it - Darius
- Change way PostgreSQL version is displayed - orotau
- Fix import error with latest django-contrib-comments - Stephen McDonald
- Fix various import warnings for Django 1.9 - Stephen McDonald
- Use a consistent version of jquery-ui - Stephen McDonald
- Add the `featured_image` as an enclosure to the rss feed. Add the featured image to the rss feed as an enclosure url, so that it can be used by feed readers. For example Zapier RSS to Facebook to use as the Post Image - Michael Best
- Restricted support Django versions to < 1.9 - Stephen McDonald
- Prefetch blog categories in RSS feed - Stephen McDonald
- Ensure host for the current site is used in RSS feeds - Stephen McDonald
- Handle parent comment ID in duplicate comment check. Closes #1286 - Stephen McDonald
- Added mezzanine-slideshows to third-party apps - Stephen McDonald
- Updated email address for security issues - Stephen McDonald
- Fix edge case where `mezzanine.accounts` code is run even though not installed (generated docs, some alternate test run setups), and the `ACCOUNTS_PROFILE_FORM_EXCLUDE_FIELDS` isn't defined - Stephen McDonald
- Added mezzanine-onepage to third-party apps - Stephen McDonald
- Ensure consistent `related_model` attribute throughout django fields - Mathias
- Keep compatibility with django 1.7 for `generic.fields.BaseGenericRelation` - Mathias
- Ensure front-end language selector is always visible - Stephen McDonald
- Switch forms admin to use the object-tools-items block in template - David Sanders
- Add some space between the filter-horizontal admin widget and its help text - Stephen McDonald
- Move default button text for forms app from model into template - Stephen McDonald
- Fix double-escaping of HTML in settings admin form field help text - Stephen McDonald
- Define correct JS media for settings admin form class - Stephen McDonald
- Ensure `mezzanine.accounts` is available when generating settings docs - Stephen McDonald

- Allow protocol to be omitted from URL arg in `import_rss` command - Stephen McDonald
- Fix mysql/unicode issue on saving Link pages - Stephen McDonald
- Use Django language code to configure language for TinyMCE - Stephen McDonald
- Remove automatic selection of site permission for staff users, since it breaks when manually choosing them in the admin interface - Stephen McDonald
- Updating documentation to account for the switch between South and Django's builtin migration tools - Mathias
- Updating multi-lingual documentation to add a note on migrations - Mathias
- Update `model-customization.rst`. Small grammar changes - orotau
- Restore support for the `ADMIN_REMOVAL` setting. Closes #1313 - Stephen McDonald
- Don't assume `ADMIN_REMOVAL` setting is defined - Stephen McDonald
- Added optional `extra_context` to applicable views - David Sanders
- Add index on `publish_date` - Frankie Robertson
- Ensure emptyish page titles in admin tree are clickable. Closes #1321 - Stephen McDonald
- Smarter handling of editable settings - Alex Hill
- Make `use_editable` backward compatible - Alex Hill
- Use `clear_cache` in tests - Alex Hill
- Remove calls to `Settings.use_editable()` - Alex Hill
- Make setting context processor lazy - Alex Hill
- Make Settings object more opaque WRT the current thread - Alex Hill
- Remove context settings caching - Alex Hill
- Show repr of setting values in warning - Alex Hill
- Add a few settings tests - Alex Hill
- Don't require `Settings.clear_cache()` outside of a request - Alex Hill
- Test conflicting settings warning - Alex Hill
- Refactor settings retrieval logic - Alex Hill
- Clear request settings cache when settings saved - Alex Hill
- Handle updated project layout in fabfile - Alex Hill
- Handle updated project layout in `supervisor.conf` - Alex Hill
- Update `settings.py` for new project layout - Alex Hill
- Fix URLs during tests - Alex Hill
- Stop server before dropping database - Alex Hill
- Fix paths in settings - Alex Hill
- More refinements to fabfile - Alex Hill
- Give all deploy files the `.template` extension - Alex Hill
- Add utility function `real_project_name` - Alex Hill
- Use `real_project_name` in `fabfile.py` - Alex Hill

- Use `real_project_name` in `manage.py` - Alex Hill
- Add missing blank line - Alex Hill
- Use `real_project_name` in `wsgi.py` - Alex Hill
- Use unicode paths in project template handling - Alex Hill
- Add missing trailing bracket - Alex Hill
- Add `BASE_DIR` to settings - Alex Hill
- Don't use DB settings at import time - Alex Hill
- Update settings docs to reflect removeal of `use_editable` - Stephen McDonald
- Fix doc generation for new project template layout - Stephen McDonald
- Update jquery to 1.7.2, jquery-mobile to 1.2.1. In an effort to gradually bring mezzanine's javascript up to date,. this commit updates mezzanine's default jquery version to the. latest in the 1.7.x series (which is 1.7.2). Because the 1.7.2 release notes state that this version. should be used with jQuery Mobile >= version 1.1, I've updated. jQuery Mobile as well. `http://blog.jquery.com/2012/03/21/jquery-1-7-2-released/` - Nik Nyby
- Prefer published date over updated date in RSS importer. Closes #1329 - Stephen McDonald
- Fix manually assigned template settings, such as `MEZZANINE_ADMIN_PREFIX` - Stephen McDonald
- Fix host themes when current site does not match domain, eg when selected via admin. Closes #1327 - Stephen McDonald
- Ensure local middleware installed if required - Stephen McDonald
- Change next release numbering in warning message - Mathias Ettinger
- Upgrade to TinyMCE 4. Closes #705 - Stephen McDonald
- Refactor dynamic inline handling. Fixes a bug where Mezzanine would delete inline rows even when its. inlines weren't being used - Alex Hill
- Correctly handle `GRAPPELLI_INSTALLED` and `ADMIN_MENU_COLLAPSED` - Alex Hill
- Amend file location for local settings when advising on contribution - Stuart Dines
- admin: use apps config `verbose_name` for display - gradel
- TinyMCE 4.2 -> 4.1 due to image insertion conflict - Stephen McDonald
- Don't show draft blog posts under related posts - Stephen McDonald
- Ensure link pages only allow external URLs. Closes #1342. Closes #1345 - Stephen McDonald
- Revert changes to Link model validation - Stephen McDonald
- Clean leading/trailing slashes from page admin slugs - Alexander Bliskovsky

Version 3.1.10 (Aug 30, 2014)

- Delete kwargs for `page_view`. This is strange, pass *view_kwargs* form some view to `page_view`. I have error if kwargs contin *slug* key or some keys with not exists in `page_view` kwargs - Pahaz Blinov
- South should not be used for Django 1.7 - Stephen McDonald
- Added mezzanine-bsbanners to third party apps - Stephen McDonald
- Clean up some docstrings - Stephen McDonald

- For dropdown fields in the forms builder app, use the placeholder text to provide an empty first choice - Stephen McDonald
- Add a custom banner to Mezzanine's runserver command, showing a logo and various software versions - Stephen McDonald
- Handle new management command loading order in Django 1.7, so we can override runserver - Stephen McDonald
- Make the setuptools test runner part of the mezzanine package so that we can call it from other packages (such as cartridge). Also make its `local_settings` module more dynamic - Stephen McDonald
- Remove working directory hack from bundled `manage.py` - conflicts with Django runserver's auto-reloader when run from outside of project root - Stephen McDonald
- Updated development instructions for setting up Mezzanine when contributing - Stephen McDonald
- Update to latest patch versions of Django in travis config - Stephen McDonald
- Better explansion example in dev setup docs - Stephen McDonald
- A test for the `login_required` property has been added - Ulrich Wagner
- A more complete test for the `login_required` property - Ulrich Wagner
- Don't apply a zindex to h1 tags in admin, since it conflicts with date picker widgets. Closes #1087 - Stephen McDonald
- Raise 404 in page view if page is not an exact match. Closes #1090 - Stephen McDonald
- Add unique constraint to site perm model. Closes #1089 - Stephen McDonald
- Provide a more meaningful exception message when the `SEARCH_MODEL_CHOICES` settings points to missing models - Stephen McDonald
- Run page processors before view in PageMiddleware - Alex Hill
- Only run page processors for Mezzanine's page view - Alex Hill
- Update `models.py`. This fixes day light saving issue. It seems that `created_at` is coming with timezone 0000 UTC, so I would like to make ti timezoneaware using just utc. Django templates shows datetime ok with timesince without one hour error - eino-makitalo
- Fix login required test for protected page - Stephen McDonald
- Fix dynamic `INSTALLED_APPS` in setuptools test runner - Stephen McDonald
- Remove non-ascii chars from comment - Stephen McDonald
- Better approach for test settings - Stephen McDonald
- Issue 1102 - fixed UnicodeEncodeError when importing non-ascii files from zip - Maciej Szulik
- Fix `local_settings` import error check for Python 3 - Stephen McDonald

Version 3.1.9 (Jul 12, 2014)

- Nicer error message when `register_setting` is called with editable and no default - Stephen McDonald
- In thumbnails richtext filter, use built-in HTML parser with BeautifulSoup to better preserve original markup - Stephen McDonald
- Bail early in thumbnails richtext filter if `MEDIA_URL` isn't used - Stephen McDonald

Version 3.1.8 (Jul 10, 2014)

- Support custom User model in AdminLoginInterfaceSelectorMiddleware - Rocky Meza
- Fix call to `get_model` in search view - Stephen McDonald
- Add test for the search view - Stephen McDonald

Version 3.1.7 (Jul 06, 2014)

- Fix twitter query encoding - Stephen McDonald

Version 3.1.6 (Jul 05, 2014)

- Stricter child page lookup when parent slug is updated. Fixes #1045 - Stephen McDonald
- In thumbnails richtext filter, maintain markup that html5lib deems belonging in a head tag, and fix any script tags that have been self-closed - Stephen McDonald
- In front-end editing, handle widget overrides not existing when `FORMS_USE_HTML5` is False - Stephen McDonald
- Remove some redundant encoding/escaping from Twitter queries, and handle some MySQL unicode warnings - Stephen McDonald
- Fix regression in generic signal handling in Django 1.7 - Alex Hill
- Initial refactor of profile support for Django 1.7 - Alex Hill
- LazyModelOperations can wait on multiple models - Alex Hill
- Add trailing newline to `accounts/models.py` - Alex Hill
- Allow `'app_label.ModelName'` passed to `get_model()` - Alex Hill
- Fail early with invalid lazy model names - Alex Hill
- Use `get_profile_for_user()` in profile signal handler - Alex Hill
- Use ProfileNotConfigured in profile functions - Alex Hill
- Use ProfileNotConfigured in `accounts/admin.py` - Alex Hill
- Update accounts tests to use ProfileNotConfigured - Alex Hill
- Use `get_profile_fields_form()` rather than importing - Alex Hill
- Fix return statement in `accounts_tags.profile_fields()` - Alex Hill
- Fix dependency: oauthlib is used - Melvyn Sopacua
- Replace html5lib with BeautifulSoup in thumbnails richtext filter, to preserve original HTML. Closes #1056 - Stephen McDonald
- In thumbnails richtext filter, only resize images uploaded to `MEDIA_URL`. Closes #1058 - Stephen McDonald
- Use beautifulsoup4 for Python 3 support - Stephen McDonald
- Added mezzanineopenshift to third-party apps - Stephen McDonald
- Remove distutils version checking code which doesn't work with strings in versions - Stephen McDonald
- Allow alternate settings module when building docs. Closes #1062 - Stephen McDonald
- Remove redundant encoding. Closes #1063 - Stephen McDonald

- Redundant forms admin attribute - Stephen McDonald

Version 3.1.5 (Jun 09, 2014)

- Small css fix for settings admin page - Mario Rosa
- Check if `django.contrib.admin` is in `INSTALLED_APPS`. Make sure `django.contrib.admin` is in `INSTALLED_APPS` before moving. it to the end of the list. Adding this enables moving admin app. on a different domain - Dovydas Stepanavicius
- Add language selector in the navigation bar - Mathias Ettinger
- Generate slug from title for default language instead of active one - Mathias Ettinger
- rtl: fix margins in admin - Ahmad Khayyat
- Fix ID check in `admin_page_ordering` view to prevent some redundant queries - Stephen McDonald
- Revert bad `get_FOO_display` change. Closes #1032 - Stephen McDonald
- Don't strip quotes from twitter template tag args which was only needed in early Django versions. Closes #1034 - Stephen McDonald
- Remove global signal handlers in `mezzanine.generic`. Fixes #1036 - Alex Hill
- Implement a new `can_move` dynamic page permission. Content types can override `can_move()` to control whether a given page. move in the page tree is permitted. `can_move()` should raise a. PageMoveException if the move is denied. PageMoveException takes a. single argument: a message explaining the reason for the denial - Ahmad Khayyat
- page tree: reload on move exception to revert the tree and display any messages - Ahmad Khayyat
- Move LazyModelOperations into `mezzanine.utils.models` - Stephen McDonald
- Added `mezzanine.utils.html.thumbnails` to `RICHTEXT_FILTERS` setting, which converts images within richtext fields into thumbnail references. Closes #567 - Stephen McDonald
- CSS fixes for send to twitter admin checkobx - Stephen McDonald
- Document the `can_move` dynamic permission - Ahmad Khayyat
- Faster admin left-hand menu animation - Stephen McDonald
- Show featured images in recent blog posts section - Stephen McDonald

Version 3.1.4 (May 05, 2014)

- Remove previous cached build dir when creating wheel builds - Stephen McDonald
- Explicitly close db connection in `poll_twitter` command - Stephen McDonald
- Remove deprecated Debug Toolbar configuration. `DEBUG_TOOLBAR_CONFIG.INTERCEPT_REDIRECTS` has been deprecated. In fact, the Redirect panel is now disabled by default, so we don't need to define the setting any more. Deprecation notice: `http://django-debug-toolbar.readthedocs.org/en/latest/changes.html#deprecated-features`. New default value: `http://django-debug-toolbar.readthedocs.org/en/latest/configuration.html#t` - Eduardo Rivas
- Bypass streaming responses in cache middleware. Closes #1020 - Stephen McDonald
- Django 1.7: avoid trying to load user model during startup for profile signals - Stephen McDonald
- Support pre/post Django 1.7 CurrentSiteManager - Stephen McDonald

- Change handling of multiple versions of password reset urlpattern format to work with Django 1.7 - Stephen McDonald
- Django 1.7: fix exception handling when testing for slugs or app/model names in page processors - Stephen McDonald
- Django 1.7: fix initial site creation, in 1.7 the default site already exists - Stephen McDonald
- Django 1.7: fix admin formfields for generic relations, thanks to @loic84 - Stephen McDonald
- added `get_XXX_display()` to MultiChoiceField - slav0nic
- Only abort second phase cache render for non-200 http status when DEBUG is enabled - Stephen McDonald
- Bump grappelli-safe version to 0.3.10 - Stephen McDonald

Version 3.1.3 (Apr 25, 2014)

- Fix regression in page middleware on non TemplateResponse responses - Stephen McDonald
- Revert change to javascript location in default templates - Stephen McDonald

Version 3.1.2 (Apr 25, 2014)

- [#1004] Move Javascript to the Bottom of the Body. * Move all Javascript Includes to Bottom of the `base.html` and. `mobile/base.html` templates. * Move `extra_js` Blocks Below main Blocks in the `pages/form.html` and. `pages/gallery.html` templates. closes #1004: JS should be at the bottom of body - Pavan Rikhi
- Widget js shouldn't need staticfiles handling - Stephen McDonald
- Allow local testing of `STATIC_URL` containing host - Stephen McDonald
- Add Additional 3rd Party Integration Documentation - Pavan Rikhi
- Indicate active tree branch in tree page menu - Stephen McDonald
- Move the logic for assigning a page variable to the template context, from PageMiddleware into a context processor. This ensures we don't reply on TemplateResponse objects - Stephen McDonald
- More reworking of the docs for integrating apps with pages - Stephen McDonald
- Give a `.template` extension to Python files under `project_template/deploy` since they aren't valid Python files and an attempt to compile them may be made when installing. Closes #1010 - Stephen McDonald
- Make the new page context processor backward compatible with older projects - Stephen McDonald
- Fix some deprecation warnings: mimetype -> `content_type` - Stephen McDonald
- Restore site selection dropdown menu in admin nav - Stephen McDonald
- Decouple the Blog Post List from RichTextPages. The `blog_post_list.html` template currently requires that any *page*. context variable passed to it has a *richtextpage* attribute. These changes modify the template, making the *richtextpage* attribute. optional. This allows users to create other Page types pointing to the. URL of the Blog Post List - Pavan Rikhi
- Fix styling of admin help text. Closes #1013 - Stephen McDonald
- Further decoupling of richtextpage in blog templates - Stephen McDonald
- Apply admin title click globally - Stephen McDonald
- Show users section in admin nav highlighted as active for the change password view - Stephen McDonald

- Include grappelli in `INSTALLED_APPS` during tests - Stephen McDonald
- Skip reset password test for older Django versions since we now include grappelli, which has a bad url name in the reset password template. Since fixed but not released - Stephen McDonald
- Ensure missing settings vars don't break admin JS. Closes #1015 - Stephen McDonald
- Bump grappelli safe version to 0.3.9 - Stephen McDonald

Version 3.1.1 (Apr 19, 2014)

- Fix richtext widget loading - Stephen McDonald

Version 3.1.0 (Apr 17, 2014)

- Allow auth admin's change password view to be reversed - Stephen McDonald
- Fix for ignoring `search_fields` when they are set on the ModelAdmin - Sam Kingston
- Make pillow an explicit dependency since wheel distributions can't have optional dependencies, and clean up uses of it - Stephen McDonald
- Update reference and link for Pillow in Deps - Matt Stevenson
- Add Debian and Ubuntu prereqs for Pillow - Matt Stevenson
- Something is broken with password-resetting. Building a test to reproduce the issue (already hitting the first problem). Will do further testing and fixes in this branch - Arnold Krille
- Further up the test-road. Running this with `Django1.5.5` works here. The test reaches the page congratulating to the new password and showing a link to "Log in". Unfortunately that link leads to an invalid url '/accounts/login' by default:-(But thats probably stuff for a different Issue/PR - Arnold Krille
- fix for the python3 problem? - Arnold Krille
- Make the urls version-dependant. This feels a bit dirty. But it could actually work - Arnold Krille
- Python3 fixes. Don't know if this is correct, its my first actual encounter with python3;-) - Arnold Krille
- Add a note about osx setup for libjpeg - Stephen McDonald
- When testing, do so in english. And follow any language-redirects should these occure - Arnold Krille
- Better uidb64 password reset version check - Stephen McDonald
- Fix the fix for reversing the change password form when multiple langages are enabled in the admin - Stephen McDonald
- Provide a warning on admin login when the default password is being used - Stephen McDonald
- Bump grappelli version to 0.3.6 - Stephen McDonald
- Fix `add_to_builtins` import for Django 1.7 - Stephen McDonald
- Initial version of new admin menu - Stephen McDonald
- Adds docs for `HOST_THEMES` configuration - geojeff
- Made a few grammatical fixes - geojeff
- Removed a caveat that can go in the PR comments - geojeff
- Update css to make side nav full height. Add js to open/close section subnav - Josh Cartmell
- Don't animate the admin menu when first loading - Stephen McDonald

- Better defaults for non-english language codes - Stephen McDonald
- Added setting `ADMIN_MENU_COLLAPSED` for controlling admin side-nav behaviour - Stephen McDonald
- Remove dashboard link from admin menu - replace with cliackable Mezzanine title - Stephen McDonald
- Highlight selected item in admin nav - Stephen McDonald
- Move site/lang drop-downs into top nav bar - Stephen McDonald
- Bigger buttons throughout admin - Stephen McDonald
- Move multi-site docs into new section - Stephen McDonald
- Fix tuple/list setting handling - Stephen McDonald
- Allow page subclasses to implement `get_template_name` to customize template selection in the page view. Closes #957 - Stephen McDonald
- Use dark chosen plugin for top nav dropdowns in admin - Stephen McDonald
- Use Mezzanine's branding for the admin nav title - Stephen McDonald
- Consistent message styling in admin - Stephen McDonald
- Mentioning `PAGE_MENU_TEMPLATES_DEFAULT`. Updated the *Filtering Menus* section to mention the *PAGE_MENU_TEMPLATES_DEFAULT* setting which controls what entries get pre-selected in the admin section - Philip Mateescu
- Apply jquery chosen to other selects in the admin - Stephen McDonald
- Remove broken/unnecessary jquery resize plugin - Stephen McDonald
- Don't use sans-serif as the admin font (should fall back to Arial) - Eduardo Rivas
- Adjust CSS selector and colors for dark language selector. The dropdown in the admin login was being targeted as a dark dropdown - Eduardo Rivas
- Fix admin nav margin with alert messages - Stephen McDonald
- Use bigreason tinymce theme - Stephen McDonald
- Actually apply tinymce skin - Stephen McDonald
- Make admin panel collapsible to the left with a nice animation - Eduardo Rivas
- Make the admin panel hidden/show state persist in between page loads. This one uses localStorage instead of cookies - Eduardo Rivas
- Make the admin menu toggle panel a bit more subtle - Stephen McDonald
- Make sure the messages bar responds to panel toggling - Eduardo Rivas
- Fix regression in save bottom of change-list view - Eduardo Rivas
- Apply z-index to the whole side panel, not just the inner list - Eduardo Rivas
- Handle edge case of bottom controls in settings page - Eduardo Rivas
- Fix message margin with collapsed nav - Stephen McDonald
- Deal with POSTS in ssl middleware. Closes #975 - Stephen McDonald
- Fix POST handling in ssl middleware - Stephen McDonald
- Allow authentiated users to undo their ratings - Stephen McDonald
- Messages shouldn't cover admin nav when scrolling horizontally - Stephen McDonald
- Added support for search fields across model relations - Stephen McDonald

- Fix device handling for non-ascii user agents. Closes #976 - Stephen McDonald
- Bring back admin meta title - Eduardo Rivas
- Added distinct to search queryset, to prevent relation spanning search fields from producing duplicate results - Stephen McDonald
- Fix non-ascii form response CSV downloads on Python 3 - Stephen McDonald
- Update test Django version - Stephen McDonald
- Only attempt second phase cache render on valid http status. Closes #967 - Stephen McDonald
- Fix deprecated depth arg to `select_related` - Stephen McDonald
- Port over `FORMS_EXTRA_WIDGETS` feature from forms-builder - Stephen McDonald
- `unicode_literals` shouldn't be used in migrations. Because strings are passed to type(), which requires a non-unicode. string in python 2. Fixes #871 refs toastdriven/django-tastypie#1007 - Gavin Wahl
- Add `FORMS_EXTRA_WIDGETS` to Mezzanine's settings - Stephen McDonald
- Changed fabfile `LIVE_HOSTNAME` setting to DOMAINS, and handle multiple domains for a single project - Stephen McDonald
- Mention `parent.page` in the menu variables docs section - Stephen McDonald
- Only use `RICHTEXT_WIDGET_CLASS` if a RichTextField's widget is not. specified in the form's Meta - Alex Hill
- Add tests for RichTextField - Alex Hill
- update the migration files of blog and generic to use custom user model - Ziwei Zhou
- Handle removal of `CurrentSiteManager._validate_field_name()` in Django 1.7 - Stephen McDonald
- Better line break and HTML entities handling in form emails. Support line breaks in HTML email (so that they don't collapse when. rendered) and HTML entities in plain text email (unescape them to make. them legible) - Dominique Bischof
- Provide separate overridable email templates for the extra recipients in `mezzanine.forms` - Stephen McDonald
- added padding functionality to thumbnail tag - Bryan Clement
- removed rogue print statement - Bryan Clement
- Support hard-coded URLs in `LOGIN_REDIRECT_URL`. Closes #992 - Stephen McDonald
- Port forms-builder fix for large file uploads - Stephen McDonald
- Allow subclasses of PageMiddleware to be used in page view check - Stephen McDonald
- Clean up thumbnail padding - Stephen McDonald
- Correctly handle 404 exceptions for static files during development - Stephen McDonald
- Store thumbnails in individual directories named after the original filename, so that we can accurately delete them from within filebrowser where necessary. Closes #995 - Stephen McDonald
- Update thumbnail test path - Stephen McDonald
- Disable the 'Add' option of comments in the admin, because the program crash when you try to create a new comment, is better delete this option for avoid this problem - Jesus Armando Anaya Orozco
- Add a new site. Add new site created with Mezzanine - Anthony

- converted ratios to floats. ratios were integer division, which didn't work as desired. casting the numerators to floats cures this ailment - lykkin
- added padding color option into thumbnail tag - lykkin
- Document optional thumbnail tag args - Stephen McDonald
- Added `USE_L10N` check for language dropdowns in admin - Rocky Meza
- Fix `USE_L10N` check for admin language selector - Stephen McDonald
- Handle null descriptions ofr settings in conf form - Stephen McDonald
- Catches exceptions in thumbnail `image.convert` - Doug Evenhouse
- Prevent some warnings during tests - Stephen McDonald
- Restructure content architecture docs - lead in with describing how Page/Displayable are built and used - Stephen McDonald
- Move `short_url` generation code from template tag to model, so we can use it for tweets - Stephen McDonald
- Added TweetableAdminMixin and applied to BlogPostAdmin - used for sending tweets from admin - Stephen McDonald
- Updated docs for twitter integration - document setup for sending tweets plus move to own section - Stephen McDonald
- Move the side-menu directly where it should be in the admin template so we don't have to juggle it with jQuery - Stephen McDonald

