
django-photologue Documentation

Release 3.1

Justin Driscoll/Richard Barran

November 03, 2014

1	Installation & configuration	3
1.1	Installation	3
1.2	Dependencies	3
1.3	Configure Your Django Settings	4
1.4	Add the urls	4
1.5	Sync Your Database	5
1.6	Instant templates	5
1.7	Sitemap	5
1.8	Sites	6
1.9	Amazon S3	6
2	Customisation: extending templates	7
3	Customisation: Settings	9
3.1	PHOTOLOGUE_GALLERY_PAGINATE_BY	9
3.2	PHOTOLOGUE_PHOTO_PAGINATE_BY	9
3.3	PHOTOLOGUE_GALLERY_LATEST_LIMIT	9
3.4	PHOTOLOGUE_GALLERY_SAMPLE_SIZE	9
3.5	PHOTOLOGUE_IMAGE_FIELD_MAX_LENGTH	9
3.6	PHOTOLOGUE_SAMPLE_IMAGE_PATH	10
3.7	PHOTOLOGUE_MAXBLOCK	10
3.8	PHOTOLOGUE_DIR	10
3.9	PHOTOLOGUE_PATH	10
3.10	PHOTOLOGUE_MULTISITE	11
3.11	PHOTOLOGUE_ENABLE_TAGS	11
4	Customisation: Admin	13
4.1	Create a customisation application	13
4.2	Changing the admin	13
4.3	Possible uses	14
5	Customisation: Views and Urls	15
5.1	Create a customisation application	15
5.2	Changing pagination from our new urls.py	15
5.3	Values that can be overridden from urls.py	16
5.4	Changing views.py to create a RESTful api	16
6	Customisation: Models	19
6.1	Create a customisation application	19

6.2	Extending	19
7	Customisation: third-party contributions	21
7.1	Old-style templates	21
8	Contributing to Photologue	23
8.1	Example project	23
8.2	Workflow	23
8.3	Coding style	23
8.4	Unit tests	23
8.5	Documentation	24
8.6	Translations	24
8.7	New features	24
8.8	And finally...	24
9	Changelog	25
9.1	3.1 (2014-11-03)	25
9.2	3.0.2 (2014-09-23)	25
9.3	3.0.1 (2014-09-16)	25
9.4	3.0 (2014-09-15)	25
9.5	2.8.3 (2014-08-28)	26
9.6	2.8.2 (2014-07-26)	26
9.7	2.8.1 (2014-07-26)	26
9.8	2.8 (2014-05-04)	27
9.9	2.7 (2013-10-27)	27
9.10	2.6.1 (2013-05-19)	28
9.11	2.6 (2013-05-19)	28
9.12	2.5 (2012-12-13)	28
9.13	2.4 (2012-08-13)	29
10	Indices and tables	31
	Python Module Index	33

The Photologue documentation is being transferred from a Google Code wiki to a Sphinx-generated setup (if you're reading this at readthedocs.org, then you're looking at these docs).

This documentation is accurate and up-to-date (hopefully!); [the old docs on the Google Code wiki](#) are extensive but have not been updated in a long time.

Contents:

Installation & configuration

1.1 Installation

The easiest way to install Photologue is with [pip](#); this will give you the latest version available on [PyPi](#):

```
pip install django-photologue
```

You can also take risks and install the latest code directly from the Github repository:

```
pip install -e git+https://github.com/jdriscoll/django-photologue.git#egg=django-photologue
```

This code should work ok - like [Django](#) itself, we try to keep the master branch bug-free. However, we strongly recommend that you stick with a release from the PyPi repository, unless if you're confident in your abilities to fix any potential bugs on your own!

1.1.1 Python 3

Photologue works with Python 3 (3.3 or later). You'll need Django version 1.5 or higher.

1.2 Dependencies

4 apps that will be installed automatically if required.

- [Django](#).
- [Pillow](#).
- [Django-sortedm2m](#).
- [Django-model-utils](#).

And 2 dependencies that you will have to manage yourself:

- [Pytz](#). See the Django release notes [for more information](#).
- [South](#). Only applies for Django 1.6. Note that South 1.0 is not compatible with Python 3 - you should install version 0.8.4 instead.

Note: Photologue has - more or less - the same support policy as Django. Photologue 3.0 is tested against Django 1.6 and 1.7.

1.2.1 That troublesome Pillow...

Pillow can be tricky to install; sometimes it will install smoothly out of the box, sometimes you can spend hours figuring it out - installation issues vary from platform to platform, and from one OS release to the next, so listing them all here would not be realistic. Google is your friend, and it's worth noting that Pillow is a fork of PIL, so googling 'PIL installation <your platform>' can also help.