Version 3.0.9 (Feb 11, 2014)

- Add noindex tag to search results. Search results shouldn't normally be indexed, given that they can be generated for arbitrary queries. This patch adds a noindex tag to the head - Samir Shah
- Fix serving uploaded files during development - Stephen McDonald
- Fix static files handling - Stephen McDonald
- Support collapse classes in admin. Closes #943 - Stephen McDonald
- Add some HTML5 tags to `RICHTEXT_ALLOWED_TAGS`. Hi,. This patch adds the following tags to `RICHTEXT_ALLOWED_TAGS`: * article. * aside. * figure. * caption. * header. * footer. * nav. * section. , all of which are becoming increasingly common, and should all be safe to allow. Thanks! - Samir Shah

Version 3.0.8 (Feb 05, 2014)

- Use binary mode when opening the image to be copied on a remote server - Sylvain Fankhauser
- Fixed regression when validating displayable content field against published status - Stephen McDonald
- Fix static proxy handling - Stephen McDonald
- Fix string checking in page processors. Closes #931 - Stephen McDonald
- Handle the different method of importing filebrowser urls when the upstream version is installed. Closes #925 - Stephen McDonald
- Rename account URL prefix to match Django's by adding the plural s. Catch and redirect any old urls to the new scheme - Sam Kingston
- Tidy up old account redirect view - Sam Kingston

- Added tests to `static_proxy` - Alejandro Peralta
- Handle next param in `old_account_redirect` - Stephen McDonald
- In Mezzanine's runserver, also serve up static files in `STATIC_ROOT` - Stephen McDonald
- Bump filebrowser-safe to 0.3.2 - Stephen McDonald
- Use `file.name` instead of `file.url` for getting name of file in GalleryImage. *file.name* is universal accross storages and can be used not only with. FileSystem storage but with S3 storage as well (*file.name* is used to. create the url). Also with using *name* instead of *url* we will not have. problems with encoding and python version - Venelin Stoykov

Version 3.0.7 (Feb 02, 2014)

- check if the user model is already registered. Fix usage of overwriting the UserProfileAdmin because it will be under `mezzanine.accounts` in `INSTALLED_APPS` - Sebastian Clemens
- check django version instead of ImportError. Closes `https://github.com/stephenmcd/mezzanine/issues/893` - Sebastian Clemens
- Use space indent instead of tab - Sebastian Clemens
- Do not show the `in_menus` field in page admin if `PAGE_MENU_TEMPLATES` is empty - Laurent Prodon
- Fix admin password reset page. Closes #909 - Stephen McDonald
- Clean up `PAGE_MENU_TEMPLATES` check in page admin - Stephen McDonald
- Fix failing account tests for foreignkey and date/datetime fields on user profile models - Stephen McDonald
- menu: include parent page in context - Laurent Prodon
- menu: modify doc accordingly - Laurent Prodon
- Handle multiple ip addresses in `mezzanine.utils.views.ip_for_request` - Stephen McDonald
- Fix handling for schemeless static urls in static proxy - Stephen McDonald
- fix on python3 loads() required string found bytes <- reads(). adding `.decode("utf-8")` to support python3 - Simone Federici
- Fixed bug in `processor_for` if it receives an unicode slug - Vindio
- Replaced unicode string for `python3.x` compatibility - Vindio
- Previous commit breaked test for `python2.x` - Vindio
- Previous commit broke test for `python2.x` - Vindio
- Dont show comments in blogpost list if they are not allowed - Laurent Prodon
- wrap ratings in a block in blogpost detail template - Laurent Prodon
- More Django 1.6 generic relation regressions - underlying sql has changed, returning duplicates, so apply distinct for correct results. Closes #918 - Stephen McDonald
- Fix tag cloud padding - Stephen McDonald
- Ensure filtering blog posts by keyword uses the correct content type - Stephen McDonald
- Much simpler keywords lookup fix - Stephen McDonald
- Don't assume static proxy url param contains a host. Closes #920 - Stephen McDonald
- Added test to check that /static/ as a prefix of url is removed - Alejandro Peralta

- Fix static proxy test for Python 3 and remove fb dependency from test - Stephen McDonald

Version 3.0.6 (Jan 17, 2014)

- Don't require specifying filebrowser urls separately in a project's urlconf - Stephen McDonald
- import the user model from settings - Sebastian Clemens
- Improved support for determining timezone with tzlocal lib - Stephen McDonald
- New setting `EMAIL_FAIL_SILENTLY` for controlling the default `fail_silently` arg in sending email - Stephen McDonald

Version 3.0.5 (Jan 12, 2014)

- Fix bytes/string handling in Python 3 `static_proxy`. Closes #866 - Stephen McDonald
- Move the file browser url to i18n patterns - Sebastian Clemens
- Use the new template tag for url generation - Sebastian Clemens
- Add a given query string to url - Sebastian Clemens
- Move jQuery setup for ajax/csrf outside of `document.ready` handler - Stephen McDonald
- In overextends template tag, don't assume extends node is first in node list - Stephen McDonald
- Fix bytes/str handling in disqus SSO - Stephen McDonald
- Communicate the requirement for twitter credentials on changing the default query in the help text - Stephen McDonald
- Slight increase to font size throughout the admin - Stephen McDonald
- Add url field type to anyFieldsDirty in `dynamic_inline.js` - Adrian Carpenter
- Patch all migrations to remove generic fields - these break with Django 1.6 and aren't actually necessary with the latest version of south - Stephen McDonald
- Added two more HTML5 input types in `dynamic_inline.js` - Adrian Carpenter
- `dynamic_inline.js` updated for HTML5 input types in Django 1.6 - Adrian Carpenter
- Remove all the `frozen_by_south` hacks and use it to provide a warning for old migrations - Stephen McDonald
- Add some more html5 field types to dynamic inlines JavaScript - Stephen McDonald
- Update to use multiprocessing library - tyescott
- Use pytz to determine a closest matching timezone fro `TIME_ZONE` default - Stephen McDonald
- Have RSS/Atom feed link attribute go to homepage of site - David Tomaschik
- Don't hard code rss link url - Stephen McDonald
- Use `FieldFile.url` to get full image-url in gallery. Closes #877 - Denis Cornehl
- SS: Bumped requests and requests-oauthlib versions - Sachin Shende
- Fix tuple/list setting handling. Closes #883 - Stephen McDonald
- allow <a> tags inside <button> in TinyMCE - Neum
- Add back `bootstrap-extras.js`. Menus stay open for a few hundred milliseconds after the cursor leaves them and the dropdowns in the main menu are useable when the menu is collapsed - Josh Cartmell

- Relative schemes for twitter avatar urls. Closes #878 - Stephen McDonald
- Fix some failing http status code tests that need to follow redirects when multiple languages defined - Stephen McDonald
- Make the label element optional for form fields. Labels for form fields can be disabled by setting them to an empty string. Previously this would produce the intended result visually, but left an empty label element in the HTML. This change causes the element to only be inserted if a label value exists - David Sanders
- Fix for uploading zip files to a homepage gallery - Jeff Fein-Worton
- Update to latest bleach/html5lib and enable for Python 3 - Stephen McDonald
- Fix admin menu app/model sorting for Python 3 - Stephen McDonald
- Bump version to 3.0.5 - Stephen McDonald

Version 3.0.4 (Dec 27, 2013)

- Shadow staticfiles runserver instead of `django.core` - Stephen McDonald
- Use local names for the language selector. Also dropping creation of the two variables LANGUAGES and `LANGUAGE_CODE`. because they already exists - Sebastian Clemens
- Correctly handle serving files under `MEDIA_ROOT` during development on Windows. Closes #865 - Stephen McDonald

Version 3.0.3 (Dec 25, 2013)

- Skip randomly failing settings test on Python 3 - Stephen McDonald
- Unicode fix for gallery image descriptions from filenames - Stephen McDonald
- More gallery unicode fixes - Stephen McDonald
- Fix some jQuery namespacing regressions in the admin - Stephen McDonald

Version 3.0.2 (Dec 24, 2013)

- No changes listed.

Version 3.0.1 (Dec 24, 2013)

- Fix Python 2/3 str/bytes error in cache bypass util - Stephen McDonald

Version 3.0.0 (Dec 24, 2013)

- Python 2/3 port: be conservative: no magic super() imports - Ed Schofield
- Put `__future__` imports below the # encoding: utf-8 lines - Ed Schofield
- Correctly handle page slug lookups when `APPEND_SLASH` is False - Stephen McDonald
- disqus counts should follow the protocol of the original request, see: `http://help.disqus.com/customer/portal/articles/542119` - John Henry
- Fall back to Python 2 urllib imports if needed. * Also fix one bug triggered by passing a newstr to `urllib.unquote` - Ed Schofield

- Remove obsolete `with_statement` import from `__future__` - Ed Schofield
- Always pass a native string to first arg of 3-arg type() call in `middleware.py` - Ed Schofield
- Add `absolute_import` to prevent implicit relative import of `html.py` on Py2 - Ed Schofield
- Python 2/3 compatibility for types in the settings registry - Ed Schofield
- Fix a few out-of-order `__future__` imports - Ed Schofield
- Python 3 compatibility: Use Django's newer `smart_text` and `force_text` if available. - `smart_unicode` and `force_unicode` aren't defined in `django.utils.encoding` on Py3 - Ed Schofield
- Python 3: fix `arg_names` lookup in `FormForForm.__init__` - Ed Schofield
- Python 3 compatibility: Fix `galleries/models.py:` use BytesIO etc - Ed Schofield
- Add Python 3.3 to `.travis.yml` - Ed Schofield
- Revert "Add Python 3.3 to `.travis.yml`". This reverts commit 4dee3b787d040613fa632c3300a29def955ca128. Django 1.4.x doesn't support Python 3.x, so the `.travis.yml` file needs to. specify that the combination of Python 3.3 and Django 1.4.x should not be. tested - Ed Schofield
- Change `__unicode__` -> `__str__` and add `python_2_unicode_compatible` decorator - Ed Schofield
- Disable `standard_library` import hooks for pychecker - Ed Schofield
- Add future == 0.8.2 to requirements; remove dependency for `setup.py` - Ed Schofield
- Change future dependency to >= 0.8.2 - Ed Schofield
- Add Python 3.3 back to `.travis.yml` and disable tests on `Py3.3` + Django 1.4 - Ed Schofield
- Fix location of urlparse on Python 2 - Ed Schofield
- Add Python 3 classifier for PyPI - Ed Schofield
- Prevent UnicodeDecodeError in test run on Py2 if files in the local dir have high-bit chars. - Also remove an extraneous import - Ed Schofield
- Python 3: import `local_settings` correctly from `project_template.settings` - Ed Schofield
- Work around Django / Python 2.x not supporting unicode cookie keys - Ed Schofield
- Change Py3-incompatible is-this-a-string type-check hack in `send_mail_template` - Ed Schofield
- Fix for `mezzanine.utils.email` imports - Ed Schofield
- Remove Django 1.4.x from `.travis.yml` for pull request for early testing - Ed Schofield
- `import_rss` and `import_tumblr` scripts: fix urllib imports on Py2 - Ed Schofield
- Remove all isinstance() imports (with future `v0.9`). - These are not needed in future `v0.9` - Ed Schofield
- Handle `context_data` is None in PageMiddleware - Stephen McDonald
- No need to provide `TEST_RUNNER` in settings - Stephen McDonald
- Restore 1.6 fix for generic relations - Stephen McDonald
- Don't use deprectaed depth arg for `select_related` in `page_menu` tag - use the built up list of subclass models instead, as per in the admin - Stephen McDonald

- Revert "Restore 1.6 fix for generic `relations.`". This reverts commit 19288b896a5ccb146ae8fe8e25cde5a768079c0d. `_meta.get_all_field_names()` load the app cache. This cannot be called. during the app cache loading. Those line have been posing problems as seen in: * 19288b896a5ccb146ae8fe8e25cde5a768079c0d. * d2b68151ca936422eef3d0b7cc2a8e63f5e2d4d1. * 69acbfd8f025d2b245c4c8e8ca4d1484f0c1228d - Antoine Catton
- Fix circular import problem and keep Django 1.6 compatibility. See: * f48390c4c8d61ca499f277f2ae1c2346262b949d. * 19288b896a5ccb146ae8fe8e25cde5a768079c0d. * d2b68151ca936422eef3d0b7cc2a8e63f5e2d4d1. * 69acbfd8f025d2b245c4c8e8ca4d1484f0c1228d. Thank you Gavin Wahl (gwahl at fusionbox dot com) for the solution - Antoine Catton
- Bump future version requirement to 0.9.0 - Ed Schofield
- Add Django 1.4.8 back to `.travis.yml` and disable tests on `Py3.3` + Django 1.4.8 - Ed Schofield
- Remove some unnecessary list() calls around map() results - Ed Schofield
- Allow fab remove to run when db/user don't exist - Stephen McDonald
- Allow multiple deployed projects with ssl in nginx conf - Stephen McDonald
- Fixed a few small spelling errors - Tuk Bredsdorff
- In `set_dynamic_settings` don't convert tuple settings back to tuples if they were already a list. Closes #821 - Stephen McDonald
- Python 3 fix - can't compare string and int when sorting content types in page admin - Stephen McDonald
- Don't install optional dependencies for Python 3 that aren't yet compatible with it - Stephen McDonald
- Fix string cast for file fields in thumbnail template tag - Stephen McDonald
- Overide staticfiles' runserver command and wsgi handler during development, to allow uploaded files to be served from within the static dir - Stephen McDonald
- Update refs to latest Django 1.4/1.5 versions - Stephen McDonald
- Add project path to search path in tests - Stephen McDonald
- Authenticate new user using a token instead of a password to support password-less configurations - Alex Hill
- If new user's password is empty, make this explicit by calling `set_unusable_password()` - Alex Hill
- Only load editable settings from the database - Alex Hill
- Add a test for fixed editable settings behaviour - Alex Hill
- Load settings in a separate method, emit warnings for settings that are defined twice - Alex Hill
- Assume settings pulled from the database are UTF-8 - Alex Hill
- Remove assignment to make pyflakes happy - Alex Hill
- Remove use of `from future import standard_library` for now (issue #826). - This feature in `future` is currently buggy. - The import was not actually needed by some modules anyway - Ed Schofield
- Remove deprecated assert methods - Stephen McDonald
- Clean up editable setting loading - Stephen McDonald
- Move special-case bytes conversion to `_load` method - Alex Hill
- Add test for special-case bytes conversion - Alex Hill
- Fix inline editing response. Closes #829 - Stephen McDonald
- Upstream update to Bootstrap 3.0.2 - Eduardo Rivas

- Update footer link to point to the Bootstrap 3 site - Eduardo Rivas
- Inline Admin: Convert `editable_form.html` to BS3 - Eduardo Rivas
- Make auto-generated slugs propagate. Fixes #831 - Alex Hill
- Use reverse instead of `models.permalink` in `BlogPost.get_absolute_url` - Stephen McDonald
- Allow all models subclassing Displayable to be searched by setting `SEARCH_MODEL_CHOICES` to None - Stephen McDonald
- Update search engine docs - Stephen McDonald
- Fix some type handling for settings in `mezzanine.conf` - Stephen McDonald
- More Python 3 fixes for types in `mezzanine.conf` - Stephen McDonald
- Allow specifying optional left and top values to the thumbnail tag which control the centering of the thumbnail. If non default values are used update the thumbnail name - Josh Cartmell
- Allow None to be used as a `per_page` arg for pagination, to bypass pagination altogther - Stephen McDonald
- Force string type names for HTML5 form fields - Stephen McDonald
- Clean up positioning code in thumbnail tag - Stephen McDonald
- added three classes “left”, “middle” and “right”. these classes are used by cartridge to use the entire space in checkout process - Sebastian Clemens
- fix on `__admin_media_prefix__` about the trailing slash - Alexandre Hajjar
- Namespace jQuery in the admin pages to prevent conflicts - Zachery Metcalf
- Upstream update to Bootstrap 3.0.3 - Eduardo Rivas
- Use the new BS 3.0.3 colors for error fields - Eduardo Rivas
- Py3k compatibility fixes in two-phase rendering - Alex Hill
- Update `forms.py`. Added a check to see if the initial value for a field is a manager - Tim Harton
- Fixed multipleschoiceselect error with profiles - Tim Harton
- Slightly cleaner related field check in profile form - Stephen McDonald
- Add a {% block %} to `.form-actions` in `account_form.html` for Cartridge. to extend. This makes it easier for Cartridge to insert an “Order History” button - Eduardo Rivas
- Increased margin around buttons in the user panel. Makes it look less. crammed - Eduardo Rivas
- Render form errors with a new {% `errors_for` %} template tag. This new template tag accomplishes three things: - Fixes the bug of multiple form error messages appearing when using. {% `fields_for` %}. This bug was intruduced in [this `commit](https://github.com/jerivas/mezzanine/commit/323660db5bee7e21358315c4e247eaa8ee` and was discovered when [migrating Cartridge to `BS3](https://github.com/clemensbasti/cartridge/pull/1)` - Decouples error message rendering from form field rendering, with. the added flexibility of placing the error messages wherever we want. - Creates a new template `(includes/form_errors.html)` as the single. location to control form error rendering through all Mezzanine and. Cartridge - Eduardo Rivas
- Admin href disqus recent comments to https or http via double slash. Changed how the disqus widget in the admin panel loads its recent. comments. This will fix any errors for loading insecure content - Daniel Lawrence
- `verbose_names` should be capitalized when output. The convention is to always use lowercase `verbose_names` and capitalize. in the template where necessary. `<https://docs.djangoproject.com/en/1.0/topics/db/models/#verbose-field-names>`.

> The convention is not to capitalize the first letter of the. > `verbose_name`. Django will automatically capitalize the first letter. > where it needs to - Gavin Wahl

- Fix a bunch of regressions from namespacing jQuery in admin - Stephen McDonald
- Add django 1.6.1 to supported/tested versions - Stephen McDonald
- Move sitemap generation logic for Displayable instances into DisplayableManager - Stephen McDonald
- Added the url/view for `displayable_links.js` which is then used by TinyMCE to render a list of site links to use - Stephen McDonald
- In TinyMCE, don't convert relative urls to absolute - Stephen McDonald
- Added `ALLOWED_HOSTS` configuration for Django. As it's part of the default django settings, we also need this in mezzanine - Sebastian Clemens
- Python 2/3 fix for forms export - Stephen McDonald
- Added LANGUAGES to settings, since they're needed for multilingual support - Sebastian Clemens
- Added `django.middleware.locale.LocaleMiddleware` to `MIDDLEWARE_CLASSES` - Sebastian Clemens
- Added a language selector field to the login and admin site - Sebastian Clemens
- Mention Widgy in the list of third-party modules - Gavin Wahl
- Only show admin language selector when multiple languages configured - Stephen McDonald
- Don't use future's int for int settings. Closes #855 - Stephen McDonald
- In admin customization section of docs, mention `in_menu` method on admin classes for controlling `ADMIN_MENU_ORDER` behavior - Stephen McDonald
- Move requirements file for `project_template` into project root to better conform with hosting providers like Heroku. Closes #859 - Stephen McDonald
- Broader excpetion handling for importing bleach since its deps aren't Python 3 ready - Stephen McDonald
- Fix for django-debug-tolbar 1.0 which prevents later middleware from running - Stephen McDonald
- Add config for wheel distribution - Stephen McDonald
- Remove use of deprecated simplejson module and clean up other imports - Stephen McDonald
- Provide read-only twitter settings for the default query - Stephen McDonald

Version 1.4.16 (Sep 30, 2013)

- Revert broken static proxy change - Stephen McDonald
- Better fix for static proxy urls - Stephen McDonald

Version 1.4.15 (Sep 29, 2013)

- Blog: Generate RSS and Atom feeds through `richtext_filters` - Eduardo Rivas
- Delete BS2 static resources. Add BS3 resources: css, js and fonts - Eduardo Rivas
- Migrated `base.html` and all it's includes to BS3 - Eduardo Rivas

- Strip `STATIC_URL`, leading / from proxied URLs. `STATIC_URL` often contains host or `generic_host` (esp. if `STATIC_URL` is a. path on the same domain), so it needs to be removed first to ensure it. is removed completely. Also removed leading '/' from URL, since it. appears staticfiles doesn't like absolute paths - Adam Brenecki
- Added a function `mezzanine.utils.urls.next_url` which is used to retrieve redirect URLs from a request's next param, while verifying that the redirect URL is valid - Stephen McDonald
- Fix min Dajngo version - Stephen McDonald
- Use `request.get_host` rather than `request.META['HTTP_HOST']` - Stephen McDonald
- Fix Django version for travis - Stephen McDonald

Version 1.4.14 (Sep 14, 2013)

- Blog: Catch exception if a non-existent month is requested from the archives - Eduardo Rivas

Version 1.4.13 (Sep 10, 2013)

- Allow for there being no "errors" in the twitter api response. When a successful "user" query to the twitter api is completed the json that is returned is a list. In order to validate the response the code tests to see if it was a dictionary with an "error" key. However passing a string as a index to a list will raise a TypeError, which was not being caught by the "except" clause. I have added TypeError to the list of items being caught. There are of course other ways of verifying the response but I think just adding the TypeError is in keeping with what you have already done. For reference, here is what I was seeing: > python `manage.py poll_twitter` –traceback –force. Traceback (most recent call last): ,. File `"/,/lib/python2.7/site-packages/mezzanine/twitter/models.py"`, line 74, in run. raise TwitterQueryException(tweets["errors"][0]["message"]). TypeError: list indices must be integers, not str - David Higgins
- Fix tag cloud factor in generic app - Stephen McDonald

Version 1.4.12 (Aug 27, 2013)

- Remove bad 1.6 handling - Stephen McDonald
- Fix settings context processor for email template contexts when cache installed - Stephen McDonald

Version 1.4.11 (Aug 26, 2013)

- Added mezzatheme themes marketplace to features list - Stephen McDonald
- Method to load all symbols of all files in a submodule - Thomas Rega
- Use new decorator `"richhtext_filters"`. The decorator `"richtext_filter"` is marked as deprecated - Thomas Rega
- Move gallery related tests into the app directory. If the app is not installed, the tests are not executed - Thomas Rega
- Move blog related tests into the app directory. If the app is not installed, the tests are not executed - Thomas Rega
- Move page related tests into the app directory. If the app is not installed, the tests are not executed - Thomas Rega

- Move account related tests into the app directory. If the app is not installed, the tests are not executed - Thomas Rega
- Move form related tests into the app directory. If the app is not installed, the tests are not executed - Thomas Rega
- Move core related tests into the app tests directory. These tests do not belong direct to an specific app, so they stay. in the core app directory for now - Thomas Rega
- Simplify new test module structure - Stephen McDonald
- Provide a common TestCase class for all app specific tests, for features such as admin user plus debug cursor for query tracking. Also consistent naming for tests - Stephen McDonald
- Move tons of tests into their correct apps - Stephen McDonald
- Patch `jquery.overlay` with `jquery.browser` support to work with new jQuery versions. Closes #701 - Stephen McDonald
- Force tinyMCE save in `dynamic_inline.js` to avoid issues with richtext fields in dynamic inlines and ordering values not correctly handled. Closes #731 - Stephen McDonald
- Update dev status classifier in `setup.py` - Stephen McDonald
- Remove inclusion of `mezzanine.accounts` when testing - Zach Gohr
- Inject all Mezzanine apps into the temp settings module when Mezzanine itself is being tested - Stephen McDonald
- Use setuptools to test on travis - Stephen McDonald
- Apply skipTests throughout different tests where apps are coupled - Stephen McDonald
- `setup.py` specifies the test deps so we don't need to grab them for travis now - Stephen McDonald
- Update `send_mail_template` to pass optional email headers to the EmailMultiAlternatives constructor. Rather than having the form `page_processor` send mail from user submitted email addresses (if present) have it specify the Reply-To header - Josh Cartmell
- Get rid of `FORMS_DISABLE_SEND_FROM_EMAIL_FIELD`, always add the Reply-To header if there is an `email_to` - Josh Cartmell
- Adding template accessible settings into context for rendering templates for emails - Danny Sag
- Handling case when diff between min and max count is smaller than size. Weights were not calculated correctly when difference between *max_count* and *min_count* was smaller than *settings.TAG_CLOUD_SIZES*. Changed calculation of weights to use floating point arithmetic. The results of weight calculations using old and new code are shown below: `http://ideone.com/fXs5aG` - Ahmet Bakan
- Adding `.control-label` to `form_fields.html` - Troy Harvey
- Be a bit more explicit with request arg in settings context processor - Stephen McDonald
- Added mezzanine-meze to third-party apps list - Stephen McDonald
- Added support for Django 1.6. - The situations in which `contribute_to_class` is called have changed. - Fixed DeprecationWarning about simplejson. - Explicitly set the `TEST_RUNNER` to the pre 1.6 one. - Set default=False on BooleanField - Rocky Meza
- Keep django version pinned - Stephen McDonald
- Ensure correct arguments are used when returning a Page in place of a 404 from a non-page urlpattern - Ben Ledbury
- Better error propagation when querying for tweets - Stephen McDonald

- Added –force option to `poll_twitter` command which will query for tweets on all queries - Stephen McDonald
- Catch and show twitter query errors in management command, and allow continuing - Stephen McDonald
- Allow twitter queries to gracefully fail in templates - Stephen McDonald
- Bump requeusts-oauthlib version. Closes #764 - Stephen McDonald
- Exempt Link pages from 404 Page replacement - Ben Ledbury
- Changed 'form' to `'editable_form'` to fix naming conflict. Editable JS no longer being pulled in - Nicole Harris
- Don't quote bullets,. The indentation causes the bullets to be treated as blockquotes - David Winterbottom
- ,but do quote quotes :grinning: - David Winterbottom
- Use correct comment field name in akismet API. Closes #768 - Stephen McDonald
- Added TimeStamped model mixin to Displayable, for created/updated timestamps on all models. Closes #661 - Stephen McDonald
- Allow account signups with profiles containing non-nullable fields. Closes #749 - Stephen McDonald

Version 1.4.10 (Jul 29, 2013)

- Added `window.__language_code` variable to admin and inline loader - Artem Gluvchynsky
- Better error message for page models that get removed from `INSTALLED_APPS`. Closes #722 - Stephen McDonald
- Allow initial user creation in syncdb when a profile model is managed by migrations and doesn't yet exist - Stephen McDonald
- Looser AJAX response check for page reordering. Closes #727 - Stephen McDonald
- Allow key settings to be defined in fab conf and injected into live `local_settings` module - Stephen McDonald
- Added valid Polish messages for mezzanine/core. Closes #729 - Marek Wywiał
- add a `tox.ini` config file - jferry
- Use protocol-relative URL to avoid SSL warnings - Vinod Kurup
- Make running fabfile outside project root optional, since it conflicts with importing the fabfile into other fabfiles - Stephen McDonald
- Specify minimum version of pytz requirement - Vinod Kurup
- Fixed view and JS to be compatible with jQuery 1.8 - Ethan Goldstine
- Fix `gravatar_url` tag for non-ascii email addresses. Closses #721 - Stephen McDonald

Version 1.4.9 (Jul 10, 2013)

- Allow deployments to be run from project subdirectories - Stephen McDonald
- Add support for `settings.RICHTEXT_FILTERS`. `RICHTEXT_FILTERS` is a list of items that are valid for the. `RICHTEXT_FILTER` setting. The plural version takes precedence if it is. available and non-empty. Each item in `RICHTEXT_FILTERS` is applied in. order. An alias for the `richtext_filter` template filter has been added to. match the plural nature of the new setting: `richtext_filters` simply. calls on `richtext_filter` for its output - Tim Valenta

- Fixed blog post categories list in `blog_post_list.html` template - Artem Gluvchynsky
- Removed redundant jQuery media from KeywordsWidget - Artem Gluvchynsky
- Use urljoin in `Page.get_absolute_url` for link pages - Dheeraj Sayala
- RTL: fix position of changelink icon in page tree after recent changes. Problem introduced in aec1a0462b60, which solves an issue due to long. page names - Ahmad Khayyat
- Comma separate categories in blog post listing - Stephen McDonald
- Update docs for new `RICHTEXT_FILTERS` setting - Stephen McDonald
- Properly deprecate `RICHTEXT_FILTER` setting in favour of `RICHTEXT_FILTERS` (plural) setting - Stephen McDonald
- Update templates to use new `richtext_filters` (plural) tag - Stephen McDonald
- Allow a single BCC address in addition to list/tuple. BCC fails if the `addr_bcc` argument is a single address rather than a list/tuple. This commit wraps a single address in a list to fix this problem - Alex Hill
- Make sure `request._messages` has been set before trying to access it. This is for cases where the MessageMiddleware hasn't had a chance to. run, `e.g` when a previous middleware returned an exception - Gu1
- Use a separate key setting for nevercache tokens - Stephen McDonald
- Add `is_current_parent` on pages filtering - Antoine Catton
- Remove field related to Django built-in user model. South is expecting those fields to be in the database. So it is. selecting them, since some custom user models don't have them, this can. break this migration - Antoine Catton

Version 1.4.8 (Jun 27, 2013)

- Fix nginx config to work on more recent ubuntu versions. Not sure how backwards compatible this is. Please see: `http://stackoverflow.com/questions/8768946/dealing-with-nginx-400-the-plain-http-reque` - David Novakovic
- dynamically generate top margin of admin content area - Andromeda Yelton
- contentMargin out of global namespace - Andromeda Yelton
- Force csrf token generation on every request with cache middleware. Closes #676 - Stephen McDonald
- Use a more explicit name in `PageAdmin.get_content_models` which won't collide with a commonly used field name such as name - Stephen McDonald
- Don't use `ugettext_lazy` for form field labels since Django will double-escape them. Closes #682 - Stephen McDonald
- Move case-insensitive keyword creation into KeywordManager, and allow for duplicate results. Closes #679 - Stephen McDonald
- Fix `ADD_PAGE_ORDER`. Closes #681 - Stephen McDonald
- Fix uses of next param for redirects where param exists but value is empty - Stephen McDonald
- Revert fix to #594 #677 - causes issues with status messages - Stephen McDonald
- TagCloser - don't close br and image tags - John Groszko
- Test changes to TagCloser - John Groszko
- Clean up some docstrings - Stephen McDonald

- When using search against an abstract model (eg Displayable), filter the combined models searched against by the models represented in the SEARCH_MODEL_CHOICES setting. Closes #684 - Stephen McDonald
- Add a note to search docs about SEARCH_MODEL_CHOICES affecting abstract search behaviour - Stephen McDonald
- Added missing class to collapsible navbar that affected nested menus - Jason Wong
- SS: Moved to the original - Sachin Shende
- long title : break words on pages tree - jferry
- SS: Changes done to Twitter app to upgrade to API 1.1. 1. Added requests==1.2.3 and requests-oauthlib==0.3.2 to the dependency list. 2. Added 4 new keys to the settings. 3. Changed models to use new authentication for Twitter API, changed urls and other changes to parse the response - Sachin Shende
- use of staticfiles to get url to tinymce - Eduardo S. Klein
- Just added my Website to the gallery ;) - Rafael Beckel
- More consistent names and validation of new twitter settings - Stephen McDonald
- Document new requirements for Twitter API - Stephen McDonald
- Fix for Issue #691 - ACCOUNTS_APPROVAL_REQUIRED bypasses ACCOUNTS_VERIFICATION_REQUIRED - Ling Thio
- Provide better default for FILE_UPLOAD_PERMISSIONS - Stephen McDonald
- fixed little firefox bug - jferry
- Improved ssl cipher settings in default nginx conf - Stephen McDonald

Version 1.4.7 (May 17, 2013)

- Added the ACCOUNTS_NO_USERNAME setting, which will hide the username field from signup/update forms, but still generate a unique username for use in profile view slugs - Stephen McDonald
- Allow querystring vars to be excluded from pagination links - Stephen McDonald
- Missing migration on site perms. Closes #655 - Stephen McDonald
- Added support for setup.py test - Stephen McDonald
- Pass in the user to page.get_ascendants in the page view. This will allow previewing of the unpublished children pages of. unpublished parent pages. fixes #653 - Rocky Meza
- Lowered MAX_POSTS_PER_CALL to 20; Added support for question/answer posts - Jeff Fein-Worton
- Use a context instance when rendering page menus, huge performance boost - Stephen McDonald
- Fixed rss import errors - Andrey Zhukov
- Fixed the igrations to be able to run with a custom user model. This uses a pattern copied from django-reversion: https://github.com/etianen/django-reversion/blob/master/src/reversion/migrations/0001_ - Rocky Meza
- Add addr_bcc arg to send_mail_template. This accommodates the new setting SHOP_ORDER_EMAIL_BCC in Cartridge - Alex Hill
- Fix lookup for username generation when ACCOUNTS_NO_USERNAME is True, closes #664 - Stephen McDonald
- Fixed 0005 migration wrt custom user models - Rocky Meza

- Correctly validate float settings in `mezzanine.conf` - Stephen McDonald
- Added some validation in the createdb command that fails if a Mezzanine table exists, to prevent people from running it and accidentally faking new migrations that need to be run - Stephen McDonald
- `mezzanine/accounts/templates/email/account_approved.html`: removed the extra. "http://" - Alexandre Hajjar
- Make fabfile work in Windows. Two small changes allow deployment via Fabric from Windows: * Use `posixpath.join` instead of `os.path.join` to construct all paths destined for the remote machine. * Check for `"fab-file.py"` as well as "fab" in `sys.argv`, to handle the way setuptools-generated command-line scripts work in Windows - Alex Hill
- Fix urlpattern for archive year - Stephen McDonald
- Hide printing `STATIC_ROOT` in deploys - Stephen McDonald
- Added paragraph to `mezzanine/docs/user-accounts.rst` about `ACCOUNTS_NO_USERNAME`. setting - Alexandre Hajjar
- Used `username_label` variable in the PasswordResetForm label. `(accounts/forms.py)` - Alexandre Hajjar
- Pin html5lib, see `https://github.com/jsocol/bleach/issues/94` - Stephen McDonald
- Added an extra safeguard for type errors in editable settings - Stephen McDonald

Version 1.4.6 (Apr 27, 2013)

- Fix `set_dynamic_settings` for projects without `AUTHENTICATION_BACKENDS` defined - Stephen McDonald
- Provide meaningful exception when dotted import fails - Sam Kingston
- SS: Line 12 `dsq.src` changed to include https if the site is runnning on SSL. Comments do not appear if the site is running on SSL and js link is http - Sachin Shende
- Adding Golds Gym Utah - Josh Batchelor
- If `static_directory` does not exist, create it. Instead of trying to tar the static directory (which. fails when the dir does not exist), we create it when. is missing - José Aliste
- Hack for generic fields that allows MySQL migrations to run correctly - Stephen McDonald
- Don't assume a site exists in some older migrations - Stephen McDonald
- Use consistent language for 'log in / sign up' - Stephen McDonald
- The `db_type` field must take a second 'connection' argument, even though unused, otherwise one gets an 'unexpected keyword argument connection' TypeError - Marcos Scriven
- Added a port of Django's RedirectFallbackMiddleware with support for Mezzanine's multi-site handling. Closes #535 - Stephen McDonald
- Changelist view signature change to work with reversion - Thejaswi Puthraya
- Mark redirects middleware as unused if redirects not installed - Stephen McDonald
- Add special handling in PageMiddleware for non-page views that raise 404s, but do so with a valid page slug - in this case, we use the page view instead, which allows pages to be created that may match non-page urlpatterns. Closes #561 - Stephen McDonald
- Fix CSRF token generation when cache is enabled, should solve #632 - Gu1
- Be more explicit in checking for a test run management command - Stephen McDonald

- Add missing reference for link - Thibault J.
- Fix `SearchableManager._search_fields` incorrectly persisting across managers for model subclasses. Closes #633 - Stephen McDonald
- Add code of conduct - Ken Bolton
- New mezzanine-file-collections reference. mezzanine-media-library got renamed to mezzanine-file-collections. The reference was updated in this commit - Thibault J.
- Added the bool setting `ACCOUNTS_APPROVAL_REQUIRED`, which defaults to False and when set to True, sets newly created public user accounts to inactivate, requiring activation by a staff member. Also added the setting `ACCOUNTS_APPROVAL_EMAILS` which can contain a comma separated string of email addresses to send notification emails to each time a new account is created and requires activation. Closes #417 - Stephen McDonald
- Document the new account approval feature - Stephen McDonald
- Better name for `emails_list` -> `split_addresses` - Stephen McDonald
- Fix thumbnail template tag for palette-mode images. Closes #636 - Stephen McDonald
- Added `select_related` for user in `blog_recent_posts` template tag - Stephen McDonald
- Fix lookup of initial data in from-builder forms, and correctly handle initial values for checkbox fields - Stephen McDonald
- Allow forms-builder forms to contain template code for default values - Stephen McDonald
- Provide more granular export filtering for multiple-choice fields in forms-builder export, eg matches/doesn't match any/all selected choices, and also allow range filters to use only one boundary - Stephen McDonald
- Fix `static_proxy` to work with //host `STATIC_URL`s. `STATIC_URL = '//mybucket.s3.amazonaws.com'` would break the `static_proxy` prefix stripper, and therefore break tinyMCE plugins. This fix adds proper handling of generic-protocol hostnames to the `static_proxy` view - Gabe Smedresman
- Reorder blog and accounts patterns in `mezzanine.urls` to allow for projects with a blog homepage that also have accounts enabled - Stephen McDonald
- Fix handling of paths in zip imports in galleries app - Stephen McDonald
- accounts: properly reject multiple matching e-mail addresses. Django allows multiple Users with the same e-mail address; the existing. form can throw MultipleObjectsReturned when get(email=email) is called. against such a dataset - mike wakerly
- Added default wsgi script to project template - Stephen McDonald
- Only add input-xlarge on inputs without a class attribute, fixes #643 - Gu1
- Replaced the `BLOG_URLS_USE_DATE` setting with a new `BLOG_URLS_DATE_FORMAT` setting - it can contain the string year, month, or day, which controls the date granularity in blog post URLs - Stephen McDonald
- Editable settings refactor - this change is to clear up confusion around editable settings being defined in a project's settings module. Previously when this happened, the `settings.py` module value would only serve as a default, which would be superceded by the db editable value as soon as the settings admin form is first saved. To address this, this change means that editable settings defined in the project's `settings.py` module now mark the setting as not editable, so it will always be the value used. We also include some handling for the migration case so that even with this change, editable settings already in the db that have a `settings.py` value defined will still use the db value and provide a warning - Stephen McDonald
- Revert the handling for still using db values for editable settings with `settings.py` values defined, since it basically defeats the purpose if a `settings.py` value is added once a project is live - Stephen McDonald

- New `INLINE_EDITING_ENABLED` setting doesn't need to be editable - Stephen McDonald
- Don't force lowercase keywords. Closes #647 - Stephen McDonald
- Allow blog feed title and description to be overridden - Stephen McDonald
- Use callable description in atom rss feed - Stephen McDonald
- Properly escape comments in `comment_filter` template tag - Stephen McDonald

Version 1.4.5 (Apr 02, 2013)

- Fix some static urls in admin to support external storage backends - Stephen McDonald

Version 1.4.4 (Mar 30, 2013)

- Added user FK to rating model, and allow authenticated users to edit their ratings. Added new setting `RATINGS_ACCOUNT_REQUIRED` to allow ratings to behave like comments, where requiring authentication can store post data in session until user logs in to complete the rating - Stephen McDonald
- If RichTextPage is unregistered in the admin, have the page add link in the dashboard go to the page tree - Stephen McDonald
- Let's go back to a fixed-width navbar - Stephen McDonald
- Give the navbar some more space - Stephen McDonald
- Docs for using the Media Library browse dialog in custom widgets - Ahmad Khayyat
- Added the `ADD_PAGE_ORDER` setting, which is a sequence of `app_label.object_name` values of Page subclasses, for defining the ordering used in the add drop-down on the admin page tree - Stephen McDonald
- Use CSS instead of JavaScript for the admin app dropdowns. There were some bugs with the dropdowns when they were in JavaScript: 1. When you open a dropdown and then scroll, the dropdown would stay. put, instead of following the scroll. 2. The JavaScript used *.live('mouseover')* which binds to body and. wastes memory because it's fired for mouseover on every single DOM. element. 3. Occasionally, the dropdowns never disappeared even after mouseout. This commit fixes those bugs by using CSS and *:hover* instead of. JavaScript. Additionally, it simplifies the JavaScript related to. setting the href of the primary menu item links to their first child. It is a pixel for pixel match of the previous functionality and. appearance - Rocky Meza
- Update to bootstrap 2.3.1 - Stephen McDonald
- Use Django's `module_has_submodule` util in any module autodiscover scenarios (page processors, conf defaults) so we can correctly propagate real errors - Stephen McDonald
- Tighten up the search form css a bit - Stephen McDonald
- Remove the model graph FAQ since no one's actually ever asked it - Stephen McDonald
- New docs sesction, Utilities, covering the models/fields in `mezzanine.generic`, as well as some of the more useful template tags in `mezzanine_tags` - Stephen McDonald
- Django 1.5 url compatability - pahaz
- Use future lib in form entries template for backward compat - Stephen McDonald
- Fix search form HTML - Stephen McDonald
- Add `JQUERY_UI_FILENAME` setting and corresponding docs - Ahmad Khayyat
- Fix rating field lookups - Stephen McDonald
- Added domain to cache key for site ID caching - Stephen McDonald

- Added some JS to the default front-end templates that delays closing of dropdown menus in the primary nav to make them more user friendly. Closes #587 - Stephen McDonald
- Added the setting `BLOG_RSS_LIMIT` defaulting to 20, which limits the number of blog posts shown in the RSS feed - Stephen McDonald
- Update `BLOG_RSS_LIMIT` setting description to describe setting it to None for no limit - Stephen McDonald
- Make `BLOG_RSS_LIMIT` setting not editable - Stephen McDonald
- A little late here, but fix Django 1.3 support - Stephen McDonald
- Provide a default `max_length` for FileBrowseField - Stephen McDonald
- Added a website powered by Mezzanine - poptosic
- Better comment button text - Stephen McDonald
- Unicode fix for comment emails - Stephen McDonald
- Don't show site selection form when there's only one site. If there's only a single site, there's no need to show this form,. because it can't do anything - Gavin Wahl
- Only show one reply form at a time in a comment thread - Stephen McDonald
- Configurable page var names in pagination querystrings - Stephen McDonald
- Pin max Django version to 1.5.x - Stephen McDonald
- RTL: adjust admin navbar thickness after the js->css change - Ahmad Khayyat
- Provide optional template for user panel in nav - Stephen McDonald
- RTL: fix position of help icon in `filter_horizontal` m2m widget - Ahmad Khayyat
- Remove content from DisplayableAdmin's `search_fields` since content is not defined on Displayable and may or may not be present on a model that subclasses it and uses the DisplayableAdmin - Josh Cartmell
- Clean up nav version of user panel - Stephen McDonald
- Don't strip any HTML in TinyMCE since filtering is handled by bleach - Stephen McDonald
- 569 - replace uses of `STATIC_URL` in templates with 'static' template tag - endophage
- site is never assigned when hostname is set, single line addition fixes the problem - endophage
- Don't depend on unloaded comment state for determining parent-most level of replies in `comment_thread` template tag - Stephen McDonald
- Fix KeywordsField swapping of name in model's `search_fields` when a sequence is used - Stephen McDonald
- Moved the logic for building up search fields in SearchableQueryset into a new method `SearchableManager.get_search_fields`, which allows externally retrieving the search fields dict that will be used - Stephen McDonald
- Use model's `search_fields` to populate `DisplayableAdmin.search_fields` - Stephen McDonald
- Fix use of `JQUERY_FILENAME` with static template tag - Stephen McDonald
- Add compress tags to js/css in base mobile template - Stephen McDonald
- Fix empty thumbnails for fielbrowser fields in AdminThumbMixin - Stephen McDonald
- Added AJAX/JSON handling for comment/rating login redirects, and comment form errors - Stephen McDonald
- Allow migrations and fixtures to run from scratch without `mezzanine.pages` installed - Stephen McDonald
- Don't update existing ratings if their values haven't changed - Stephen McDonald

- Fix dot lookup in template settings - Stephen McDonald
- Upgrade bitly integration - added new `BITLY_ACCESS_CODE` setting to replace the old api settings - Stephen McDonald
- Upgrade `select_related` call in `recent_comments` template tag - Stephen McDonald
- Remove all use of `django.conf.urls.defaults` since we don't support Django 1.3 anymore. Closes #539 - Stephen McDonald
- Remove all special handling for Django 1.3 since it's no longer supported - Stephen McDonald
- Removed all use of Django's deprecated `ADMIN_MEDIA_PREFIX` since we no longer support Django 1.3 - Stephen McDonald
- Added keyword/category filtering to blog post admin - Stephen McDonald
- Remove the `USE_REVERSION` setting since it's incomplete - Stephen McDonald
- Remove stray deprecated `django.conf.urls.defaults` - Stephen McDonald
- Update to latest grappelli/filebrowser-safe - Stephen McDonald
- Bump `grappelli_safe` to 0.2.16 - Stephen McDonald
- Fix list/tuple handling for `AUTHENTICATION_BACKENDS` checks in `set_dynamic_settings` - Stephen McDonald
- Revert sequence settings back to tuples in `set_dynamic_settings` since some Django tests expect them to be tuples - Stephen McDonald
- Rename sr@latin locale folders to `sr_Latn` - Sebastián Ramírez Magrí

Version 1.4.3 (Feb 27, 2013)

- domain change to wdiaz - William Díaz
- Fixed dynamic admin inlines for subclasses doing fields magic. (assuming that `InlineAdmin.fields` is a user-provided list) - wrwrwr
- Mezzanine's auth backend incompatible with custom user model tests in Django 1.5 - Stephen McDonald
- Added Django 1.5 to travis config - Stephen McDonald
- Add a fallback for the newly required `ALLOWED_HOSTS` setting in Django 1.5, that will use the domains defined in the Site model - Stephen McDonald
- Use the string name for user relationships in models since trying to import a custom user model falls apart - Stephen McDonald
- Remove upgrade flag from installation instructions in case people don't know how pip works - Stephen McDonald
- Drop Python 2.5 tests in travis since Django 1.5 doesn't support it and we'll be dropping it soon - Stephen McDonald

Version 1.4.2 (Feb 23, 2013)

- Added ratings to comments, with new settings `COMMENTS_USE_RATINGS` for toggle ratings form in comments, and `RATINGS_RANGE` for defining valid ratings, replacing the old min/max settings. Also added `_sum` field injections for models with rating fields, and `rating_date` field on ratings, for use with time scaled scores - Stephen McDonald

- Ensure emails are lowercased for gravatar hashes - Stephen McDonald
- Fix page tree admin template when reversion is used - Stephen McDonald
- Enhanced args to gravatar URLs - Stephen McDonald

Version 1.4.1 (Feb 19, 2013)

- Remove unnecessary permission from `live_settings` module. Closes #568 - Stephen McDonald
- Test slug after setting parent of an unsaved page with autogenerated slug - wrwrwr
- Bum `filebroswer_safe` to 0.2.16 - Stephen McDonald
- Prefix `BLOG_USE_FEATURED_IMAGE` in blog templates with `settings.`, otherwise it always evalutes to False - Josh Cartmell

Version 1.4.0 (Feb 17, 2013)

- Added a `has_home` variable to templates for the `page_menu` template tag, which indicates whether a homepage object exists, and can be used for checking whether a hard-coded homepage link should exist in the menu template - Stephen McDonald
- Update the default twitter query since it's been flooded by movie tweets - Stephen McDonald
- Add a deprecation layer for settings in templates, and deprecate the `PAGES_MENU_SHOW_ALL` setting since it's too specific for a setting and can be implemented in one line in a template - Stephen McDonald
- Added an example to the page menu docs of rendering a tree representing the current section of a site being viewed - Stephen McDonald
- Don't need to uncheck `in_menus` for an editable homepage anymore, so remove the comment describing that - Stephen McDonald
- Correctly handle file uploads in profile forms - Stephen McDonald
- Alpha-sort options for the search form - Stephen McDonald
- Remove Nimbis Services link for now. We haven't deployed our Mezzanine-based Nimbis Services site. into production yet (the old link was to a testing site that is only. used internally). We'll add this back in once we go live with our Mezzanine site - Lorin Hochstein
- Also check `BLOG_USE_FEATURED_IMAGE` in templates when displaying blog post's featured image - Stephen McDonald
- Added a `sort_by` template filter for general use - Stephen McDonald
- Removed `Slugged.Meta.ordering` since it'll generally always be nuked by Meta on a subclass - and added correct ordering to BlogCategory - Stephen McDonald
- Move `clean_content` to new DisplayableAdminForm - Alex Hill
- Fix parent of Team and History pages in fixtures. Assign the Team and History pages to the About page (id 2) instead of the Blog page (id 1) in Page fixtures - Alex Hill
- Fix generating descriptions when saving page instances directly, as their content type subclass fields weren't available for the description - Stephen McDonald
- Allow for no content model in `Page.description_from_content` - Stephen McDonald
- Fixed duplicate home IDs in menu templates and add some missing IDs - Stephen McDonald

- Check `has_home` to avoid duplicates. Updated `footer_tree.html` to behave the same as the other menu templates, checking `has_home` so that a page that is also the home doesn't end up in the menus twice - joshcartme
- Strip language prefix from request path, before trying to match it against pages slugs - wrwrwr
- Drupal blog importer for mezzanine blog - #issue 527 - Bryden Frizzell
- Fixed `import_posterous` for module requests `v1.0.1` and above. - issue #528 - Skooch
- Restore permission check for editable JS/CSS - Stephen McDonald
- Added handling for model field defaults in dynamic inlines. Closes #526 - Stephen McDonald
- Precedence of conflicting page processor context. The order of execution of page processors was reversed in #315 so that custom page processors returning an HttpResponse would bypass the default processors. That had the side-effect of making context variables in default processors overwrite those in custom processors, which isn't very intuitive. This change restores the original behaviour of context variables, while retaining the reversed execution order - Alex Hill
- Added a welcome message and quick links for getting started, for new developers, to the default homepage template - Stephen McDonald
- Fixed conditional context updates in page processors for Python < 2.7 - Stephen McDonald
- Fix handling of non-alpha search terms in SearchableQuerySet - Stephen McDonald
- Fixed support for automatically adding custom ManyToMany fields in PageAdmin. Closes #534 - Stephen McDonald
- Improved some of the messages shown through installation (createdb/syncdb signals) - Stephen McDonald
- Clarify requirements for `search_fields` in the search api docs - Stephen McDonald
- Hide the help text for the slug field for Link pages in the admin - Stephen McDonald
- Fix JS/CSS file names in base mobile template. Closes #537 - Stephen McDonald
- use `AUTH_USER_MODEL` if available - Ben Wilson
- Fix Manager MRO issue where `search_fields` param threw errors - David Novakovic
- Test for SearchableManager in DisplayableManager - David Novakovic
- Hopefully fix MRO regression - David Novakovic
- Fix MRO issues and avoid regression at the same time - David Novakovic
- Protect sequences provided or generated for the default value of `MenusField` from being forced to unicode (as for example u'[1, 2, 3]'). Django forces fields defaults to unicode unless they're callable (see `Field.get_default`). This is done to prevent problems that could arise from setting the same mutable object as a default for many fields (see Django ticket #18478) - wrwrwr
- add fa and `fa_IR` locales - Mahdi Bornazadeh
- Clean up use of `AUTH_USER_MODEL` - Stephen McDonald
- Database-prepare tuples in the same way lists are handled in `MultiChoiceField` - wrwrwr
- Allow pages to be dragged out of a subtree to the root level in page admin - wrwrwr
- Check that setting a new page parent won't cause a cycle in the parent-child graph. Such cycles lead to an infinite loop in `Page.save` (`e.g.` python process consuming all resources) - wrwrwr
- Altered git `repo_url` checks to allow ssh hosted git repositories - Travis Nickles
- Fixed indentation issue and PEP-8 issue with fabfile mods - Travis Nickles

- Don't try to create any pages if the models it uses aren't installed - Gavin Wahl
- Support for Django 1.5 custom user models. Uses `get_user_model` for every reference to User, and provides a default. implementation of `get_user_model` for Django <= 1.4 - Gavin Wahl
- Clean up hg/git checks in fabfile - Stephen McDonald
- Move RSS url parsing code from the drupal importer into the main RSS blog importer, and remove the drupal importer since it isn't specific to drupal - Stephen McDonald
- Fix import error message in rss importer - Stephen McDonald
- Don't use Bootstrap's navbar-inverse class by default, for better theme compatibility. Closes #551 - Stephen McDonald
- Fix some missing imports and settings import errors for the new user model hooks - Stephen McDonald
- Added possibility to set custom menu titles for models in `ADMIN_MENU_ORDER`, using the same notation as for views `(e.g. (_("News"), "blog.BlogPost"))` - wrwrwr
- Avoid fixing parent slug in `Page.set_parent` if the page had no slug to start with - wrwrwr
- Use `current_page` instead of request for `is_current`. Since we already have the 'current page' object, we can compare it. against ourself to find if we are current - Gavin Wahl
- Replace the rating form with a message after user casts a vote - wrwrwr
- Use `content_model` not the base Page in PageAdmin. When calling methods on a page, they should be called on the subclass,. not the base Page. This allows page types to override them - Gavin Wahl
- The usage of reversion can now be disabled for DisplayableAdmin - uli
- Update admin menu docs to mention labels for regaulr models - Stephen McDonald
- Change new reversion setting to be opt-in instead of opt-out - Stephen McDonald
- Moved the `MEDIA_LIBRARY_PER_SITE` setting from `filebrowser_safe` into Mezzanine so it's documented. It allows per-site filebrowser root directories - Stephen McDonald

Version 1.3.0 (Dec 26, 2012)

- added dob field to list of form fields - mmuk2
- Update url templatetags for Django 1.5. See `https://docs.djangoproject.com/en/1.4/releases/1.3/#chang` "{% load url from future %}" is omitted in favour of a global import in. `boot/__init__.py` - Alex Hill
- Bring templates in line with latest master - Alex Hill
- Move forward compatibility code to `utils/conf.py` - Alex Hill
- Assume development server if command is "harvest". Lettuce uses the "harvest" command to run a development server. See `http://lettuce.it/recipes/django-lxml.html#lettuce-run-the-tests`. Note that if this isn't set, then media will not be served correctly. when testing with lettuce - Lorin Hochstein
- Bump versions: `filebrowser_safe` >= 0.2.12, `grappelli_safe` >= 0.2.10 - Stephen McDonald
- Use non-minified `jquery.tools` and `jquery.ba-resize` - Per Andersson
- Render admin "add" link if no change permission. Handle the case where a non-superuser staff member has "add". permission but not "change" permission - Lorin Hochstein
- Escape backticks in python task in fabfile. Closes #396 - Stephen McDonald
- Ensure last output line is used to determine remote `STATIC_URL` in fabfile, since warnings may occur in output - Stephen McDonald

- add `related_posts` for blog - Dmitry Falk
- fix `related_posts` in template - Dmitry Falk
- Allow users to sign up with capital letters in their username - David Novakovic
- Update `mezzanine/core/admin.py`. Add some stuff to OwnableAdmin to make its use more obvious to new users - David Novakovic
- Filter urls that use https - Eduardo Rivas
- Added mezzanine-polls to third party apps - Stephen McDonald
- Update `mezzanine/accounts/__init__.py`. Display more informative error if this exception is thrown. This exception handler can hide informative errors about model unrelated model declaration. - David Novakovic
- Update `mezzanine/accounts/__init__.py`. Even better checks for the profile model string - David Novakovic
- Fix unfiltered RSS feeds for Django 1.3 - Stephen McDonald
- Use tag slugs for tag RSS feeds - Stephen McDonald
- Fix unicode handling for slugs in Django 1.5 - Stephen McDonald
- Fix urls in mobile search include for Django 1.5 - Stephen McDonald
- Fix mobile tests for Django 1.5 - Stephen McDonald
- Handle invalid images in thumbnail tag. Closes #410 - Stephen McDonald
- Use Page URLs without trailing slash when `settings.APPEND_SLASH` is False - Kenneth Falck
- Full support for `APPEND_SLASH` is False - Stephen McDonald
- Removing initial content from createdb when –nodata parameter is present - Sean Voss
- Added `TWITTER_STRIP_HIGH_MULTIBYTE` setting to strip mb3/mb4 characters in Tweets (mainly Emoji), which cause problems with MySQL UTF-8 collation - Kenneth Falck
- Added the setting `SSL_FORCED_PREFIXES_ONLY`, which defaults to True and controls whether URLs not matched by `SSL_FORCE_URL_PREFIXES` are redirected back to HTTP if accessed over HTTPS - Stephen McDonald
- Added the `COMMENT_FILTER` setting for controlling how comments are rendered. Works the same as the `RICHTEXT_FILTER` setting. Closes #416 - Stephen McDonald
- Added `has_children_in_menu` and `num_children_in_menu` attributes to page objects in the `page_menu` template tag, for determining valid children in the context of a menu and the `in_emnus` field. Closes #413 - Stephen McDonald
- Added automated hg tagging for versions in changelog generation. Closes #259 - Stephen McDonald
- Fixed misspelling of argument in `send_verification_mail` - Zean Tsoi
- Framework to allow `EXTRA_FORM_FIELDS` - Sean Voss
- Allow subclasses to define their own ProfileFieldsForm - David Novakovic
- patches to be jython compatible - Donneker
- Fixes #427: Disqus comment counts are now pulled in on the blog post detail page, if available - cato
- Fix incorrect status on quick-blog form. Closes #429 - Stephen McDonald
- Make form fixtures optional and remove old fixtures - Stephen McDonald

- Use createdb –nodata in fabfile - Stephen McDonald
- Use actual keyword instances in blog listing. Closes #431 - Stephen McDonald
- Put block tags into all blog templates so they can be overridden. Closes #443. This resulted in the splitting of the editable field for the title and. publication date in the list page into two editable fields, so they. could be two separate blocks. I notice that the blog detail page. doesn't have an editable field for the publish date at all, which I. shall address separately. block tags are namespaced by `blog_post_detail_` and `blog_post_list_`. respectively, and inside the list page, the blocks related to an. individual post are namespaced with `blog_post_list_post_` - Penny Leach
- Made publication date an editable field in the blog post detail template - Penny Leach
- Remove selection disabling in `page_tree.js` - causing issues with latest Firefox - Stephen McDonald
- Added some missing calls to `richtext_filter`. Closes #438 - Stephen McDonald
- Correctly handle empty password in login form. Closes #439 - Stephen McDonald
- Move error templates into custom paths so that Django's tests can trigger errors using its own error templates, since Mezzanine's urlpatterns aren't used which its error templates depend on - Stephen McDonald
- Add some extra comments and validation for the new `FORMS_EXTRA_FIELDS` setting - Stephen McDonald
- Allow LoginForm to be inherited and extended - Renyi Khor
- Slugged model now uses `self.title` to generate slug. Fixes #445 - Andrey Shipilov
- Update `mezzanine/blog/models.py`. wrong keyword argument passed to `blog_post_list_category` in `get_absolute_url` for the BlogCategory model. This results in an empty url when using Link to my Category. The problem was that the `blog/urls.py` uses 'category' as the keyword and the `get_absolute_url` used 'slug' as the keyword. I changed it within `get_absolute_url` because I guess changing it within `blog/urls.py` may break backwards compatibility - Andre Graf
- Port gallery expose to updated jquerytools version. On overlay load, expose the `.image-overlay`. Fixes bug where every other image was not exposed due to timing issue. when #exposeMask fades out when already switched to next image - Per Andersson
- Use local copies of instead of cdn. * html5shiv. * jquery mobile - Per Andersson
- Move `html5shiv.js` outside of Mezzanine's js directory, since it's not required by Mezzanine itself (eg it's project specific and can be removed per project) - Stephen McDonald
- Update `blog_recent_posts` to allow an optional slug. If the slug is specified returned blog posts will be restricted to being in the category matching the slug. If the slug does not match a category, posts will be returned as normal - joshcartme
- Added support for keyword args in the `as_tag` template tag wrapper - Stephen McDonald
- Fix for issue #450: `home_slug` with prefix - uli
- Fix bad semicolon in `gallery.js` - Stephen McDonald
- Use `PROJECT_NAME` fabric setting as `CACHE_MIDDLEWARE_KEY_PREFIX` in `live_settings.py` - Stephen McDonald
- Update twitter bootstrap to `v2.2.1` - Ivan Teoh
- Inverse the top navbar from white to black - Ivan Teoh
- Superusers should be able to select any site - Josh Cartmell
- Disable front end editing for users who don't have access to a site - Josh Cartmell
- Include AdminProfileInline so that it is not lost if the user enables Mezzanine accounts - Josh Cartmell

- Check if the user `is_staff` first to avoid unecessarily reversing admin:index on every request - Josh Cartmell
- Only load and display inline editor if the user has access to the current site's admin - Josh Cartmell
- Only check if a user has access to the current site in the middleware. Save the result on `request.user` and use this elsewhere - Josh Cartmell
- Added the setting `OWNABLE_MODELS_ALL_EDITABLE` which allows a sequence of `app_label.model_name` models to be defined, that are Ownable subclasses which won't have their change-list admin views filtered by user - Stephen McDonald
- Updated signal to only automatically create admin profiles for staff and not break the User add view if a site is selected - Josh Cartmell
- Fix for issue #470: Right subclass instance in BaseGenericRelation - Thomas Jetzinger
- Add homepage url to `sitemap.xml` - Stephen McDonald
- Add handling for multi-tenancy in `sitemap.xml` - Stephen McDonald
- Check for published objects in `Orderable.next/previous` and allow kwargs to be used - Stephen McDonald
- Fixed margins on user-panel buttons - Stephen McDonald
- Added Displayable methods `get_next/previous_by_publish_date`, and used in blog post templates for next/previous blog posts - Stephen McDonald
- More accurate template block name for blog post prev/next links - Stephen McDonald
- Fix showstopper on first comment due to url being clobbered and never reset - Grant Warren-Robertson
- No need to log user out for invalid admin - Stephen McDonald
- Check for login form instead of user perms when choosing which js to load in admin's base template - Stephen McDonald
- Still log user out for invalid admin access - Stephen McDonald
- add environment setting to `supervisor.conf` to ensure locale is set correctly for gunicorn subprocesses - Doug Evenhouse
- modify environment setting to inject locale specified in FABRIC setting - Doug Evenhouse
- Allows regular link Cmd+Click behaviour on OS X. - Regular click behaviour still applies. - Tested to work in OS X Chrome 24 beta and Firefox 16 beta. - TODO: test on other platforms - Matt Stevenson
- Document how to run unit tests - Lorin Hochstein
- Don't run view functions from page middleware when no page can be found, just pass through. Closes #476 - Stephen McDonald
- Update jquery-ui to full 1.9.1, and include smoothness theme. This allows other apps to use a single version of jquery-ui that is. known to be compatible with Mezzanine's version of jquery. This is `jquery-ui-1.9.1.all`, so all widgets are available and no. additional jquery code is needed. Also, the full smoothness theme. is included. Third-party apps may include other themes - Ahmad Khayyat
- Added Django < 1.5's adminmedia tag lib for 1.5 compatibility - Stephen McDonald
- Clean up dev server check - Stephen McDonald
- Allow category/tag titles to be used in `blog_post_recent` tag - Stephen McDonald
- Bump grappelli safe version to 0.2.11 - Stephen McDonald
- Bump filebrowser safe version to 0.2.13 - Stephen McDonald

- Added the setting `UPLOAD_TO_HANDLERS` for configuring the `upload_to` arg per file field. Closes #480 - Stephen McDonald
- Added missing word in Blogger import notes - Matt Stevenson
- Change feedparser URL to authoritative fork. - The original author's website(s) returns HTTP 410. - Refer to: `http://en.wikipedia.org/wiki/Mark_Pilgrim_(software_developer)` - Matt Stevenson
- Resolves html entity output re: #482 - Matt Stevenson
- Generate better meta descriptions from markdown content. By using the newline character as the first pattern in the generation of the meta description, markdown content (which normally lacks closing *</p>* tags) is processed correctly - Eduardo Rivas
- Parse content with `rich_text` filter - Eduardo Rivas
- Moved import inside method - Eduardo Rivas
- Added optional parameters to search view - Eduardo Rivas
- Request filters: specified using `REQUEST_FILTERS` in `settings.py` - Chris Ravenscroft
- Added default setting for `REQUEST_FILTERS` - Chris Ravenscroft
- Works better with the proper values in `settings.py` - Chris F Ravenscroft
- Escape miscellaneous percent symbols in deployment templates; fixes #494 - Olivier Harris
- spam filter code moved back to `views.py`; using mezzanine's module import mechanism - Chris Ravenscroft
- Added newline at the end of `search_form.html` - Eduardo Rivas
- JavaScript localization added for `mezzanine.forms` application - Oleg Churkin
- Correct varible name in single model search - Eduardo Rivas
- Fix site perms template error in admin logout - Stephen McDonald
- Update notes in the `project_template's urls.py` describing how the homepage object should not be assigned to any menu templates - Stephen McDonald
- Add new field `Displayable.in_sitemap` which appears in the meta data section of each admin form, and controls whether the object appears in `sitemap.xml`. Closes #499 - Stephen McDonald
- Added {% `search_form` %} section to the docs - Eduardo Rivas
- Update `mezzanine/pages/admin.py`. Remove a blank line so tests will pass - Kenneth Love
- Don't assume {form, gallery} apps are installed. Importing these in `core.management.__init__` causes problems when they. aren't installed. Instead, import them in the function where they're. used - Gavin Wahl
- Adds the ability to move a page under a page with no children. I switched to using the jQuery nestedSortable plugin instead of the. sortable plugin provided by jQuery UI, because Pages actually being in a. tree structure, they need a tree editor. This commit temporarily breaks. some functionality such as remembering which pages were open and closed - Rocky Meza
- fixed pagetree hiding of subpages - Rocky Meza
- Provide Mezzanine's settings object to the `COMPRESS_OFFLINE_CONTEXT` setting for django-compressor. Closes #505 - Stephen McDonald
- Fix the bugs that we had with nestedSortable - Gavin Wahl
- only `.nestedSortable()` the first ol - Gavin Wahl

- Fix front-end editing links for elements not positioned relative to the document. Use visibility hidden and jquery offset function to ensure edit links are always positioned relative to the document and not relative to a positioned ancestor - Jonathan Potter
- Clean up the new page sorting view - Stephen McDonald
- Remove old hack for initial page tree click bug that no longer exists. Closes #509 - Stephen McDonald
- Fix null handling in page sorting view - Stephen McDonald
- Specify widget for keywords field so it can be overridden properly. Closes #421 - Stephen McDonald
- Bug fix for wrong argument ordering for ssl cert handling in `fabfile.py` - David Hess
- Remove some commented out editable settings from the project template's `settings.py` module, since defining these at the Python level can be confusing once the settings form in the admin is updated. Also made a note of this scenario in the settings docs. Closes #515 - Stephen McDonald
- Add ssl port to `nginx.conf`. Closes #514 - Stephen McDonald
- Bump filebrowser-safe version to 0.2.14 - Stephen McDonald
- Don't run redirects tests for Django 1.5 - Stephen McDonald
- More commit log filtering for changelog - Stephen McDonald

Version 1.2.4 (Sep 03, 2012)

- Added `mezzanine.utils.urls.home_slug` which will return the `slug` arg of the ``home` urlpattern, when a urlpattern is defined for an edtiable homepage. This ensures that we don't hard-code the URL for the homepage anywhere, and allows the editable homepage to work correctly when a `SITE_PREFIX` setting is defined - Stephen McDonald
- Added autofocus to first field of the form - Renyi Khor
- Added Html5Mixin to PasswordResetForm - Renyi Khor
- Add initial support for importing blog posts from posterous - David Novakovic
- Import comments for each post - David Novakovic
- Importer docs and small doco fix in code - David Novakovic
- We only need the hostname if you have more than one posterous blog - David Novakovic
- Host is optional if you have one blog - David Novakovic
- Remove requests import from global scope - David Novakovic
- Make the `page.in_menus` check a bit more robust in the `page_menu` template tag, in case it doesn't actually have a value, which may have occured if migrations weren't run when the `in_menus` field was added - Stephen McDonald
- Allow non-page views to specify their own `editable_obj` context variable, which is then used to determine the url for the admin link in the editable toolbar, falling back to the current page object. Allows for things like blog posts and Cartridge products to contain a direct admin link from the ditable toolbar - Stephen McDonald
- Remove unused `grappelli_safe` urlpatterns - Stephen McDonald
- Bump `grappelli_safe` version to 0.2.9 - Stephen McDonald
- Added accessor methods for blog post keywords and categories, so that when we use `prefetch_related` with Django >= 1.4 we don't need to iterate through every blog post to set up keywords and categories. Closes #383 - Stephen McDonald

- Use the named home url for the View site link in the admin header. Closes #389 - Stephen McDonald
- Ensure consistent path separators in overextends template tag on Windows. Closes #386 - Stephen McDonald

Version 1.2.3 (Aug 22, 2012)

- Only hide delete button in the submit row for SingletonAdmin. Closes #376 - Stephen McDonald
- Correctly handle invalid form fields when save is clicked in SingletonAdmin. Closes #375 - Stephen McDonald
- Added Ken Bolton's quote to docs homepage - mezz is django - Stephen McDonald
- Fix kwargs usage to work with other auth backends - David Novakovic
- Bump filebrowser version for security fix - Stephen McDonald

Version 1.2.2 (Aug 15, 2012)

- Update page menu handling in blog importer - Stephen McDonald
- Fix missing import in blog importer - Stephen McDonald
- Ensure `extra_context` in SingletonAdmin is always a keyword arg. Closes #370 - Stephen McDonald
- Clean up deploy doc - kevinlondon
- Initial layout for filtering RSS feeds by tag/category - Stephen McDonald
- Final bits for author/tag/category rss feeds in the blog app - Stephen McDonald
- Fixed auth for password reset - Stephen McDonald

Version 1.2.1 (Aug 11, 2012)

- Bump min Django version to 1.3.3 - Stephen McDonald
- Fix dict handling in changelog builder (not actually used) - Stephen McDonald
- Don't rebuild host var in `static_proxy`. Closes #361 - Stephen McDonald
- Fix bug in `Page.get_ascendants()` - pass a map to. `PageManager.with_ascendants_for_slug` instead of a tuple - Alex Hill
- Added more tests for `Page.get_ascendants()` - Alex Hill
- Allow unicode cache keys - Stephen McDonald
- Add `_order` to `Page.META.ordering` - Ken Bolton
- Bump `grappelli_safe` version to 0.2.8 - Stephen McDonald
- Added a check in `footer_scripts` to only include the analytics tracking code if user is not part of the staff team - Pedro Araújo

Version 1.2.0 (Aug 05, 2012)

- Redirect to next param or home on signup with pending account verification. Closes #289 - Stephen McDonald
- Prevent certain exceptions from swallowed by the cache middleware - Stephen McDonald

- Removed `in_navigation` and `in_footer` fields on the Page model, and replaced them with the `in_menus` field, which stored a list of IDs specifying which menu templates the page should appear in. Menu IDs are mapped to templates with the new `PAGE_MENU_TEMPLATES` setting - Stephen McDonald
- Template tag changes for the new `page.in_menus` field - Stephen McDonald
- Added mezzanine-twittertopic to third-party apps - Stephen McDonald
- Update fixtures for new `Page.in_menus` field - Stephen McDonald
- Move the page permissions section of the docs to underneath more important topics - Stephen McDonald
- Added page menu docs - Stephen McDonald
- Ensure unique slugs even when slug is provided. Closes #290 - Stephen McDonald
- Add a comment to the default urlconf about changing the admin urlpattern - Stephen McDonald
- Don't allow pages to be added as children to a homepage page. Closes #286 - Stephen McDonald
- Added more notes around the new `SITE_PREFIX` setting, and refactored the code a bit - Stephen McDonald
- Remove old page admin code for forcing order/slug to be set - Stephen McDonald
- Only set `COMMENTS_APP` if not defined. Closes #294 - Stephen McDonald
- Allow internal `PAGES_SLUG` setting to be configurable - Stephen McDonald
- register ThreadedComment admin for `mezzanine.generic` `COMMENTS_APP` only - Dmitry Falk
- Fix for progressive jpgs in thumbnail template tag. Closes #268. Closes #295 - Stephen McDonald
- Don't assume `COMMENTS_APP` is set - Stephen McDonald
- add a block `left_panel` in `base.html` to make it easier to over-ride / over-extend - Sanjay B
- Ensure urls are only added once to the list of items. It might happen that pages are listed multiple times since for. instance a RichTextPage is also a Page and both are subclasses. of Displayable - Enrico Tröger
- Redirect the /account/ URL to the profile update form, and the /users/ URL to the logged in user's profile. Closes #291 - Stephen McDonald
- Clean up sitemap URL handling - Stephen McDonald
- Use `publish_date` for BlogPosts in `/sitemap.xml` - Enrico Tröger
- `FORMS_USE_HTML5` is a core setting - Stephen McDonald
- Allow page objects with removed apps to still render - Stephen McDonald
- Ensure mezzanine's apps have their settings loaded before any others - Stephen McDonald
- fix utils if `mezzanine.accounts` not installed - Dmitry Falk
- Fix reference to richtext filter settings defaults which are now in `mezzanine.core` - Stephen McDonald
- Fix serialization of `Page.in_menus` fields for dumpdata command. Closes #303 - Stephen McDonald
- Fix initial tuple for ignorable nexts in `mezzanine.utils.login_redirect` - Stephen McDonald
- Make gunicorn names in supervisor project specific. Closes #285 - Stephen McDonald
- Added i18n cache key suffix - Renyi Khor
- Fix edge case in url templatetag causing ViewDoesNotExist error. It happened when `ACCOUNTS_PROFILE_VIEWS_ENABLED` was set to False. and profile app called `profile` (same as url name) was added. to `INSTALLED_APPS` - Michał Oleniec

- Fix TypeError on ProfileFieldsForm save. Passing `cleaned_data` from ProfileForm into ProfileFieldsForm. caused doubled validation which in case of ForeignKey. field tried to get instance by field value which was instance already. (excepting int from `request.POST`) - Michał Oleniec
- Add customizable profile form. - new setting `ACCOUNT_PROFILE_FORM`. - add `get_profile_form` help method. - add generic form getter into views. - update `mezzanine.account.templatetags` - Michał Oleniec
- Make `editable.js` work with `JQuery.noConflict()` - Adam Brenecki
- changes to detect the appropriate page when making the homepage part of the page tree. Old code did not detect the slug appropriatley forcing you to make the slug / in the admin area - James Page
- Make some template tags more robust (`keywords_for` and editable) by failing silently when given an empty variable, as the case may be in the blog templates when no blog page object exists, so we don't need to check for this case in the templates themselves - Stephen McDonald
- In the overextends template tag, only remove template paths from the list of available paths when the first call to `find_template` is made in each call to `get_parent`, otherwise every second parent template found is skipped - Stephen McDonald
- Bump `filebrowser_safe` to 0.2.9 - Stephen McDonald
- Switch page processor execution order so custom slug processors are executed before model processors - Hakan Bakkalbasi
- Change `@processor_for` registration logic so most recently registered page processors are run first - Hakan Bakkalbasi
- For custom hompegae slug lookup in PageMiddleware, fix missing import and only call resolve once - Stephen McDonald
- Hash cache keys when talking directly to the cache API, to avoid keys longer than the backend supports (eg memcache limit is 255) - Stephen McDonald
- overextends tag path fix for uwsgi - Stephen McDonald
- Added new optional field `MetaData._meta_title` for overriding HTML title tag value, accessible via `MetaData.meta_title`, which will return the string version of an instance of `_meta_title` is not provided - Stephen McDonald
- Add parent hierarchy to page template rendering - Ken Bolton
- Added rollback command for deploys - Stephen McDonald
- Only pip install requirements if the requirements file has changed - Stephen McDonald
- Use `file.url` instead of `file.path` to auto-generate descriptions for gallery image, as remote storage backends such as S3BotoStorage do not support the `file.path` method - Hakan Bakkalbasi
- Added Django's tz context processor to `project_template.settings`. Closes #319 - Stephen McDonald
- Move settings specific to the pages app into their own defaults module, and add `PAGE_MENU_TEMPLATES` commented out in `project_template's settings.py` along with other common settings - Stephen McDonald
- Added missing defaults module for pages - Stephen McDonald
- Don't use the timezone context processor on Django 1.3 - Stephen McDonald
- Update docs for page hierarchy. Fix page template hierarchy issues around `content_model` - Ken Bolton
- Still call `contribute_to_class` for dynamic fields in `mezzanine.generic` even when frozen by south. Closes #321 - Stephen McDonald

- Improve page template hierarchy documentation - Ken Bolton
- In fabfile, always update requirements if any are unpinned - Stephen McDonald
- Fix indentation. <leader>-fef does not understand rst! - Ken Bolton
- Remove global from `get_parents` - Ken Bolton
- Adds `get_ascendants()` to Page. This returns all pages along the path from the root of the Page tree to. this page. The value is pre-calculated in PageMiddleware - Alex Hill
- Add comments and remove an obsolete variable - Alex Hill
- Ensure editable integer settings always have a value. Closes #325 - Stephen McDonald
- Fix regression in `keywords_for` tag for class args. Closes #326 - Stephen McDonald
- Remove HTML filtering from tincymce setup since we're filtering server-side - Stephen McDonald
- Allow comments in HTML filtering - Stephen McDonald
- Change build IRC notifications to only occur if the build status changes - Stephen McDonald
- Upgrade `manage.py` to the new cli handler, and throw out some old dev code. Closes #330 - Stephen McDonald
- Further mimic Django's new project layout - Stephen McDonald
- Addedd a note to the deployment docs describing how alternative web servers and DBs can be used - Stephen McDonald
- Prevent docs build import errors when optional dependencies for the rss blog importer aren't installed - Stephen McDonald
- Add links in the depooyment docs to the web and database server sections in the Django docs - Stephen McDonald
- Save a query in `page.get_ascendants` - Stephen McDonald
- Update url for mezzanine-stackato - Stephen McDonald
- Fix tests asserting number of queries used - Stephen McDonald
- Move ascendant page lookup by slug, from PageMiddleware into a method on a new PageManager manager for the Page model, and use it as the first attempt at loading ascendants in `Page.get_ascendants`, before falling back to recursive queries in the case of a custom slug in the ascendants chain - Stephen McDonald
- Add tests for the new page ascendant lookup methods - Stephen McDonald
- Bump `filebrowser_safe` to 0.2.10 - Stephen McDonald
- Added some notes about parent template selection in the page view - Stephen McDonald
- Don't use `with_ascendants_for_slug` in `Page.get_ascendants` if a slug hasn't been created yet - Stephen McDonald
- Make relation check in signals for generic fields more robust - Stephen McDonald
- Different attempt at making relation check in signals for generic fields more robust - Stephen McDonald
- Allow static proxy URL to be configured - Stephen McDonald
- Also rename `static_proxy` URL default in case anyone else's web server alias is slightly off - Stephen McDonald
- Preserve slugs & URLs when pages are moved - Alex Hill
- Check overridden() when changing slug - Alex Hill

- In `set_parent()`, call save() before `get_slug()` - Alex Hill
- Make slug changes propagate to all descendant pages - Alex Hill
- use `ugettext_lazy` strings for settings form - Dmitry Falk
- Allow unicode twitter search queries - Stephen McDonald
- `static_proxy` only needed a rename, not to be congifurable - Stephen McDonald
- Fix menu test to work with lazy unicode settings - Stephen McDonald
- Added new, working, repo for mezzanine-openshift. Since the old one doesn't work anymore and is unmaintained - Isaac Bythewood
- Fix collision of all task with built-in all function - Lorin Hochstein
- Fixed duplicate posts, –noinput handling and entity decoding in base blog importer - Stephen McDonald
- Restore automatic redirects creation for the wordpress blog importer - Stephen McDonald
- Add day parts to the date urlpattern for blog posts - Stephen McDonald
- Fix bug in `Page.set_parent()` when no parent set - Alexander Hill
- Allow passing None to `Page.set_parent()` - Alexander Hill
- Add tests for `Page.get_slug()` and `Page.get_parent()` - Alexander Hill
- Swallow import exception when importing `settings.py` from fabric - Lorin Hochstein
- Don't mask import errors in the actual call to `set_dynamic_settings` - Stephen McDonald
- add ajax hook for generic rating - Dmitry Falk
- Fix references to the `recent_comments` template tag - Enrico Tröger
- Fix bug in slug handling when adding a new page - Alex Hill
- Fix #349 - regression in `set_page_permissions` - Alex Hill
- Madee the labels and help text for the email fields more descriptive - Stephen McDonald
- Added mezzanine-events to third-party apps list - Stephen McDonald
- Added FAQ to the docs covering the HTML filtering settings - Stephen McDonald
- Add a setting to send notification mails to MANAGERS when a new comment is posted - Enrico Tröger
- PageAdmin now respects excluded fields - Aaron Merriam
- Fix #348, RichTextFields in IE - Ken Bolton
- Change the rating ajax response to return the new rating - Stephen McDonald
- Added the bool setting `PAGES_PUBLISHED_INCLUDE_LOGIN_REQUIRED` which when set to False (default) will exclude pages with `login_required` set to True in `PageManager.published`. This affects the `page_menu` template tag which renders menus, and pages listed in search results - Stephen McDonald
- Added `get_next_by_order` and `get_previous_by_order` methods to the Orderable model - Stephen McDonald
- Allow `PAGES_PUBLISHED_INCLUDE_LOGIN_REQUIRED` handling to be overridden by callers to `PageManager.publsihed` for cases when they want to deal with `login_required` manually, such as in the case of PageMiddleware - Stephen McDonald
- Marked fabfile functions explicitly with task decorator, and added a custom docs generator for each task - Stephen McDonald

Version 1.1.4 (Jun 27, 2012)

- Add custom introspection rules that prevent duplicate field creation on dynamic fields during migration - Stephen McDonald
- Use filebrowser field's format attribute rather than extensions. Closes #287 - Stephen McDonald

Version 1.1.3 (Jun 26, 2012)

- Allow rel attributes in anchor tags - Stephen McDonald
- Don't cast to list in paginate - Stephen McDonald
- Remove redundant ampersands in pagination links - Stephen McDonald
- Update the configuration docs example to use author/blooks instead of gallery/images, and add the new options for registered settings, choices and append - Stephen McDonald
- Allow default twitter feed to be managed via admin settings - Stephen McDonald
- Raise NotImplementedError on Displayable subclasses that don't implement `get_absolute_url` - Stephen McDonald
- Add new setting `SITE_PREFIX` to configure a custom prefix. This is useful if Mezzanine doesn't run at the root of the domain - Enrico Tröger
- Add and use `utils.urls.get_page_slug_from_path()` to handle non-root configurations. For the pages app, we need to handle removing the `SITE_PREFIX` and `PAGES_SLUG` if. they are set - Enrico Tröger
- Set `is_current` on Pages when added to context in PageMiddleware - Alex Hill
- Permit disabling page processors at external apps' urls in the page tree - Alex Hill
- Refactored overextends template tag to not depend on template origins since they're not available with DEBUG off - Stephen McDonald
- Fix variable resolution for `as_tag` template tags - Stephen McDonald
- Added template tags for the various account forms - Stephen McDonald
- Bump `grappelli_safe` to 0.2.7 for admin column sorting fix - Stephen McDonald
- Clean up exact page matching for page processors - Stephen McDonald
- Updated jQuery Form Plugin - Renyi Khor
- Fix `_current_page` in middleware - Stephen McDonald
- Reorganised page middleware for fewer queries and readability - Alex Hill
- page middleware: use `request.path_info` - Dmitry Falk
- Correctly handle root URL - Alexander Hill
- Add check for `page_branch_in_footer`. Without this check, `footer.html` is rendered for every page in the tree,. returning an empty string - Alexander Hill
- Add perms to existing context page instead of overwriting it. Previously the template tag `set_page_permissions` would retrieve the. page's content model, set the perms attribute on it, and then replace. the page object in the context with the retrieved object. Setting perms. on the existing page object instead preseves attributes set by `set_helpers` - Alexander Hill
- Check `has_children` before calling `page_menu`. This saves a lot of template renders in wide page trees - Alexander Hill

- backport of django-forms-builder signals to `mezzanine.forms` - Brian Schott
- set mimetype to empty string in case path is not found - Brian Schott
- Handle no blog page existing for meta keywords in the blog list template - Stephen McDonald
- Fix path lookup for Python 2.5 - Stephen McDonald
- Handle FileBrowseField args in Django FileField fallback - Stephen McDonald
- Use image formats for image FileBrowse fields - Stephen McDonald
- Bump `filebrowser_safe` to 0.2.7 - Stephen McDonald
- Cleaned up blog import redirect creation - Zachary Gohr
- Bugfix: Account form validation errors on non-html5 browsers - Renyi Khor
- added in-navigation test to level 1 - Brian Schott
- fix migration without blog app - Dmitry Falk
- Ensure Mezzanine's auth backend is enabled if `mezzanine.accounts` is installed. Closes #281 - Stephen McDonald
- Eval settings choices when generating settings docs - Stephen McDonald

Version 1.1.2 (Jun 04, 2012)

- Fix slug handling in page middleware for homepage as page object - Stephen McDonald
- add some verbose names - Dmitry Falk

Version 1.1.1 (Jun 04, 2012)

- Don't assume rating field is named rating - Stephen McDonald
- Handle `PAGES_SLUG` in the page middleware - Stephen McDonald
- Make the creation of `PAGES_SLUG` not dependant on the position of the blog urlpatterns in urlpatterns created before the page urlpatterns - Stephen McDonald
- Fix quoting unicode thumbnail filenames - Stephen McDonald
- Move lookup of page subclasses into classmethod `Page.get_content_models`, and call `select_related` on all page subclasses in the `page_menu` template tag when used for the admin page tree, since we need to touch all the related content type instances to check page paermissions - Stephen McDonald
- Don't assume request is available in `page.set_menu_helpers` - Stephen McDonald
- Move cache-busting querystring into `mezzanine.utils.cache.add_cache_bypass` and apply it to comments and ratings redirects so that posted content appears immediately - Stephen McDonald

Version 1.1.0 (Jun 03, 2012)

- Added `MetaData.gen_description` bool field for controlling whether description fields are automatically populated via `MetaData.description_from_content` - Stephen McDonald
- Emit the `comment_was_posted` signal in the comments view - Stephen McDonald
- Correctly handle model field defaults in the quick blog post form - Stephen McDonald

- Added the setting `COMMENTS_ACCOUNT_REQUIRED`, which when True, will store an unauthenticated user's comment in the session and redirect to login/signup, and save their comment once they're authenticated - Stephen McDonald
- Use setting names as labels if they're missing - Stephen McDonald
- Wrap data access in migrations with checks against the `dry_run` arg - Stephen McDonald
- added missing fr `django.mo` for the conf app - Nicolas Perriault
- Only pre-populate name in the comment form with the user's username if it's not their email address, which it is by default - Stephen McDonald
- Always use the name from the comment form, rather than the user's username, since by default it's their email address - Stephen McDonald
- Use `comments.select_related(user)` when loading comments, since Django's Comment model will query for the user each time a comment is loaded - Stephen McDonald
- Added the setting `ACCOUNTS_VERIFICATION_REQUIRED` which when set to True, will create new accounts as inactive, and send the user an email with a verification link to activate their account - Stephen McDonald
- Remove invalid examples of gettext in settings module - Stephen McDonald
- Fixed slug-based template name loading for non-ascii slugs - Stephen McDonald
- Fix unencoded template names from slugs in blog also - Stephen McDonald
- Added the SLUGIFY which takes a dotted Python path to the slugify function to use when converting strings into slugs. Defaults to `mezzanine.utils.urls.slugify_unicode` which allows for non-ascii URLs - Stephen McDonald
- Use the text required for the help text for required fields in `mezzanine.forms` when no help text is entered - Stephen McDonald
- Add HTML5 features to the comments form - Stephen McDonald
- Fixed assignment of page permissions in the admin page tree - Stephen McDonald
- Hide the delete button for `mezzanine.core.admin.SingletonAdmin` - Stephen McDonald
- Added the view `mezzanine.core.static_proxy` which is used to serve TinyMCE plugin templates, and uploadify's SWF, as these break with cross-domain errors when `STATIC_URL` is an external host - Stephen McDonald
- Fix with statement in Python 2.5 - Stephen McDonald
- Bump grappelli and filebrowser versions - Stephen McDonald
- Moved all user account features into a new app `mezzanine.accounts` - Stephen McDonald
- Handle non-ascii filenames on non-utf8 filesystems. Convert filenames and warn when saving them, and raise exceptions if trying to access them and the filesystem encoding has changed. Closes #186 - Stephen McDonald
- Add new exceptions module - Stephen McDonald
- Added the decorator `mezzanine.pages.decorators.for_page`, which can be used for wrapping views that map to protected pages. The decorator adds the page instance to the template context, and handles login redirects if `page.login_required` is True. Applied to the blog views, and also added handling for `login_required` on the blog page in the blog feeds, which if True, stops the feeds from producing any blog posts or meta data - Stephen McDonald
- Don't disconnect the default site signal if we're not connecting our own one - Stephen McDonald

- Only try and modify template lists when they're available - not the case when the response is pulled from cache - Stephen McDonald
- Added the ifisinstalled template tag to replace the `is_installed` template filter, which properly handles include tags when the given app is not installed. Closes #181 - Stephen McDonald
- Allow pages without children to serve as targets for sortable - Aleksandr Vladimirskiy
- Fixed regression in admin login interface selector middleware. Closes #192 - Stephen McDonald
- Fixed ifinstalled template tag so that it removes all tokens, not just include tags. Closes #193 - Stephen McDonald
- Use `prefetch_related` in Django 1.4 for categories and keywords in the blog post list view. Closes #190 - Stephen McDonald
- Backout admin tree empty child fix for now as it doesn't work quite correctly - Stephen McDonald
- Fixed settings docs generator. Closes #189 - Stephen McDonald
- Refactoring of blog feed view. Returns a http 404 instead of http 500 when the feed does not exists - Thomas Wajs
- Clean up the blog feeds - Stephen McDonald
- Dev started in 2009 - Stephen McDonald
- Added fix for thumbnail generation which would previously not work for images which contained special characters in the file path and used url encoding - Kowaleski, Jason
- Added page import to wordpress - Alvin Mites
- restore utils/device for fork - Alvin Mites
- Added blog post content for the feed description - Thomas Wajs
- Allow the homepage to be login protected - Stephen McDonald
- Added handling for filebrowser's FileBrowseField directory arg. Closes #202 - Stephen McDonald
- Increased field lengths for `Displayable.title` from 100 to 500 and `Displayable.slug` from 100 to 2000 - Stephen McDonald
- Move ajax csrf setup into its own JS file that's loaded even when a popup interface is loaded. Closes #206 - Stephen McDonald
- Added the new app `mezzanine.accounts`, which handles user login, signup, update, password reset, profile, and integration with Django's user->profile features - Stephen McDonald
- Use ifinstalled for the accounts user panel - Stephen McDonald
- Added some commas to the username format error - Stephen McDonald
- Give the admin drop-down menu elements the same hover/click state as their anchors. Also closes #208 - Stephen McDonald
- Bump filebrowser-safe to 0.2.5 - Stephen McDonald
- Properly handle optional file upload fields in `mezzanine.forms` - Stephen McDonald
- clarify south usage in overview - Brian Schott
- Manually assign the parent to each page in the `page_menu` template tag, to prevent queries being triggered if they're accessed - Stephen McDonald
- Update notes about dependencies, and remove notes about setuptools - Stephen McDonald
- fixed docstring error in `mezzanine_tags.ifinstalled` - Brian Schott

- Added dynamic validation for content in DisplayableAdmin based on the value of status - Stephen McDonald
- Added handling for slug-based template when the homepage is a page object - Stephen McDonald
- Add handling for Django 1.4's timezone support - Stephen McDonald
- Remove DEBUG check from site/content signals, and prompt the user for the site domain in interactive mode, with local/live fallbacks for non-interactive mode - Stephen McDonald
- Added optional support for django-compressor - Stephen McDonald
- Fix `thumb_url` for root images on remote CDNs - Stephen McDonald
- Remove old fixes for Postgres and timezones - Stephen McDonald
- Allow initial dicts to be used for forms in `mezzanine.forms` - Stephen McDonald
- Update to new `gravatar_url` in comments admin - Stephen McDonald
- Use Django 1.4's `bulk_create` when creating field entries in `mezzanine.forms` - Stephen McDonald
- Added multi-tenancy support. A threadlocal object is used to store the current request, and a custom manager for site-related models is used, that checks for the current request and matches the host to a site domain. Current site can also be defined by a session var (for the admin), and an environment var (for management commands) - Stephen McDonald
- Made some visual enhancements to the settings admin, added support for settings with choices, and added the `RICHTEXT_FILTER_LEVEL` setting with choices for controlling the level of HTML filtering that occurs on the RichTextField - Stephen McDonald
- Proper timezone support for tweets - Stephen McDonald
- Update docs on multi-site to describe the new multi-tenancy approach - Stephen McDonald
- Use default `STATICFILES_FINDERS` setting when setting up compressor - Stephen McDonald
- Update travis config to test multiple Django versions - Stephen McDonald
- Fix Django install for travis - Stephen McDonald
- Added IRC notifications for travis builds - Stephen McDonald
- added remote url config script - Kent Hauser
- improved collecttemplates conflict messages - Kent Hauser
- remove `git.config.sh` for pull request - Kent Hauser
- Added `mezzanine.pages.middleware.PageMiddleware`, which handles loading the current page, running page processors, and checking `page.login_required`. Previously handled in `mezzanine.pages.views.page`, but move to middleware to allow pages to point to non-page urlpatterns, without any configuration via the now redundant `page_for` decorator. The page view remains for handling template selection and 404 handling - Stephen McDonald
- Added fabfile and configs for server setup and deploys - Stephen McDonald
- allow H1s in tinymce - lexual
- Handle homepage as page object in the new age middleware - Stephen McDonald
- Added a Link content type for creating external URLs in the page tree - Stephen McDonald
- Added the setting `ACCOUNTS_MIN_PASSWORD_LENGTH` for minimum password length for user accounts - Stephen McDonald
- Added the setting `ACCOUNTS_PROFILE_FORM_EXCLUDE_FIELDS` for excluding profile model fields from the profile form - Stephen McDonald

- Ensure min password length in accounts tests - Stephen McDonald
- Hides pagination if only one page - Renyi Khor
- Allow `auth.User` fields to be excluded from the profile form via the `ACCOUNTS_PROFILE_FORM_EXCLUDE_FIELDS` setting - Stephen McDonald
- Initial docs for the bundled fab deployments - Stephen McDonald
- fix i18n settings title in admin - Dmitry Falk
- Don't show excluded profile fields in profile view - Stephen McDonald
- Allow existing virtualenvs to be removed/replaced in fabfile - Stephen McDonald
- Added handling for settings with choices in settings doc generator - Stephen McDonald
- Added docs for `mezzanine.accounts` - Stephen McDonald
- Added optional quality arg to be passed to the thumbnail tag, and changed default from 100 to 95 as per PIL docs. Closes #221 - Stephen McDonald
- Ensure responses in PageMiddleware are valid responses for adding context to via page processors, eg not redirects - Stephen McDonald
- Added the {% overextends %} built-in template tag which allows templates to be both overridden and extended at the same time - Stephen McDonald
- In-line edit enhancements re-align on show/resize/expand - Van Nguyen
- Added body resize event for triggering realign of edit controls - Stephen McDonald
- added dropdown menu support - Brian Schott
- added default navlist sidebar - Brian Schott
- only activate current page - Brian Schott
- Fix original image links in gallery template - Stephen McDonald
- Refactored fabfile: - Move all templates into a config. - Move template upload and optional reload into deploy. - Added crontab handling - Stephen McDonald
- Add proc name to gunicorn conf - Stephen McDonald
- Clean up the new primary dropdown menu - Stephen McDonald
- Fixed non field errors in `fields_for` template tag - Stephen McDonald
- Merge navlist into tree menu - Stephen McDonald
- In fabfile, prompt to create project if it doesn't exist on deploy - Stephen McDonald
- Require hosts in fabfile - Stephen McDonald
- Ensure fabfile has hosts, and imports settings from the current path - Stephen McDonald
- Clean up ^M characters at end of lines using dos2unix and find: find . -type f -exec egrep -q $'r$' {} ; -exec dos2unix {} ; - Thomas Lockhart
- Fix missing tag - Pavel Ponomarev
- fix `get_absolute_url` for homepage - Dmitry Falk
- Allow superuser password to be defined in fabric settings, and create superuser if defined - Stephen McDonald
- Added the setting `ACCOUNTS_PROFILE_VIEWS_ENABLED` for explicitly enabling public profile pages, which defaults to False - Stephen McDonald

- Only validate fabric settings when fab is run - Stephen McDonald
- Shadow the admin password in fabfile - Stephen McDonald
- Add handling for the hotfix releases in the changelog builder - Stephen McDonald
- Allow large uploads in `nginx.conf` - Stephen McDonald
- Don't fail on fabfile import (for docs build) - Stephen McDonald
- Added owner/mode handling for templates in fabfile - Stephen McDonald
- Fix keyword queries in blog listing - Stephen McDonald
- Use standard page in mobile blog post listing - Stephen McDonald
- Add a cache-busting querystring to device switching - Stephen McDonald
- add some verbose names for blog - Dmitry Falk
- Remove deprecated clear attr from br tags. Closes #241 - Stephen McDonald
- Added some more notes around twitter cron jobs - Stephen McDonald
- Fixed initial values for entry instances on multi-value fields - Stephen McDonald
- Better locale error messages - Stephen McDonald
- Added Mezzanine's own cache system - combination of Django's cache middleware, two-phased render cache, and mint cache - Stephen McDonald
- Added `robots.txt/favicon.ico` handling in `nginx.conf` - Stephen McDonald
- Added docs for the new cache middleware - Stephen McDonald
- Clean up the deprecated middleware classes - Stephen McDonald
- Default `CACHE_MIDDLEWARE_SECONDS` to a minute in deployed settings - Stephen McDonald
- Add `SECURE_PROXY_SSL_HEADER` to deployed settings. Closes #246 - Stephen McDonald
- Fix var names in deploy configs - Stephen McDonald
- Cleaned up descriptive text - Ross Laird
- Added "timesince" to displayable - Renyi Khor
- Added thumbnail to blogpost admin - Renyi Khor
- Add SSL config to `nginx.conf` and self signed cert setup to fabfile - Stephen McDonald
- git pull -f in deploy - Stephen McDonald
- Added `mezzanine.utls.models.AdminThumbMixin` which provides a method for admin classes to refernce in their `list_display` that will render a thumbnail. Used for `BlogPost.featured_image` and `Product.image` in Cartridge - Stephen McDonald
- Revert cache changes to Twitter queries - since authenticated users bypass the cache, and the Twitter call will generate a lot of queries - Stephen McDonald
- Quote thumb names in thumbnail template tag - Stephen McDonald
- Use cache backend for sessions in deployed settings - Stephen McDonald
- Don't remove key/cert when blowing away a deployed instance in fabfile - Stephen McDonald
- Use the parent breadcrumb in blog templates, so as not to assume a single root blog page - Stephen McDonald
- Rewrite `Page.set_menu_helpers` to use the currently viewed page instead of the current URL - Stephen McDonald

- Ensure `Page.get_absolute_url` returns absolute URLs for Link page types - Stephen McDonald
- Allow overridden pages (eg the blog) to be deleted and have child pages added to - Stephen McDonald
- Recompile all `.mo` files - Closes #250. Closes #251 - Stephen McDonald
- Right-align drop-down menus when `.pull-right` is used - Stephen McDonald

Version 1.0.10 (Apr 28, 2012)

- Bump filebrowser-safe for security fix to 0.2.6 - Stephen McDonald

Version 1.0.9 (Apr 26, 2012)

- Add HTML sanitizing on RichTextField instances. Closes #211 - Stephen McDonald

Version 1.0.8 (Mar 24, 2012)

- Fixed `.navbar .container` responsive width - Stephen McDonald
- Added default blank favicon and replace Bootstrap's collapse JS with all Bootstrap JS - Stephen McDonald
- Added nav dividers in primary menu - Stephen McDonald
- Fixed leftover tag loading in form response emails - Stephen McDonald

Version 1.0.7 (Mar 24, 2012)

- Upgrade Bootstrap to 2.0.2 - Stephen McDonald

Version 1.0.6 (Mar 21, 2012)

- Fixed draft status for quick blog form in dashboard. Closes #172 - Stephen McDonald
- Format newlines in the quick blog form since the expected format is HTML - Stephen McDonald

Version 1.0.5 (Mar 19, 2012)

- Fixed admin navigation showing in inline filebrowser popups when called from TinyMCE - Stephen McDonald
- Bump `filebrowser_safe` to 0.2.3 - Stephen McDonald

Version 1.0.4 (Mar 19, 2012)

- No changes listed.

Version 1.0.3 (Mar 19, 2012)

- Don't restrict image width in default css since it's now responsive - Stephen McDonald
- Updated `templates_for_host` to insert default templates after the associated custom template, rather than putting all defaults at the end - Josh Cartmell
- Updated `templates_for_device` to insert default templates after the associated custom template, rather than putting all defaults after all custom templates - Josh Cartmell
- Disable nav in popups. Closes #152 - Stephen McDonald
- Refactored model graph building in docs - call management command natively, and handle all the error conditions - Stephen McDonald
- Internal refactoring of abstract models in `mezzanine.core`. Move `admin_link` from Displayable to Slugged, since it is more closely related to URLs. Move `description_from_content` from Slugged to MetaData, since it is more related to description on MetaData. Don't rely on title in `description_from_content`, just use string version of an instance, which is title anyway via Slugged - Stephen McDonald
- Added handling for having 'save' and 'save and continue' in SingletonAdmin - Stephen McDonald
- Make pillow an optional dependency, only used when PIL isn't installed - Stephen McDonald
- Added bootstrap's collapsible navbar, upgraded jQuery to 1.7, and added a setting `JQUERY_FILENAME` so that the jQuery file/version is stored in one place - Stephen McDonald
- Fix cyclic import in Django 1.4 - Stephen McDonald
- Don't abort on graph generation in docs build, since we can use the repo version of it - Stephen McDonald
- Pin exact versions in dependencies - Stephen McDonald
- Fix form export encoding - Stephen McDonald
- Updated database settings to use prefixed format. unprefixed format removed from django 1.4. Added `django.db.backends.` to. `settings.py` and `local_settings.py` templates - Patrick Taylor
- Clean up db settings and remove helpers from `mezzanine.utils.conf` - Stephen McDonald
- Added more info and examples of different homepage patterns in `project_template/urls.py` - Stephen McDonald
- Added FAQs section to docs - Stephen McDonald
- Skinned the docs to be in line with the Mezzanine project's homepage styling - Stephen McDonald
- Added storage API to thumbnail template tag, and zip upload for galleries - Stephen McDonald
- Fix use of with statement for Python 2.5 - Stephen McDonald
- Use django's conf at the module level in `mezzanine.core.fields`, so that fields can be loaded prior to `mezzanine.conf` being loaded - Stephen McDonald
- Exclude static dir from package - Stephen McDonald
- Added the collecttemplates management command, for copying all (or app specific) templates to a project - Stephen McDonald
- Added secure arg and default expiry seconds to `mezzanine.utils.views.set_cookie` - Stephen McDonald
- Added `mezzanine.utils.email.send_mail_template` for sending templated email, and integrated with `mezzanine.forms`. Closes #165 - Stephen McDonald

- Fixed weird double-click bug in admin page tree - Stephen McDonald
- Fixed regression in orderable inlines from upgrading to latest jQuery - Stephen McDonald
- Fixed regression in keywords field from upgrading to latest jQuery - Stephen McDonald
- Fixed signature change in Django 1.4's admin `change_view` - Stephen McDonald
- Fixed admin login redirect for non-login view URLs - Stephen McDonald
- Fixed removed `project_template` setup in mezzanine-project. Closes #167 - Stephen McDonald
- Use operating system separator - Chris Trengove

Version 1.0.2 (Mar 05, 2012)

- Update setup to exclude new dev db name - Stephen McDonald

Version 1.0.1 (Mar 05, 2012)

- Add a patch to the changelog generator for the versioning blunder - Stephen McDonald
- Added a new middleware which will serve templates from a theme, based upon the host accessing the site - Josh Cartmell
- Separated the logic a little more to make `host_theme_path` more reusable - Josh Cartmell
- Remove mention of `site_media` which no longer applies with staticfiles used - Stephen McDonald
- Avoid file-in-use exception when deleting (on Windows) - Chris Trengove
- Added quote by Antonio Rodriguez and one line bio for each of the quoters - Stephen McDonald
- Fix a couple of test failures on Windows - Chris Trengove

Version 1.0.0 (Mar 03, 2012)

- Fixed runserver arg parsing for grappelli media hosting. Closes #110 - Stephen McDonald
- Added a note to the docs about not subclassing RichTextPage - Stephen McDonald
- Raise a more meaningful error message when someone tries to subclass a custom content type, which isn't supported - Stephen McDonald
- Every model mixing Slugged in with a cyclical dependency fails with dumpdata in current Django (including a tree with a fix applied for Django ticket #14226). The natural key declared in Slugged is the culprit - derkaderka
- Bookmarks are removed from `grappelli_safe` - Stephen McDonald
- Fixed duplicate keyword handling regression and added support for automatically removing unused keywords. Closes #116 - Stephen McDonald
- Added patching of `django.contrib.admin.site` in `mezzanine.boot` to defer certains calls to unregister/register to work around some loading issues for custom model fields - Stephen McDonald
- Don't use form email field as from address if `FORMS_DISABLE_SEND_FROM_EMAIL_FIELD` setting is True - John Barham
- Register `FORMS_DISABLE_SEND_FROM_EMAIL_FIELD` in `mezzanine.conf` - Stephen McDonald
- Fixed migration forms/0003 failure for Postgres - Luke Plant

- Fixed dependencies of migrations, so that `'./manage.py` migrate' works even if starting from scratch - Luke Plant
- Added installation instructions for adding Mezzanine to an existing project - Luke Plant
- Added a generic RSS blog importer - Stephen McDonald
- Added a type attribute to fields in `mezzanine.forms.forms.FormForForm` for use in styling, and removed CSS class assignments - Stephen McDonald
- `Addedmezzanine.mobile` commented out to `INSTALLED_APPS` in `project_template.settings` - Stephen McDonald
- Fixed authentication check in base admin template - Stephen McDonald
- Ported default templates from 960.gs to Twitter Bootstrap - Stephen McDonald
- Merge paging links settings into a single `MAX_PAGING_LINKS` setting - Stephen McDonald
- Cleaned up settings ordering - Stephen McDonald
- Stub out empty comment forms in the context for the comments test - Stephen McDonald
- Don't show help text for form fields with errors assigned, and show all errors rather than just the first - Stephen McDonald
- Added docs for the RSS importer - Stephen McDonald
- Update the docs copyright date and fix some warnings - Stephen McDonald
- Fix template path for cartridge hook - Stephen McDonald
- Added Number and URL field types to `mezzanine.forms` - Stephen McDonald
- Unicode fixes for MS Excel in forms export - Stephen McDonald
- Added a work-around for performance issues with `jQuery.ui.sortable` and large page trees - Stephen McDonald
- Add pillow as a dependency for getting PIL install properly - Stephen McDonald
- Added handling in PageAdmin for picking up any extra fields defined by subclasses of Page, when the admin class being used doesn't implement any fieldsets - Stephen McDonald
- Added a wrapper field `mezzanine.core.fields.FileField` for filebrowser's FileBrowseField, falling back to Django's FileField if unavailable - Stephen McDonald
- Changed the filebrowser urlpattern to match the admin menu name - Stephen McDonald
- Changed thumbnailing to use a separate directory defined by the setting `THUMBNAILS_DIR_NAME` - Stephen McDonald
- Added additional URL structure. To better mimic wordpress and other blogs URL I added a. /year/month/slug url path - Josh
- Changing name of url pattern - Josh
- Added an image gallery app `mezzanine.galleries` - Stephen McDonald
- Give blog post with date urlpattern a unique name and correct regex - Stephen McDonald
- Added the setting `BLOG_URLS_USE_DATE` to control blog post url format - Stephen McDonald
- Added my site which has taken the fairly popular pixel theme from Wordpress and partially created it from the html5boilerplate. I'll be working on rounding it out even further - joejulian
- Fixed Joe Julian's site link - Stephen McDonald

- Device detection uses lowercase strings - Alvin Mites
- Added unique URLs for gallery photo overlays - Stephen McDonald
- Updated device checking based on conversation from Stephen McDonald - Alvin Mites
- Added a `num_children` attribute to page objects in page menus - Stephen McDonald
- Changed LICENSE from 3-clause to 2-clause BSD - Stephen McDonald
- Fixed unicode handling in gallery image description from name - Stephen McDonald
- Added gallery image tests - Stephen McDonald
- Added demo fixtures for galleries - Stephen McDonald
- Add Blog Featured Images. Added featured images for blogs as well as settings to turn the feature. on and off - Josh
- Migration file for Featured image and setting the field to null - Josh
- Updated `page_menu` and `tree.html` to avoid creating uls if no pages in the `page_branch` are `in_navigation` - Josh Cartmell
- Updated `page_menu` `page_branch_in_navigation` and `page_branch_in_footer` to be more concise. Updated `tree.html` and `footer_tree.html` not print out uls unless `page_branch_in_navigation` or `page_branch_in_footer` are set - Josh Cartmell
- Accidentally omitted if from tag - Josh Cartmell
- Updated `footer.html` to avoid unecessary uls - Josh Cartmell
- Rolling back as the previous change to `footer.html` did not work with 3rd level menus - Josh Cartmell
- Updated `footer.html` again to avoid unecessary uls - Josh Cartmell
- Updated `footer.html` identation to be more consistent - Josh Cartmell
- Refactored device handling to be based on TemplateResponse objects since dropping Django 1.1/1.2 support - Stephen McDonald
- Use filebrowser field for blog feature image, and add template handling for it - Stephen McDonald
- Removed all uses of ifequal and ifnotequal templatetags - Stephen McDonald
- Added model graph to docs - Stephen McDonald
- Change `Displayable.status` default to published - Stephen McDonald
- Create dest directories in `mezzanine.utils.tests.copy_test_to_media` - Stephen McDonald
- Prevent child pages being added to protected pages. Closes #131 - Stephen McDonald
- Added SSLMiddleware which redirects based on matching url prefixes. Updated `defaults.py` with new settings related to the middleware. Added deprecation warning if `SHOP_SSL_ENABLED` or `SHOP_FORCE_HOST` is found in settings - Josh Cartmell
- Updated deprecation warnings to work - Josh Cartmell
- Middleware now redirects back to non-secure if the request is secure but does not have a prefix from `SITE_FORCE_SSL_URL_PREFIXES` - Josh Cartmell
- Added fix for `footer.html` if a page is primary, in footer and the first in the loop - Josh Cartmell
- Removed cartridge checks from `SITE_FORCE_SSL_URL_PREFIXES` defaults. Moving to cartridge and using append - Josh Cartmell

- Restored `mezzanine.core.AdminLoginInterfaceSelector` and added a deprecation warning - Stephen McDonald
- Added the setting `TINYMCE_SETUP_JS` which controls the URL for the TinyMCE setup JavaScript file - Stephen McDonald
- Renamed SSL settings to begin with SSL and moved deprecation warnings to Cartridge - Stephen McDonald
- Moved account functionality from Cartridge into Mezzanine, and added data migrations for editable setting name changes - Stephen McDonald
- Make generated fields in `mezzanine.generic` (`_string`, `_count`, `_average`, etc) uneditable, to prevent them from appearing in admin change views that don't have explicit admin classes registered - Stephen McDonald
- Ensure generated fields in `mezzanine.generic` are unique instances - Stephen McDonald
- Fixed branch clicking in admin page tree so that open/close for a branch doesn't toggle its children (Thanks Jason Kowaleski) - Stephen McDonald
- Changed admin dropdown menu to be injected into breadcrumb area, rather than floating on it, to allow for the breadcrumb background to wrap with menu items when the browser window is thin - Stephen McDonald
- Fixed admin page tree on reload. The problem occured when reloading a page after setting an open child branch's. parent as closed. When you reloaded a page the routine that would. reopen previously opened child branches (that are currently hidden by a parent). was causing said child branch displaying both the show(+) and hide(-). icons side by side which could be seen when reopening the parent. It would also cause this said hidden, opened child branch. to no longer be registered in the opened branch cookie. So if you were to. reload the page again, this branch wouldn't be opened at all. The solution involves simply reopening all previously opened branches on. reload without worrying about adding their ID's again to the cookie. It also. avoids using the Jquery toggle() function which seemed to be the problem. that caused both the show(+) and hide(-) buttons to appear - Kowaleski, Jason
- Refactored rating form and templatetag to remove hard-coded field name - Stephen McDonald
- Raise exception if any of the generic fields are used multiple times on the same model, since we don't have access to the field being modified in the signals - Stephen McDonald
- Added migrations for `mezzanine.galleries` - Stephen McDonald
- Fail silently and return an empty list for objects given without a KeywordsField - Stephen McDonald
- Refactored comment handling into its own view, and removed `mezzanine.generic.utils.handle_comments` - Stephen McDonald
- Revert previous change for removing hard-coded rating field name, and remove the hard-coded field name by simply finding the first RatingField for the given object, since there can only be one - Stephen McDonald
- Fix logic in form export - Stephen McDonald
- In `mezzanine.forms`, allow FormEntry instances to be provided for FormForForm and handle loading and updating FieldEntry values - Stephen McDonald
- Update packages docs and re-generate settings docs - Stephen McDonald
- Remove unnecessary `time_format` handling in SplitSelectDateTimeWidget which doesn't exist in Django 1.4 - Stephen McDonald
- Add missing messages context processor for Django 1.4 - Stephen McDonald
- Allow docs to build even if model graph can't be built - Stephen McDonald
- Allow `BLOG_SLUG` to be set to an empty string, in which case the catch-all urlpatterns belong to the blog, and page urlpatterns get their own URL prefix - Stephen McDonald

- Use a generic sqlite db name in `local_settings.py` - Stephen McDonald
- Upgrade to Bootstrap 2.0 - Stephen McDonald
- Added Javascript to show only pages with children in tree, and to update this after moving pages (via drag and drop) - Kowaleski, Jason
- Add fallback for blog title when blog page isn't available - Stephen McDonald
- Fix gallery overlay close handler - Stephen McDonald
- Add the missing viewport for the responsive layout to work correctly - Stephen McDonald
- Updating doc for model customization, registering works better in `admin.py` - Ismail Dhorat
- Change the template copying option in the mezzanine-project script to default to False - Stephen McDonald
- Create entries for empty fields, so that export filtering works correctly - Stephen McDonald
- Setup `local_settings` template when testing - Stephen McDonald
- Updated -t help text to reflect that it is no longer the default - Josh Cartmell
- Updated the mezzanine-project command to have a -m option which must be specified to copy over mobile templates. The -t option now skips over mobile templates - Josh Cartmell
- Removed the `make_grappelli/filebrowser_safe` scripts as they're no longer useful since we've customized those packages - Stephen McDonald
- Remove themes from feature list - Stephen McDonald
- Version bump to 1.0 - Stephen McDonald

Version 0.12.4 (Dec 03, 2011)

- Synchronize PO files with tip - Sebastián Ramírez Magrí
- Added a note to the documentation overview about assumed Django knowledge with a reference to the tutorial - Stephen McDonald
- Let messages fail silently for Django < 1.3 - stephenmcd
- Don't rely on version checking for adding cookie-based messaging - stephenmcd

Version 0.12.3 (Nov 22, 2011)

- Fixed Disqus single-sign-on bug where message is overwritten to <message, timestamp> and returned incorrectly in payload - Brett Clouser
- Changed thumbnail test to remove test thumbnail even if test fails - Stephen McDonald

Version 0.12.2 (Nov 18, 2011)

- Added the `mezzanine.utils.html.TagCloser` class that closes open tags in a string of HTML. Used in `Displayable.description_from_content` to ensure valid HTML is returned when extracting the first block/sentence. Fixes #100 - stephenmcd

Version 0.12.1 (Nov 18, 2011)

- possibility to insert fieldsets' fields in classes extended from DisplayableAdmin (was not possible, tuples are immutable) - Zdeněk Softič
- Added handling in BaseGenericRelation for actual instance being deleted. Fixes #103 - stephenmcd
- Added testing for correct keyword string population on keyword removal - stephenmcd

Version 0.12 (Nov 05, 2011)

- added `allow_comments` flag to blog, and moved the site filed up the class hierarchy from Displayable to Slugged, plus migrations - legutierr
- KeywordManager needs to subclass CurrentSiteManager in order to take advantage of multi-site capability added to Slugged - legutierr
- This is probably the most complex migration I have written. Read inline comments for more information - legutierr
- Fixed unicode handling in CSV export in the forms app - stephenmcd
- Fixed Django 1.3/1.4 feed handling - stephenmcd
- Added fallbacks for blog feed title and description for when the blog page doesn't exist - stephenmcd
- Added response tests for the blog feeds - stephenmcd
- Added handling for spaces in keywords - stephenmcd
- Fixed meta keywords loading in blog post templates - stephenmcd
- Upgraded keyword handling in mobile templates - stephenmcd
- Changed `keywords_for` template tag to handle None being given as an instance - stephenmcd
- Added support for using generic relations as `order_with_respect_to` on subclasses of Orderable, and applied to AssignedKeyword so that keyword order is maintained - stephenmcd
- Fixed check for generic relations in Orderable - stephenmcd
- Stringify `secret_key` because hmac hates unicode - Ken Bolton
- Fix issue #97. Add PNG support - Ken Bolton
- Renamed export related areas to entries in the forms app and added handling for deleting form entries - stephenmcd
- Added `mezzanine.utils.messages` module with fallbacks for the `django.contrib.messages` app - stephenmcd
- Added a count for the number of entries displayed in the admin for the forms app - stephenmcd
- Use css selectors rather than JS for injecting the count in the admin entries view for the forms app - stephenmcd
- Added a comment to the urlconf in `project_template` describing the importance of ordering in relation to `mezzanine.pages` urlpatterns when adding your own - stephenmcd
- Added the `mezzanine.boot` app which exists for handling setup code, and added the `EXTRA_MODEL_FIELDS` setting which is used by boot to inject extra fields onto any models required via the `class_prepared` signal - stephenmcd
- Use the `DEV_SERVER` setting when setting up Grappelli media hosting - stephenmcd
- Updated the `EXTRA_MODEL_FIELDS` example in `settings.py` - stephenmcd

- Added `EXTRA_MODEL_FIELDS` to `mezzanine.conf.defaults` - stephenmcd
- Added initial docs for model field customization - stephenmcd
- Restructured the docs into more logical paragraphs and added some missing modules to the packages docs - stephenmcd
- Allow for non-keyword args for fields in `EXTRA_MODEL_FIELDS` - stephenmcd
- Initial attempt at a subclassable MixinModel for injecting fields and methods into external models - stephenmcd
- Add png & gif thumbnailing. Support for filebrowser FileBrowseField thumbnailing - Ken Bolton
- Somehow, this didn't make it up to my repo - Ken Bolton
- if setting in registry is no more registered, delete it from registry - btx
- Added sections to the model customizations docs about field injection caveats and exposing custom fields in the admin - stephenmcd
- Updated grappelli version requirement - stephenmcd

Version 0.11.10 (Sep 24, 2011)

- Upgraded pyflakes test to handle latest version of pyflakes - stephenmcd
- better fix by Stephen for dynamic inline fields focus issue - Eli Spizzichino
- Changed install command to only fake migrations when South is installed - stephenmcd
- Renamed install command to createdb and added deprecation warning for install - stephenmcd

Version 0.11.9 (Sep 21, 2011)

- Added defaults for cookie messaging with Django >= 1.3 - stephenmcd
- Moved description and keywords fields out of Displayable and into their own MetaData abstract model - stephenmcd
- Added handling for changes to the syndication app in Django 1.4 - stephenmcd
- Added feed imports to suppressed pyflakes warnings - stephenmcd
- Removed fixtures from tests - stephenmcd
- Fixed device template test - stephenmcd
- Enable iframe, xhtmlxtras in tinymce - Ken Bolton
- Bumped grappelli-safe version requirement - stephenmcd

Version 0.11.8 (Aug 23, 2011)

- Fixed incorrect setting name in device handling docs - stephenmcd

Version 0.11.7 (Aug 18, 2011)

- Upgraded DISQUS handling in the blog templates to properly use the generic app, as well as fixing DISQUS identifiers to be unique across different models - stephenmcd

Version 0.11.6 (Aug 13, 2011)

- Decorate blog posts in `blog_post_list` with lists of categories and keywords - stephenmcd
- Added a `has_childen` helper to page objects in the page menus - stephenmcd
- Fixed styling of fixed footer in admin change form when Grappelli is not used - stephenmcd
- Fixed migration of `object_pk` in Rating and AssignedKeyword - David Prusaczyk
- Added null defaults for generic migration fix - stephenmcd
- Created an install management command that combines syncdb and migrate –fake to correct the issue of initial migrations failing with multiple apps. As a result reverted `USE_SOUTH` default to True and removed the handling of south for fixture loading - stephenmcd
- Fixed a bug in orderable inlines where order fields would be wiped on inlines that only contain a file upload field - stephenmcd
- Fixed quick-blog form styling to be fluid - stephenmcd
- Fixed bug with url field hiding logic - stephenmcd
- Added a custom slugify function to `mezzanine.utils.urls` that preserves unicode chars to support non-English URLs - stephenmcd
- Updated jquery-ui version. Fixes #80 - stephenmcd
- Add placeholders for dynamic inline sorting - stephenmcd
- Fixed category decorator query in blog post list when there are no blog posts - stephenmcd
- merging pending changes to mezzanine trunk - legutierr
- Migration adding site field to dynamic settings needs to be split into separate schema and data migrations - legutierr
- Fixed slug calculation for pages so that actual parent slugs are used. Fixes #82 - stephenmcd
- fixed unicode encode error with cyrillic slugs in template loader - Andrew Grigrev
- switch to turn comments on blog posts on/off - Johnny Brown
- fixed unicode encode error with cyrillic slugs in template loader in other places - Andrew Grigrev
- changed google analytics js to what they gave me - Johnny Brown
- Added ARA Consultants to site using Mezzanine - stephenmcd

Version 0.11.5 (Jul 03, 2011)

- Changed device test to use a page it creates itself - stephenmcd
- Updated old contentpage template in the mobile theme to richtextpage - stephenmcd

Version 0.11.4 (Jul 03, 2011)

- fixes 500 error on mobile theme (bad template tag) - Owen Nelson
- Updated `processor_for` as exceptions received were TypeError `get_model()` takes at least 3 arguments (2 given) not a ValueError - Josh Cartmell
- Fixed some new pyflakes warnings - stephenmcd

- Only run thumbnail test when the test image is in the current project (eg Mezzanine dev) - stephenmcd
- Fixed tinyMCE setup to allow tables - Zeke Harris
- Fix allowing inline editing of form content on form pages by avoiding naming conflicts with the inline editing form - Josh Cartmell
- Update example settings. Fixes #70 - stephenmcd
- Don't use HTML5 required attributes on multiple checkboxes - stephenmcd
- Adding site FK to `mezzanine.conf.models.Setting` and read/write hooks to present content based on `current_site` - Ken Bolton
- Allow override of `GRAPPELLI_ADMIN_HEADLINE` and `_TITLE` in `settings.py` - Ken Bolton
- Proper setting of default values for `GRAPPELLI_ADMIN_HEADLINE` and `_TITLE`, to fix #74 - Ken Bolton
- Proper setting of default values for `GRAPPELLI_ADMIN_HEADLINE` and `_TITLE` - Ken Bolton
- Update the site for existing settings when migrating - stephenmcd
- added `post_count` to `blog_categories` tag - Michael Delaney
- Added `select_related` for blog list view - stephenmcd

Version 0.11.3 (Jun 09, 2011)

- catches exception generated when trying to retrieve the admin url for a model that is not registered, to allow some Page models not to be registered in the admin - legutierr
- migration 0004 conflated a schema migration and a data migration, which was causing problems with MySQL. The two are now separated - legutierr
- pass all form media to the template - Owen Nelson
- adding docs for `RICHTEXT_FILTER` setting - Owen Nelson
- updated docs on how to customize `RICHTEXT_FILTER` - Owen Nelson

Version 0.11.2 (May 31, 2011)

- compile language files, compiled blog, mobile, twitter language files - Alexey Makarenya
- Updated 960.gs to fluid version - stephenmcd
- Remove mezzanine from internal Mezzanine urls - stephenmcd
- Test to verify if thumbnail generation is working - Brent Hoover
- Added 500 handler view that adds `MEDIA_URL` to the context - stephenmcd
- Fixed unicode handling in KeywordsWidget rendering - stephenmcd
- Added pip requirments to `project_template` and use it to define Mezzanine's actual version number - stephenmcd
- Reverted change to storing version number to work with docs generation - stephenmcd

Version 0.11.1 (May 24, 2011)

- Upgraded comment handling to work with new comment models in base blog importer. Fixes #59 - stephenmcd
- Only look for tags if it isn't going to throw an AtributeError - rich
- Only look for tags if it isn't going to throw an AttributeError - rich
- Split `mezzanine.core.admin.DynamicInlineAdmin` out into `TabularDynamicInlineAdmin` and `StackedDynamicInlineAdmin` - stephenmcd
- Fixed missing media from dynamic admin form - stephenmcd
- Added the template filter `is_installed` which can be used to test for values in the `INSTALLED_APPS` setting from within templates - stephenmcd
- Added `is_installed` for blog app around feed urls in mobile base template - stephenmcd
- Added integration with django's sitemaps app - stephenmcd
- Added handling in KeywordsWidget for the keywords field not existing in the request. Fixes #64 - stephenmcd
- Fixed issue where `admin.StackedInlines` would not display in the admin - Josh Cartmell
- Updated `tinymce_setup.js` to only initialize when TinyMCE is available - stephenmcd
- Updated `dynamic_inline.js` to support StackedDynamicInlineAdmin - stephenmcd
- Reordered jQuery in `base_site.html` to avoid issues when Grappelli isn't installed - stephenmcd
- Added CSS classes to each of the comment fields - stephenmcd
- Addd better hanadling in the keyword widget for when no keyword field is in the request. Previous fix only corrected the field not existing in the form object - stephenmcd
- Fixed the version check for `collapse_backport.js` - stephenmcd
- Added Single-Sign-On support to Disqus templates - Brett Clouser
- Added handling for unauthenticated users and empty key settings for Disqus single sign-on - stephenmcd
- Updated auto-generated settings docs - stephenmcd
- Added some `sys.path` fixing in `manage.py` to avoid some cron issues - stephenmcd
- Changed `object_pk` fields to integer fields in the generic app to resolve some issues with Postgres - stephenmcd
- Added migrations for `object_pk` change in generic. Fixes #66 - stephenmcd
- Fixed loading of blog posts for a tag - stephenmcd

Version 0.11 (Apr 30, 2011)

- Created a `GRAPPELLI_INSTALLED` setting that is dynamically set, and made it available to JavaScript in the admin so that this can be determined reliably without depending on Grappelli specific HTML/CSS - stephenmcd
- Made the default value for the `DASHBOARD_TAGS` setting dynamically created based on whether `mezzanine.blog` is in `settings.INSTALLED_APPS` - stephenmcd
- Added commented-out versions of some common Mezzanine settings to the `project_template's` settings module - stephenmcd
- French locale for all other apps - Dominique Guardiola
- Updated inline-editing docs to include a note about the tags already being provided by themes - stephenmcd

- Added setting for specifying the delimiter for CSV exports in the forms app - stephenmcd
- Added an option to view entries in a HTML table when exporting for the forms app - stephenmcd
- Fixed `Page.get_absolute_url` to use its static slug rather than dynamic `get_slug`. Fixes #45 - stephenmcd
- Making `Query.value` a varchar(300) to allow for larger queries - John Campbell
- make value length 140 instead of 300 since the max twitter query is 140 currently - John Campbell
- Added migration for twitter query length - stephenmcd
- Converted blog categories to a ManyToManyField - stephenmcd
- Added migration scripts for blog categories - stephenmcd
- not sure how there wasn't one of these already - Tom von Schwerdtner
- Added post counts to archive and author listings for blog posts - stephenmcd
- add a label to registered settings for a more human-friendly admin UI - Tom von Schwerdtner
- A meta title for the default project homepage - Tom von Schwerdtner
- add title/tagline to admin settings - Tom von Schwerdtner
- a (slightly) better default tagline, and make settings available to templates - Tom von Schwerdtner
- Move the `LOGIN_URL` default into the project's settings module so it can be modified - stephenmcd
- Modified the `AdminLoginInterfaceSelector` middleware to recognise `next` paramters in the querystring, and redirect to those regardless of the interface option selected on the login form - stephenmcd
- Applied `SITE_TITLE` and `SITE_TAGLINE` to templates - stephenmcd
- Made description field for meta data into plain text - stephenmcd
- Added descriptions for new settings - stephenmcd
- Added styling for the blog tagline - stephenmcd
- Updated the auto-generated settings docs - stephenmcd
- Implemented initial version of custom per-page permissions - stephenmcd
- Added some template code to the gallery example in docs - stephenmcd
- Changed TinyMCE setup to properly support embed code - stephenmcd
- Integrated the `SITE_TITLE` and `SITE_TAGLINE` settings better into templates - stephenmcd
- Removed handling of HTML from `Displayable.description` - stephenmcd
- Updated the settings docs with the restored defaults for the `SITE_TITLE` and `SITE_TAGLINE` settings - stephenmcd
- Added a section to the admin customization docs about defining custom widget classes for HtmlField fields - stephenmcd
- Changed mezzanine-project script to exclude admin templates - stephenmcd
- Added note to deployment docs about setting up a cron job for Twitter feeds - stephenmcd
- Added embedded `robots.txt` to prevent spidering when DEBUG is enabled - stephenmcd
- Fixed handling of anonymous comments in the Disqus API - stephenmcd
- Changed handling of editable settings to force unicode for settings with string defaults. Fixes #52 - stephenmcd

- Initial version of refactoring comments into Django's built-in comments, and moving them into the new generic package - stephenmcd
- Added multi-site capability and tests, updated jso page fixtures to include site reference - legutierr
- added migrations for the new site field on Displayable - legutierr
- Fixed bug in login redirect - was defaulting to /accounts/profile/ upon login before and showing the logged in user a 404 error. Now defaults to /admin/ - Audrey M Roy
- Added migrate command to setup steps. Closes #54 - stephenmcd
- Fixed incorrect tag lib name in template - stephenmcd
- Added documentation regarding multi-site to the deployment page in the docs - legutierr
- Fixed mezzanine-project script where an error would occur when more than one project template with admin templates was used - stephenmcd
- Refactored the `Keywords` model to use generic relations and moved it and all related functionality into `mezzanine.generic` - stephenmcd
- Fixed a bug where `django.conf.settings` would override `mezzanine.conf.settings` - stephenmcd
- Added tests for keywords - stephenmcd
- Added migrations for keywords - stephenmcd
- Updated `mezzanine/core/media/js/dynamic_inline.js` to allow multiple DynamicInlineAdmins on a single admin page - Josh Cartmell
- Fixed a potential circular import bug - stephenmcd
- Added more error handling to the `processor_for` page processor decorator - stephenmcd
- Added delete links to the admin page tree - stephenmcd
- Updated search to respect published status - Josh Cartmell
- Small fix to Keywords Field. Stops instance from saving if keyword data is empty - Osiloke Emoekpere
- Removed `DEV_SERVER` setting from `local_settings` module template, since this is defined dynamically - stephenmcd
- Removed `south` from the `OPTIONAL_APPS` setting, since the addition of this to a project needs to be controlled manually, as the order of initial migrations for each app cannot be guarenteed and will break if used to create the tables for these apps. Added the `USE_SOUTH` boolean setting which can be defined to automatically have south added to `INSTALLED_APPS` when available. Fixes #53 - stephenmcd
- Removed handling of admin user for returning unpublished search results - stephenmcd
- Added test to ensure only published objects are returned as search results - stephenmcd
- Fixed bug where superclasses in concrete model inheritence chains would cause duplicate search results - stephenmcd
- Fixed bug where `_order` values were not being set for dynamic inlines - stephenmcd
- Added `extra_context` arg to `mezzanine.pages.views.page` - stephenmcd
- Refactored the page processor to only accept one argument since its behaviour is to only deal with one - stephenmcd
- Added note to docs about slug-based page processors - stephenmcd

- Removed migrate command from installation notes since south is no longer automatically configured - stephenmcd
- Re-sequenced the migrations for the `Displayable.site` field - stephenmcd
- Applied workaround for unexplainable Django issue where certain signals get lost - stephenmcd
- Updated settings form template to have a submit row and error note consistent with other admin change forms - stephenmcd
- Added ratings to `mezzanine.generic` and applied to the blog app - stephenmcd
- Updated auto-generated settings docs - stephenmcd
- Added handling for page menus where parent page is explicitly provided. Fixes #58 - stephenmcd
- Renamed `Content` to `RichText`, `ContentPage` to `RichTextPage`, and `HtmlField` to `RichTextField` - stephenmcd
- Fixed handling of `USE_SOUTH` setting so that south is also removed when explicitly set to False - stephenmcd
- Updated template for RichTextPage - stephenmcd
- Fixed toolbar styling for TinyMce inside the inline editing form - stephenmcd

Version 0.10.6 (Feb 13, 2011)

- blog strings from html templates - Dominique Guardiola
- Apply the CSRF token to all AJAX posts in the admin - stephenmcd

Version 0.10.5 (Feb 10, 2011)

- Updated `mezzanine.utils.importing` name in package docs - stephenmcd
- Changed cache handling to remove middleware classes if no cache backend specified - stephenmcd
- Refactored adding of optional apps so that it only occurs once, and the ordering of installed apps so that order is not modified unless necessary (eg grappelli) - stephenmcd
- Moved generation of `docs/settings.rst` and CHANGELOG from `docs/conf.py` into functions in `mezzanine.utils.docs` - stephenmcd
- Fixed admin fieldsets example in docs - stephenmcd
- Removed includes from mobile theme that replicated JavaScript common to all devices - stephenmcd
- Fixed JavaScript for Discus comments - include the absolute URL - stephenmcd
- Fixed module margin in admin dashboard - stephenmcd
- Changed Google Anylatics code so that the main tracking args can be overridden via a block - stephenmcd
- Reverted Google Analytics block in favour of checking for an existing `_gaq` JavaScript var - stephenmcd
- fix for ajax in admin not using csrf token for forms. fix for django 1.2.5 - lexual

Version 0.10.4 (Jan 28, 2011)

- Fixed regression in cache defaults. Django defaults to a 5 minute memory cache which functions with Mezzanine's caching middleware installed by default. We now set the cache backend to dummy if no cache backend is defined in the project's settings module - stephenmcd

Version 0.10.3 (Jan 28, 2011)

- Renamed the module `mezzanine.utils.path` to the more accurate `mezzanine.utils.importing` - stephenmcd
- Added the function `mezzanine.utils.importing.import_dotted_path` for importing via Python paths to names which are defined as string settings - stephenmcd
- Removed the cache defaults - stephenmcd

Version 0.10.2 (Jan 26, 2011)

- Updated docs to describe approach for adding fieldsets to subclasses of PageAdmin - stephenmcd
- Added a depth arg for `select_related` in the recent comments panel of the admin dashboard - stephenmcd
- Restored depth arg for `select_related` in blog manager - stephenmcd
- Added deployment section to docs describing the various aliases required for serving media files, and added a management command which prints these out - stephenmcd
- Grammar fix in docs - stephenmcd
- Added lost password link to login template - stephenmcd
- Fixed the handling for creating the default user when south is installed. Closes #34 - stephenmcd

Version 0.10.1 (Jan 12, 2011)

- Fixed bug in `PageAdmin._maintain_parent` where it was assumed a location header exists for a redirect, which isn't actually the case when the page is being edited via a popup window as a forgien key - stephenmcd

Version 0.10 (Dec 21, 2010)

- Renamed fixtures to not be installed with syncdb and added signal to install them when pages are first installed - stephenmcd
- Renamed example mobile template so that it won't be rendered by default - stephenmcd
- Updated device template test to only run when device templates exist - stephenmcd
- Added a setting for restricting setting available in templates - stephenmcd
- Fixed some CSS around inline editing - stephenmcd
- Added hook for third-party apps to extend existing settings - stephenmcd
- Fixed settings append hook - stephenmcd
- Backported inline editing helptext markup for Django <= 1.2 - stephenmcd
- Fixed settings append hook again - stephenmcd
- Added handling for variable template names in include tags - stephenmcd
- Cleaned up a ton of unused imports. Fixes #29 - stephenmcd
- Added initial south migrations for all apps - stephenmcd
- Added initial optional support for HTML5 with placeholder attributes in the forms app - stephenmcd
- Added support for HTML5 required attributes in the forms app - stephenmcd

- Refactored values for field types in the forms app to separate out classes and widgets - stephenmcd
- Added HTML5 field types to the forms app: date, datetime, email - stephenmcd
- Rename user variable to author in `mezzanine.blog.views.blog_post_list` to avoid clobbering Django's user context variable. Fixes #30 - stephenmcd
- Update to new author var in blog listing template - stephenmcd
- Reduced the width of text fields for field inlines in the form admin - stephenmcd
- Updated the layout for auto generated packages docs as well as adding new missing modules. Made a giant sweep of the code base adding and updating docstrings that appear in the packages docs - stephenmcd
- Removed unused admin template filter `is_page_content_model` - stephenmcd
- Fixed south compatibility with fixture loading - stephenmcd
- make save/delete buttons in admin, always visible at screen's bottom edge - lexual
- Added a CSS shadow to the inline editing form - stephenmcd
- Fixed missing hidden fields in the inline editing form - stephenmcd
- Added a split datetime widget with select fields for date parts in the inline editing form - stephenmcd
- Refactored `mezzanine.utils` module into a package - stephenmcd
- Moved pyflakes test runner into utils - stephenmcd
- Updated package docs layout with new utils package - stephenmcd
- make static save buttons in admin, not affect admin login page - lexual
- Fixed path for serving of theme assets - stephenmcd
- Moved handling of serving assets during development from project's urlconf into `mezzanine.urls` - stephenmcd
- Removed favicon handling during development - stephenmcd
- Refactored urls so that `mezzanine.urls` becomes the main point for combining urls for all the different apps. Also moved homepage url into the project's `urlconf` as it's expected to be modified - stephenmcd
- Removed use of Django's `LOGIN_FORM_KEY` from Mezzanine's `AdminLoginInterfaceSelector` middleware since it was just removed from Django trunk and now breaks. Fixes #31 - stephenmcd
- Added a background gradient to pages in the admin page tree - stephenmcd
- Moved admin submit-row buttons CSS into base admin template - stephenmcd
- Fixed serving of media files outside of a theme when a theme is defined as in development - stephenmcd
- Added support in the admin page tree for changing parents via dragging between branches - stephenmcd
- Fixed failures in Django's tests caused by automatically using a cache backend when available - stephenmcd
- Added handling for regenerating slugs when a page's parent changes - stephenmcd
- Fixed bug where editable settings were being loaded from the DB on every access - stephenmcd
- Updated each of Mezzanine's apps to use its version number as their own - stephenmcd
- Restored empty string as default `TIME_ZONE` value so Django uses the system timezone - stephenmcd
- Moved the Grappelli/Filebrowser/caching setup into `mezzanine.utils.conf` - stephenmcd
- Made `editable` template tag fail silently if None is given - stephenmcd
- Fixed overridden slugs changing on pages when their parent changes - stephenmcd

- Changed `Page.overridden` to be more reliable by not using `get_absolute_url` which can be incorrect without a permalink - stephenmcd
- tinymce: remove word styling when cutting and pasting. Remove unnecessary toolbar buttons - lexual
- remove more MS word paste junk from tinymce pasting - lexual
- Updated handling of `post_syncdb` signal to still execute when south is installed - stephenmcd
- Fixed unicode bug when non-ascii strings are used in the blog comment form and break when persisted to a cookie - stephenmcd
- Refactored out the widget for the HtmlField into its own widget that can then be replaced via the setting `HTML_WIDGET_CLASS` - stephenmcd
- Fixed bug in `post_syncdb` signal handler names - stephenmcd
- Added new hooks for page menus for determining whether a page is a child or sibling of the current page - jdeblank
- Added initial version of a mobile menu that only renders child page links - jdeblank
- Removed redundant setuptools requirement - stephenmcd
- Cleaned up unused imports - stephenmcd
- Fixed default settings ordering - stephenmcd
- Updated auto-generated settings docs - stephenmcd
- Fixed a pathing bug in creating themes on Windows - stephenmcd
- Added HTML5 form features to inline edit forms - stephenmcd
- Added a context-aware version of Django's `inclusion_tag` template tag - stephenmcd
- Moved assignment of menu helper page attributes into `Page.set_menu_helpers` and renamed some of them to be clearer in purpose - stephenmcd
- Refactored menu template tags into a single tag `page_menu` which accepts the name of the menu template to use - stephenmcd
- Added initial handling for overriding device in a cookie - stephenmcd
- Changed `mezzanine.core.models.Displayble.set_searchable_keywords` to only trigger a save when the keyword list changes - stephenmcd
- Moved the call to `mezzanine.core.models.Displayble.set_searchable_keywords` inside `mezzanine.core.admin.DisplayableAdmin` from `save_form` to `save_model` so that it is only triggered when the entire form including inline formsets are valid - stephenmcd
- Changed `mezzanine.utils.conf.set_dynamic_settings` to ensure `debug_toolbar.middleware.DebugToolbarMiddleware` is only ever added once when installed - stephenmcd
- Added a `set_cookie` function to save repeating seconds conversion and encoding - stephenmcd
- Changed the check for a device in cookies to only match if the value is a valid device - stephenmcd
- Added a `set_device` view for explictly requesting the site for a particular device via cookie - stephenmcd
- Moved mobile templates to mobile theme directory - stephenmcd
- Moved determining device from request into `mezzanine.utils.device_from_request` - stephenmcd
- Created a device aware version of Django's cache middleware that uses the device for the request as part of the cache key - stephenmcd

- Updated device section in docs to include a section about the `mezzanine.mobile` theme - stephenmcd
- Updated text for link to mobile site - stephenmcd

Version 0.9.1 (Nov 28, 2010)

- stop creation of empty p id="description" (Potentially needs refactoring) aka not too elegant - Lee Matos
- Fixed white-space in blog list template - stephenmcd
- Fixed branching of admin media hosting for Grappelli - stephenmcd

Version 0.9 (Nov 28, 2010)

- Change the logic around settings loading to avoid some untrappable errors creating the DB table - stephenmcd
- Update setting names in docs - stephenmcd
- Update conf app name in packages docs - stephenmcd
- Update to multiple DB settings - stephenmcd
- update to jquery 1.4.4 `http://blog.jquery.com/2010/11/11/jquery-1-4-4-release-notes/` - lexual
- Fixed the `blog_categories` template tag so that it returns a list of categories without duplicates - Brad Montgomery
- Added a `"get_recent_posts"` template tag - Brad Montgomery
- Update template loader and auth context processor names to newest versions with fallbacks for Django 1.1 - stephenmcd
- Add south introspection rules for `mezzanine.core.fields.HtmlField` - stephenmcd
- allow definition lists in tinymce - lexual
- Modification of the importer script to be more streamlined. Moved importer to the blog module main and still to refactor the changes to the command line module. Can be run from a django shell and import blogger and word press - ajfisher
- Initial layout for themes - stephenmcd
- finished refactoring of importers module and wrote new import blog handler to import the various blog types into mezzanine. Also stripped down the params to be passed in on the word press blog - now treating any path as a url and dealing with it system side rather than user side - ajfisher
- Added documentation around blogger import stuff - ajfisher
- Remove some old redundant template tag loading - stephenmcd
- Add admin change logging to inline editing - stephenmcd
- Allow newer versions of Django to determine full paths for templates in the `start_theme` command - stephenmcd
- if image is already the right size, don't change it (fixes bug where image quality is degraded if same `size.)` - lexual
- Add copying of media files to `start_theme` command - stephenmcd
- Initial support for hosting a theme - stephenmcd
- Fix check for exact image size in thumbnail template tag - stephenmcd

- Make use of conf module's name within itself dynamic - stephenmcd
- Create a `path_for_import` utils function for calculating package/module paths - stephenmcd
- Add media hosting for a theme when defined - stephenmcd
- Further refactoring of the import process using a BaseImporterClass which is a Command and then setting up specific implementations for Wordpress and Blogger - ajfisher
- Modification to the docs in order to update the new structure of the commands and also how to implement a new importer class - ajfisher
- removed all the now-superfluous files - ajfisher
- Wrap lines in blog import docs - stephenmcd
- Modifications to make the class abstraction more tidy and clean up some other bits and pieces of code as well - ajfisher
- First round of edits for the blog import docs - stephenmcd
- Fix up constructor logic - stephenmcd
- Fix `mezzanune_user` reference in base blog importer - stephenmcd
- Move the output messages for blog importing into the base importer class - stephenmcd
- Fix settings access for THEME in `urls.py` - stephenmcd
- Fix duplicate months in archive list for blog - stephenmcd
- Initial version of `install_theme` command - stephenmcd
- Add handling for `interactive` option in `install_theme` command - stephenmcd
- Rename scripts directory to bin for consistency with Django - stephenmcd
- Rename Blog importer `convert` method to `handle_import` and pass it options directly to mimic Django commands more closely - stephenmcd
- Clean up unused exceptions in Blog importer - stephenmcd
- Add a `old_url` arg for posts in base Blog importer for creating redirects - stephenmcd
- Upgrade `import_tumblr` command to use new importer base - stephenmcd
- Add handling in the `import_tumblr` command for more posts that a single call to Tumblr's API allows - stephenmcd
- Add handling for verbosity option in base Blog importer - stephenmcd
- Add handling for all post types in the `import_tumblr` command - stephenmcd
- Fix some errors and add Tumblr info to the blog importing doc - stephenmcd
- Move Google Analytics and `editable_loader` tag into their own include template `footer_scripts.html` - stephenmcd
- Add docs for themes - stephenmcd
- Rename `recent_posts` blog template tag to be consistent with other tags - stephenmcd
- Add recent blog posts to `filter_panel.html` - stephenmcd
- js fix for ie bug with formbuilder - lexual
- Modified the blog's `filter_panel` tempate so <ul> tags get closed properly - Brad Montgomery
- More robust handling for class-based views in mobile middleware. Closes #23 - stephenmcd

- add primary menu id to the UL for semantic and/or styling uses - Lee Matos
- Moved `mezzanine.templates` into a package - stephenmcd
- Add context-aware replacements for Django's `render_to_response`, `select_template`, `get_template` and template tags `include` and `extend` - stephenmcd
- Changed calls to `select_template` and `render_to_response` to use Mezzanine's context-aware versions - stephenmcd
- Added main handling for device specific template directories - stephenmcd
- Added a context-aware replacement for Django's `direct_to_template` - stephenmcd
- Moved the test mobile homepage into its device specific subdirectory - stephenmcd
- Fixed renaming of node class in `extends` tag - stephenmcd
- Replaced mobile middleware test with device specific template test - stephenmcd
- Added "blog-post-tile" class for semantic/styling purposes - Lee Matos
- Added documentation for device specific template loading - stephenmcd

Version 0.8.5 (Nov 10, 2010)

- CSS update for default templates - stephenmcd
- Add more fine-grained error handling for `tumblr_import` management command - stephenmcd
- Change TinyMCE options to relax allowed HTML - stephenmcd
- CSS updates to inline editing form - stephenmcd
- Initial version of admin dashboard plugin system with Quick Blog and Recent Comments implemented as dashboard widgets - stephenmcd
- Convert remaining dashboard sections into dashboard tags - app list and recent actions - stephenmcd
- Add the new screenshot - stephenmcd
- Add docstring to inline editing view - stephenmcd
- Add basic support for class-based views in mobile middleware and a more explicit check for unique mobile template names - stephenmcd
- Backed out changeset: c2ed0a189648 - stephenmcd
- Re-apply `TINYMCE_URL` setting, lost from merge - stephenmcd
- Move settings for forms app into main settings module - stephenmcd
- Fix `filebrowser_safe` generator script to add a dummy `Image` module which will prevent breaking when PIL isn't installed. Closes #15 - stephenmcd
- Give the `ContentPage` model a more descriptive name for the content type dropdown menu in the admin page tree - stephenmcd
- Convert `mezzanine.settings` into an app with values lazy loaded via DB - stephenmcd
- Add a default user when syncdb is called - stephenmcd
- Rewrite settings app to give more control over when settings are loaded so that fewer DB queries are used - stephenmcd
- Prevent settings from being loaded from DB during syncdb - stephenmcd

- Change settings from dicts into objects so they can be more easily identified when iterating through the `mezzanine.settings.defaults` module - stephenmcd
- Add admin view and form for editing all settings - stephenmcd
- Fix `post_syncdb` signal for demo user to work with Django 1.1 - stephenmcd
- Fix casting of boolean settings from DB - stephenmcd
- Add a redirect on successful update of settings - stephenmcd
- Add tests for settings app - stephenmcd
- Fix custom field HTML for Django 1.1 - stephenmcd
- Add hook for apps to register their own settings - stephenmcd
- Update docs to use new settings app - stephenmcd
- Fix export for forms with deleted fields in forms app - stephenmcd
- Allow comma separated list of field choices to contain commas when quoted in forms app - stephenmcd
- Add a back button to the admin export view in the forms app - stephenmcd
- Fix missing import in forms export - stephenmcd
- Allow multiple fields to be used in a single editable tag - stephenmcd
- Update docs with information about grouping together fields for inline editing - stephenmcd
- Update creation of default user to only run with `--noinput` passed to `syncdb` - stephenmcd
- `tree_menu_footer` tag added. Exact same as `"tree_menu"` but checks if in footer not if in nav - lexual
- Hide the slug field and delete button in the admin for pages with an overridden urlpattern - stephenmcd
- Display list bullets and numbers in content - Eric Floehr
- Fix rendering editable fields when not authenticated - stephenmcd
- Update `mezzanine-project` script to remove pyc files when creating new projects - stephenmcd
- Remove admin menu from popups - stephenmcd
- Add `mezzanine.core.templatetags.mezzanine_tags.thumbnail` for image resizing - stephenmcd
- Add docs for the `mezzanine.settings` app - stephenmcd
- Strip newlines from commit messages in the auto-generated CHANGELOG - stephenmcd
- use export instead of checkout - Tom von Schwerdtner
- Use svn export in grappelli/filebrowser scripts. Closes #16 - stephenmcd
- Fix split on commit author in automated CHANGELOG generator - stephenmcd
- Fix unrequested settings being loaded from DB - stephenmcd
- Allow no names to be provided when calling `editable_settings` - stephenmcd
- Sort setting names for settings form in admin - stephenmcd
- Add Django as a dependency and remove import from `project_template.settings` in `setup.py` which depends on Django - stephenmcd
- Remove redundant call to `jQuery.noConflict` since `editable_loader` is now at end of the document. Also check for an existing jQuery instance before including it - stephenmcd

- Fix isDirty() check for file fields in dynamic inlines - stephenmcd
- Fix inline editing for file uploads - stephenmcd
- Give each inline editable form field a unique ID to allow multiple TinyMCE editors to work correctly - stephenmcd
- add `csrf_token` to form for inline editing (django 1.2 fails without `this.`) - lexual
- admin now contains link back to site - lexual
- Move site link in admin to user-tools panel - stephenmcd
- move toolbar for editable inline to the right hand side - lexual
- Backed out changeset 50aa6171231d - lexual
- move inline editable toolbar to top right - lexual
- Make number of comments for a `BlogPost` available via `BlogPostManager` - stephenmcd
- Add `mezzanine.utils.admin_url` which handles reversing different admin URLs. Also rename `admin_url` template tag to `try_url` to better reflect its purpose - stephenmcd
- Add a (yet to be used) `SingletonAdmin` class for creating admin classes that manage models with a single instance - stephenmcd
- Clean up the dynamic inline hooks Django uses that get left behind by using Grappelli's inline template - stephenmcd
- Remove redundant reference to jquery - stephenmcd
- Different approach to cleaning up `__prefix__` templates from inlines - just remove them - stephenmcd
- Hide the unwanted add link - stephenmcd
- `admin_app_list` template tag bugfix - lexual
- make inline editable forms pretty - lexual
- Backed out changeset: 7a1d5a321032 - stephenmcd
- Add support for custom navigation items in `ADMIN_MENU_ORDER` and configure `filebrowser` as an item - stephenmcd
- Add docs for custom navigation items in admin - stephenmcd
- Add Wordpress support to blog importer - ajfisher
- Added importer with command line option - ajfisher
- 1. Some changes to the importer module in order to clean up. 2. Implementation of framework to use tumblr in importer module. 3. Addition of new tumblr module, adapting from @stephenmcd 's previous work but extending it to work into new importer framework - ajfisher
- Catch DatabaseError instead of trying to check for syncdb when reading DB settings to allow for other DB management related commands to run such as south - stephenmcd
- Rename `mezzanine.settings` to `mezzanine.conf` - stephenmcd
- Make the DatabaseError import compatible with Django 1.1 - stephenmcd
- Put fixtures into a potentially more stable order - stephenmcd
- Update the admin menu with the new conf name - stephenmcd
- fixed some code logic to enumerate more cleanly and removed the testing 5 item max results - ajfisher

- modified to include tries on the feedparser import and exit gracefully if not. Also cleaned up some enumeration stuff and making the tags code into a list comprehension - ajfisher
- added some graceful exit handling if the gdata library isn't available - ajfisher
- streamlined tag stuff to use a list comprehension - ajfisher
- Replace the approach of calling `mezzanine.conf.load_settings` to create new instances of settings objects with a single instance via `mezzanine.conf.settings` that contains a method `use_editable` which when called will mark the settings object for reloading settings from the db - stephenmcd
- Refactor settings loading to reload settings when `use_editable` called - stephenmcd
- Remove unused func `editable_settings` - stephenmcd
- Explicitly evaluate the results for blog template tags so that queries are only executed once - stephenmcd
- Replace `load_settings` template tag with a context processor for a global settings object - stephenmcd
- Remove the `SETTINGS_EDITABLE` setting and check for `mezzanine.conf` in installed apps - stephenmcd
- Remove the `MEZZANINE_` prefix from checking project's settings for default values, since non-mezzanine apps may register settings themselves - stephenmcd
- Group the form fields for editable settings by prefix - stephenmcd
- Update documentation to reflect refactoring of the conf app - stephenmcd
- Allow the `BlogCategoryAdmin` to be displayed in the admin menu when explicitly defined in `ADMIN_MENU_ORDER` - stephenmcd

Version 0.8.4 (Sep 30, 2010)

- Fix PostgreSQL error on tweet lookup - stephenmcd
- Use dynamically generated intro for posts in blog listing page rather than the description field - stephenmcd

Version 0.8.3 (Sep 29, 2010)

- Workaround for when mezzanine is hosted under a different urlspace. (Only tested with Django admin, not grappelli). The `keywords.js` file needs to reference a mezzanine URL, from the admin site. It used to use a hard-coded string with a root-absolute path, but this failed when mezzanine was hosted elsewhere. Instead, we now reference a global Mezzanine JS object, which is set by template, using URL reversal to find the correct url. This requires a reworking in how the PageAdmin object accesses its Media settings - previously the list of js files was calculated at module load time, but at this stage the url reversal mechanism won't work, because the `urls.py` aren't all loaded yet, Instead, we hide the list generation inside a lazy iterator object and create the list on demand - Toby White
- Fix admin menu for earlier than Python 2.6 which lacks `tuple.index` - stephenmcd
- add active class to footer menu - lexual
- Don't fail if `PACKAGE_NAME_FILEBROWSER` or `PACKAGE_NAME_GRAPELLI` aren't set - just don't try & load them - Toby White
- Fix grappelli/filebrowser package creation scripts to be Python 2.5 compatible. Closes #12 - stephenmcd
- Create a template tag for reversing urls from within admin templates that fail silently when the url can't be reversed, as this is always the case when running admin tests. Apply this to both the admin dropdown menu and to the base admin template making all templates aware of the `admin_keyword_submit` URL so that it does not need to be hard-coded - stephenmcd

- Backed out changeset: d43f3e430d1f - stephenmcd
- Replace MobileTemplate middleware with a decorator. If mezzanine is being used as an application within another project, then the MobileTemplate middleware may not be appropriate to use on URLs outside of mezzanine's control. In fact, if the project uses other calling conventions (eg class-based views) then the middleware may fail completely - Toby White
- Fix positioning of admin dropdown menu in Firefox. Closes #11 - stephenmcd
- Let the location of the tinymce scripts be overridden - Toby White
- Give the Page object a `get_admin_url` property, which we can use to provide direct links to a Page's admin page from the editable toolbar - Toby White
- add id's to tree-menu, fix bug with multiple "first" class being set - lexual
- Add a filtering form for exporting responses in the forms app - stephenmcd
- Add `branch_level` and `html_id` attributes to pages in menu template tags - stephenmcd
- Add `TEST_DATABASE_COLLATION` for MySQL - stephenmcd
- Fix field length of test data - stephenmcd
- Remove trailing commas from `tinymce_setup.js` that break IE. Fixes #14 - stephenmcd

Version 0.8.2 (Sep 23, 2010)

- Backed out changeset 0e7907eef4fc - lexual
- move editable-loader to bottom of template to fix weird webkit layout bug - lexual
- 960 stuff into separate files - lexual
- custom css in separate files - lexual
- Modify absolute `ADMIN_MEDIA_PREFIX` value when using grappelli to read in ip/port from `sys.argv` - stephenmcd
- Clean up a bunch of dead code. Fixes #10 - stephenmcd
- Test for existence of TinyMCE before using it in JS - Toby White
- Fix missing quotes for `CONTENT_MEDIA_URL` setting - stephenmcd
- Type in setting function - stephenmcd
- Fix handling of empty field values in forms app by not saving them - stephenmcd

Version 0.8.1 (Sep 19, 2010)

- No changes listed.

Version 0.8 (Sep 19, 2010)

- Allow search fields for `SearchableManager` to be defined across multiple models in an inheritance chain - stephenmcd
- Refactor models to remove the `content` field from the `pages.Page` model (and therefore the `core.Displayable` model from which it inherits) so that custom content types can be created without the `content` field. Introduces a new default content type `pages.ContentPage` - stephenmcd

- Remove `BLOG_TITLE` and `BLOG_DESCRIPTION` from `mezzanine.settings` and replace use of these with the title and description of the blog page from the pages app allowing them to be in-line editable - stephenmcd
- Separate dynamic inlines into its own js file - stephenmcd
- Make class name unique for dynamic inlines - stephenmcd
- Fixed a bug with the stripping of comment dates - ajfisher
- Added comment migration to the post importing. Have disabled keywords /. tags for the moment due to an error from refactoring - ajfisher
- added some exception handling to start cleaning up things ready for. the proper management command set up - ajfisher
- removed some of my testing params and made them generic - ajfisher
- Fix with statement for `Python2.5` in `setup.py`. Closes #9 - stephenmcd
- Refactor ordering and dynamic "Add another" enhancements to admin inlines so that they explicitly target the correct inlines - stephenmcd
- Move scripts into scripts directory and use OptionParser in mezzanine-project script to allow options for copying templates, package source and specifying an alternate package to install from - stephenmcd
- Fix logic of checking a page's slug to be selected in `page_menu` template tag - stephenmcd
- Remove the list of apps/models from the admin dashboard and move them into a navigation menu persistent throughout the entire admin - stephenmcd
- Trap failure to resolve admin URLs so that tests can pass - stephenmcd
- Set `mezzanine.core.admin.DynamicInlineAdmin.extra` to 20 unconditionally - stephenmcd
- Try and check for jQuery before loading it for admin menu - stephenmcd
- Fix styling of messages to prevent them being layered on top of the admin menu - stephenmcd
- Update auto-generated settings docs - stephenmcd

Version 0.7.4 (Sep 11, 2010)

- Use `ADMIN_MEDIA_PREFIX` in path to TinyMCE js. Closes #6 - stephenmcd
- Refactor generation of `Displayable.description` to not explicitly use content field - stephenmcd
- Fix sequence of styling for selected nav in tree menu - stephenmcd
- Let blog views render even if the blog page object doesn't exist - stephenmcd
- Add a test for generated page descriptions - stephenmcd
- Allow test for overriden pages to pass when blog page doesn't exist - stephenmcd
- fix up footer positioning - lexual
- Fix field length for field types in forms app - stephenmcd
- Update `mezzanine-project` script to copy templates into newly created project - stephenmcd
- Fix missing enctype in forms template for forms with file uploads - stephenmcd
- Add a new `help_text` field to form field model in forms app - stephenmcd
- Add `email_subject` and `email_message` fields to form model in forms app - stephenmcd

- Fix `pages.page_processors.processor_for` to return the function it decorates so they can be referenced from their modules for documentation purposes - stephenmcd
- Fix docs in `mezzanine.utils` - stephenmcd
- Add `mezzanine.forms` to package docs - stephenmcd

Version 0.7.3 (Sep 03, 2010)

- Alignment fixes to the footer menu - stephenmcd

Version 0.7.2 (Sep 02, 2010)

- Refactor `mezzanine.template` to use `functools.wraps` - stephenmcd
- Move `local_settings` module into a template - stephenmcd
- Align TinyMCE width with other admin fields - stephenmcd
- Refactor slug creation functionality out of `Displayable` model into `Slugged` model - stephenmcd
- Add `BlogCategory` model and associated functionality - stephenmcd
- Added BooleanField `in_navigation` and `in_footer` to `Page` model to allow for controlling navigation placement of pages - stephenmcd
- Bugfix to slug fields - change to CharField to allow slashes - stephenmcd
- Better styling for the footer nav - stephenmcd
- Add a `primary` attrib for page objects in menu templates - stephenmcd
- More styling enhancements to footer menu - stephenmcd
- Add new fixtures for demonstrating footer menu - stephenmcd

Version 0.7.1 (Aug 30, 2010)

- Bugfix to mobile middleware for view functions without keyword args - stephenmcd

Version 0.7 (Aug 29, 2010)

- Integrate 960.gs CSS framework into default templates - stephenmcd

Version 0.6.4 (Aug 29, 2010)

- Backed out changeset: 8dac998c6f0c - stephenmcd
- Add `expiry_date` field to `DisplayableAdmin` - stephenmcd
- Change `if` tags in breadcrumbs and `toplevel_menu` templates to be Django 1.1 compatible, and to use the `page.selected` attribute rather than `template_utils` lib, allowing it to be removed from `pages_tags` - stephenmcd
- Use consistant naming for each type of page menu and include all types of page menus in default templates - stephenmcd
- Create a custom breadcrumb menu for blog posts - stephenmcd

- Replace the `setting` tag with a `load_settings` tag that takes a list of setting names and injects them into the template context - stephenmcd
- Bugfix template tag name for admin page menu - stephenmcd

Version 0.6.3 (Aug 26, 2010)

- Bugfix login redirect to be compatible with Django 1.2 - stephenmcd

Version 0.6.2 (Aug 26, 2010)

- More error handling to CHANGELOG generator - ensure hg repo also exists - stephenmcd
- Add a `button_text` field to forms model for editing the text of the form's submit button - stephenmcd
- Bugfix to forms button text - stephenmcd
- Add new field `Displayable.expiry_date` and relevant handling in `PublishedManager.published` - stephenmcd
- Add field for default values in forms app and new field types: Check boxes, Radio buttons, Hidden - stephenmcd
- Add `login_required` field to page model for restricting pages to authenticated users - stephenmcd

Version 0.6.1 (Aug 23, 2010)

- Update to Mezzanine 0.6 - VanL
- Update `pages_tags` to include comparisons and toplevel/breadcrumbs tags; added associated templates - VanL
- Set `TIME_ZONE` to use the system timezone - stephenmcd
- Change CHANGELOG generator to fail silently when mercurial isn't installed - stephenmcd

Version 0.6 (Aug 09, 2010)

- Bugfix to mobile template test to allow for no template inheritance - stephenmcd
- Initial import of django-forms-builder - stephenmcd
- Bugfix to `orderable_inline.js` to correctly detect dirty checkboxes - stephenmcd
- Move `mezzanine.core.models.HtmlField` to new module `mezzanine.core.fields` - stephenmcd
- Allow model/field passed to `editable` templatetag to contain extra dot notation - stephenmcd
- Convert `forms_builder` app to a Mezzanine content type - stephenmcd
- Extend admin definitions from inherited admin classes for `FormAdmin` - stephenmcd
- Bugfix for generating slugs with parent pages for subclasses of `Page` - stephenmcd
- Initial commit of the importer code for blogger. Comprises the baseline generic importer code for mezz that will be used by all input types and the baseline processor for blogger. 0.1 - ajfisher
- Updated sequence of classes in documentation to run correctly - thanks Nick Hagianis for picking this up - stephenmcd

- Handful of patches to correct bugs around creating slugs, titles and ordering for pages and their subclasses - stephenmcd
- Add a contact form to fixtures - stephenmcd
- Added built-in styling to form fields in forms app - stephenmcd
- unnecessary imports removed ? - lexual
- Added missing line in docs for in-line editing example - stephenmcd
- Remove natural keys from fixtures for Django 1.1 compatibility - stephenmcd

Version 0.5.4 (Jul 25, 2010)

- Bugfix to in-line editing view - missing import - stephenmcd
- Moved setting of class attribute for TinyMCE into HtmlField - stephenmcd
- Added loading animation to in-line editing - stephenmcd

Version 0.5.3 (Jul 24, 2010)

- Use names of packages from settings in setup script - stephenmcd
- Make changelog generator in Sphinx conf list changesets within a version in chronological order - stephenmcd
- Update CSS for in-line editing toolbar to stay fixed at top of the page - stephenmcd
- Added handling for models to define their own `editable` method for controlling in-line editing - stephenmcd
- Added the abstract model `Ownable` for defining models with instances owned by users, containing `is_editable` hook and admin class for setting the owner of new objects and restricting objects to their owners - stephenmcd
- Apply `Ownable` abstract model to `BlogPost` model and related classes - stephenmcd
- Wrap quickblog form in permission check - stephenmcd

Version 0.5.2 (Jul 22, 2010)

- Allow interface selection on admin login screen to prepopulate from querystring - stephenmcd
- spelling fixed in docs - lexual
- Added more backward-compatible csrf handling - stephenmcd
- Added more robust handling for csrf and apply to forms - stephenmcd
- Bugfix setting author of blog post in BlogPostAdmin to be compatible with DisplayableAdmin - stephenmcd

Version 0.5.1 (Jul 18, 2010)

- Bugfix to permission check for in-line editing - stephenmcd
- Bugfix to persist values for in-line TinyMCE fields - stephenmcd
- Created HtmlField and TinyMceWidget for more control over targetting textareas as TinyMCE fields - stephenmcd
- Bugfix to TinyMceWidget name - stephenmcd

- Include Csrf Middleware when available - stephenmcd

Version 0.5 (Jul 18, 2010)

- Added 'Posted by' text to blog posts - stephenmcd
- Fixed grammar error in docs - stephenmcd
- Added routine to Sphinx conf to auto-generate changelog from mercurial repo - stephenmcd
- Change admin title to Mezzanine - stephenmcd
- Make slugs editable in admin - stephenmcd
- Bugfix links to RSS feeds - stephenmcd
- Update `to_end_tag` with context and token args, but only use as many args as the tag supports - stephenmcd
- Created system for inline-editing - stephenmcd
- Apply in-line editing to templates - stephenmcd
- Add option to admin login screen to log into site for in-line editing - stephenmcd
- Added docs for in-line editing - stephenmcd

Version 0.4 (Jul 11, 2010)

- Added search functionality and moved pagination out into utils and templatetags - stephenmcd
- Remove weight from search results output - stephenmcd
- A bunch of updates to doc strings - stephenmcd
- Added documentation for search API - stephenmcd
- Added highlighting to blog post author's own comments - stephenmcd
- Save blog commenter's details in a cookie - stephenmcd
- Bugfix to links in recent comments section of admin dashboard - stephenmcd

Version 0.3.5 (Jul 08, 2010)

- Bugfix to page template missing tag library - stephenmcd
- Bugfix to tests loading and version bump - stephenmcd

Version 0.3.4 (Jul 08, 2010)

- Add blog migration to feature list - stephenmcd
- Added support for natural keys - stephenmcd
- Bugfix to natural key format - stephenmcd
- Cleaned up interface for custom tag types - stephenmcd
- Update docs with references to new modules and version bump - stephenmcd

Version 0.3.3 (Jul 06, 2010)

- No changes listed.

Version 0.3.2 (Jul 06, 2010)

- Reformatted docs to lines less than 80 chars - stephenmcd
- Revert some of `mezzanine.settings` back to not being overridable - stephenmcd
- Added routine to sphinx `conf.py` to auto-generate docs for `mezzanine.settings` - stephenmcd
- Prevent sphinx blank-line warning - stephenmcd
- Make building docs `Python2.5` compatible - stephenmcd
- Bugfix for losing parent ID when editing existing child pages - stephenmcd
- fix bug with ordering field in pages. Add error handling for page ordering - lexual

Version 0.3.1 (Jul 04, 2010)

- Fixed some spelling mistakes throughout docs - stephenmcd
- Bugfix to unique slug generation method - stephenmcd
- Added redirects app to default settings - stephenmcd
- Added management command to blog app for migrating a Tumblr blog - stephenmcd
- Allow members of `mezzanine.settings` to be defined in the project's settings module prefixed with **MEZZANINE_** - stephenmcd

Version 0.3.0 (Jul 03, 2010)

- Bugfix to template lookup for custom content model - stephenmcd
- Added page processor system for manipulating context and response per page type - stephenmcd
- Added docs for page processors and bumped version - stephenmcd

Version 0.2.4 (Jul 02, 2010)

- Add warning to `mezzanine_project` script to prevent the user from creating a project name that conflicts with an existing package or module - stephenmcd
- fix `mezzanine_project.py` handling –options or multiple arguments - lexual
- `project_name` variable instead of continual using `sys.argv[1]` - lexual
- make proj name start with "-" illegal and print usage - lexual

Version 0.2.3 (Jun 30, 2010)

- added `content_model` to json - lexual
- Bugfix to dynamic admin inlines - stephenmcd

Version 0.2.2 (Jun 27, 2010)

- No changes listed.

Version 0.2.1 (Jun 27, 2010)

- Added more documentation around extending pages - stephenmcd
- Update fixtures with new name for ordering field - stephenmcd
- Added notes about contributing with links to github and bitbucket repos - stephenmcd
- Fixes to Mezzanine's sphinx theme - stephenmcd
- Added initial layout template for docs with GA code - stephenmcd
- Bugfix to template loading in page view - stephenmcd

Version 0.2 (Jun 27, 2010)

- Reinstate required ordering for correct admin template loading - stephenmcd
- Fixed incorrect project name in license - stephenmcd
- Created framework for inheriting from `Page` model to create custom content types for pages, and added new abstract model `Orderable` for managing orderable models - stephenmcd
- Initial version of documentation - stephenmcd
- Bugfix to submit overriding for keyword field - stephenmcd

Version 0.1.4 (Jun 15, 2010)

- Switch out filebrowser to use a custom version as done with grappelli - correctly packaged and Django 1.1 compatible - stephenmcd
- Add script for generating fork of filebrowser - stephenmcd
- Simplify structure for optionally installed apps and exclude all optional apps from testing - stephenmcd
- In mobile middleware, don't assume user-agent exists since the test client doesn't use one - stephenmcd
- Bugfix to mobile middleware - missing imports - stephenmcd
- Made comments IP address nullable - stephenmcd
- Use url tags in templates instead of `get_absolute_url` - stephenmcd
- Don't assume request is in context in inclusion tags - stephenmcd
- Added error templates and example mobile template for homepage - stephenmcd
- Added test suite and version bump - stephenmcd

Version 0.1.3 (Jun 14, 2010)

- Moved the blog landing page's slug into a setting - stephenmcd
- Add homepage to menu - stephenmcd
- Update to layout of sharing panel - stephenmcd

- Bugfix to AJAX submit for admin keywords field - stephenmcd
- Added a dynamically set "selected" attribute for pages rendered in the page menu - stephenmcd
- Bugfix to tweets for search terms - missing profile image and invalid date format - stephenmcd
- Bugfix to tweets - invalid import - stephenmcd
- Added demo twitter feed - stephenmcd
- Bugfix to blog view - old variable name - stephenmcd
- Added username fallback for displaying author's name for list of blog posts - stephenmcd
- Added "powered by" copy - stephenmcd
- Added setting `GOOGLE_ANALYTICS_ID` for integrating Google Analytics - stephenmcd
- Added setting `PAGES_MENU_SHOW_ALL` to control whether all levels in page menu are shown by default - stephenmcd
- Changed manual file exclusion in setuptools script to maintain owner and permissions - stephenmcd

Version 0.1.2 (Jun 10, 2010)

- Remove local settings module from repo and packaging - stephenmcd
- Actual `local_settings` module removal - stephenmcd

Version 0.1.1 (Jun 10, 2010)

- No changes listed.

Version 0.1 (Jun 10, 2010)

- No changes listed.

m

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W