1. You should not have installed both PIL and Pillow; this can cause strange bugs. Please uninstall PIL before you install Pillow.
2. In some situations, you might not be able to use Pillow at all (e.g. if another package has a dependency on PIL). Photologue has a clumsy answer for this: write a temporary file `/tmp/PHOTOLOGUE_NO_PILLOW`, then install Photologue. This will tell Photologue to install without Pillow. It *should* work, but it hasn't been tested!
3. Sometimes Pillow will install... but is not actually installed. This 'undocumented feature' has been reported by a user on Windows. If you can't get Photologue to display any images, check that you can actually import Pillow:

```
$ python manage.py shell
Python 3.3.1 (default, Sep 25 2013, 19:29:01)
[GCC 4.7.3] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from PIL import Image
>>>
```

1.3 Configure Your Django Settings

1. Add to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    # ...other installed applications,
    'photologue',
    'sortedm2m',
    'south',    # Only if you're relying on South for migrations.
)
```

2. Confirm that your `MEDIA_ROOT` and `MEDIA_URL` settings are correct (Photologue will store uploaded files in a folder called 'photologue' under your `MEDIA_ROOT`).
3. Enable the admin app if you have not already done so.
4. Also enable Django's site framework.

1.4 Add the urls

Add photologue to your projects `urls.py` file:

```
urlpatterns += patterns('',
    ...
    url(r'^photologue/', include('photologue.urls', namespace='photologue')),
)
```


1.5 Sync Your Database

Database migrations can be managed either with South, or with Django's migrations module.

1.5.1 South

If you're on Django < 1.7, then you should be using [South](#). You'll need to add it to your `INSTALLED_APPS` and also add this to your settings:

```
SOUTH_MIGRATION_MODULES = {
    'photologue': 'photologue.south_migrations',
}
```

1.5.2 Django migrations

Starting with version 1.7, Django has a new migrations module - Photologue uses it out of the box.

1.5.3 Then...

In all cases, you can now sync your database:

```
python manage.py migrate photologue
```

If you are installing Photologue for the first time, this will set up some default PhotoSizes to get you started - you are free to change them of course!

1.6 Instant templates

Photologue comes with basic templates for galleries and photos, which are designed to work well with [Twitter-Bootstrap](#). You can of course override them, or completely replace them. Note that all Photologue templates inherit from `photologue/root.html`, which itself just inherits from a site-wide `base.html` - you can change this to use a different base template.

1.7 Sitemap

The [Sitemaps protocol](#) allows a webmaster to inform search engines about URLs on a website that are available for crawling. Django comes with a high-level framework that makes generating sitemap XML files easy.

Install the sitemap application as per the [instructions in the django documentation](#), then edit your project's `urls.py` and add a reference to Photologue's Sitemap classes in order to included all the publicly-viewable Photologue pages:

```
...
from photologue.sitemaps import GallerySitemap, PhotoSitemap

sitemaps = {...
    'photologue_galleries': GallerySitemap,
    'photologue_photos': PhotoSitemap,
    ...
}
etc...
```

There are 2 sitemap classes, as in some cases you may want to have gallery pages, but no photo detail page (e.g. if all photos are displayed via a javascript lightbox).

1.8 Sites

Photologue supports [Django's site framework](#) since version 2.8. That means that each Gallery and each Photo can be displayed on one or more sites.

Please bear in mind that photos don't necessarily have to be assigned to the same sites as the gallery they're belonging to: each gallery will only display the photos that are on its site. When a gallery does not belong the current site but a single photo is, that photo is only accessible directly as the gallery won't be shown in the index.

Note: If you're upgrading from a version earlier than 2.8 you don't need to worry about the assignment of already existing objects to a site because a datamigration will assign all your objects to the current site automatically.

Note: This feature is switched off by default. [See here to enable it](#) and for more information.

1.9 Amazon S3

Photologue can use a custom file storage system, for example [Amazon's S3](#).

You will need to configure your Django project to use Amazon S3 for storing files; a full discussion of how to do this is outside the scope of this page.

However, there is a quick demo of using Photologue with S3 in the `example_project` directory; if you look at these files:

- `example_project/example_project/settings.py`
- `example_project/requirements.txt`

At the end of each file you will commented-out lines for configuring S3 functionality. These point to extra files stored under `example_project/example_storages/`. Uncomment these lines, run the example project, then study these files for inspiration! After that, setting up S3 will consist of (at minimum) the following steps:

1. Signup for Amazon AWS S3 at <http://aws.amazon.com/s3/>.
2. Create a Bucket on S3 to store your media and static files.
3. Set the environment variables:
 - `AWS_ACCESS_KEY_ID` - issued to your account by S3.
 - `AWS_SECRET_ACCESS_KEY` - issued to your account by S3.
 - `AWS_STORAGE_BUCKET_NAME` - name of your bucket on S3.
4. To copy your static files into your S3 Bucket, type `python manage.py collectstatic` in the `example_project` directory.

Note: This simple setup does not handle S3 regions.

Customisation: extending templates

Photologue comes with a set of basic templates to get you started quickly - you can of course replace them with your own. That said, it is possible to extend the basic templates in your own project and override various blocks, for example to add css classes. Often this will be enough.

The trick to extending the templates is not special to Photologue, it's used in other projects such as [Oscar](#).

First, set up your template configuration as so:

```
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.Loader',
    'django.template.loaders.app_directories.Loader',
)

from photologue import PHOTOLOGUE_APP_DIR
TEMPLATE_DIRS = (
    ...other template folders...,
    PHOTOLOGUE_APP_DIR
)
```

The `PHOTOLOGUE_APP_DIR` points to the directory above Photologue's normal templates directory. This means that `path/to/photologue/template.html` can also be reached via `templates/path/to/photologue/template.html`.

For example, to customise `photologue/gallery_list.html`, you can have an implementation like:

```
# Create your own photologue/gallery_list.html
{% extends "templates/photologue/gallery_list.html" %}
```

... we are now extending the built-in `gallery_list.html` and we can override the content blocks that we want to customise ...

Customisation: Settings

Photologue has several settings to customise behaviour.

3.1 PHOTOLOGUE_GALLERY_PAGINATE_BY

Default: 20

Number of galleries to display per page for GalleryListView.

Deprecated since version 2.8: Instead, override the view; [see here](#).

3.2 PHOTOLOGUE_PHOTO_PAGINATE_BY

Default: 20

Number of photos to display per page for PhotoListView.

Deprecated since version 2.8: Instead, override the view; [see here](#).

3.3 PHOTOLOGUE_GALLERY_LATEST_LIMIT

Default: None

Default limit for gallery.latest

3.4 PHOTOLOGUE_GALLERY_SAMPLE_SIZE

Default: 5

Number of random images from the gallery to display.

3.5 PHOTOLOGUE_IMAGE_FIELD_MAX_LENGTH

Default: 100

max_length setting for the ImageModel ImageField

3.6 PHOTOLOGUE_SAMPLE_IMAGE_PATH

Default: `os.path.join(os.path.dirname(__file__), 'res', 'sample.jpg')`

Path to sample image

3.7 PHOTOLOGUE_MAXBLOCK

Default: `256 * 2 ** 10`

Modify image file buffer size.

3.8 PHOTOLOGUE_DIR

Default: `'photologue'`

The relative path from your `MEDIA_ROOT` setting where Photologue will save image files. If your `MEDIA_ROOT` is set to `"/home/user/media"`, photologue will upload your images to `"/home/user/media/photologue"`

3.9 PHOTOLOGUE_PATH

Default: `None`

Look for user function to define file paths. Specifies a “callable” that takes a model instance and the original uploaded filename and returns a relative path from your `MEDIA_ROOT` that the file will be saved. This function can be set directly.

For example you could use the following code in a util module:

```
# myapp/utils.py:

import os

def get_image_path(instance, filename):
    return os.path.join('path', 'to', 'my', 'files', filename)
```

Then set in settings:

```
# settings.py:

from utils import get_image_path

PHOTOLOGUE_PATH = get_image_path
```

Or instead, pass a string path:

```
# settings.py:

PHOTOLOGUE_PATH = 'myapp.utils.get_image_path'
```

3.10 PHOTOLOGUE_MULTISITE

Default: `False`

Photologue can integrate galleries and photos with Django's [site framework](#). Default is for this feature to be switched off, as only a minority of Django projects will need it.

In this case, new galleries and photos are automatically linked to the current site (`SITE_ID = 1`). The Sites many-to-many field is hidden in the admin, as there is no need for a user to see it.

If the setting is `True`, the admin interface is slightly changed:

- The Sites many-to-many field is displayed on Gallery and Photos models.
- The Gallery Upload allows you to associate one more sites to the uploaded photos (and gallery).

Note: Gallery Uploads (zip archives) are always associated with the current site. Pull requests to fix this would be welcome!

3.11 PHOTOLOGUE_ENABLE_TAGS

Default: `False`

Deprecated since version 3.0.

Photologue used to include tagging on both the Gallery and Photo models. This relied on the 3rd party [django-tagging library](#), which is no longer maintained.

As a consequence, tagging functionality is being removed from Photologue itself. Adding a 3rd party tagging library is pretty straightforward - [See here for an example](#).

As a first step, the tags are no longer accessible in the admin - this is to make clear that they are being deprecated.

The models have not been changed - the tags are still there, and any data is preserved. You can *choose* to re-enable tags in the admin with the `PHOTOLOGUE_ENABLE_TAGS` setting.

This change was put in place to make it very clear that tags are going to be removed. You should make plans to migrate your tags to a new tagging library; tags will be removed entirely from django-photologue in version 3.1.

Customisation: Admin

The Photologue admin can easily be customised to your project's requirements. The technique described on this page is not specific to Photologue - it can be applied to any 3rd party library.

4.1 Create a customisation application

For clarity, it's best to put our customisation code in a new application; let's call it `photologue_custom`; create the application and add it to your `INSTALLED_APPS` setting.

4.2 Changing the admin

In the new `photologue_custom` application, create a new empty `admin.py` file. In this file we can replace the admin configuration supplied by Photologue, with a configuration specific to your project. For example:

```
from django import forms
from django.contrib import admin

from photologue.admin import GalleryAdmin as GalleryAdminDefault
from photologue.models import Gallery

class GalleryAdminForm(forms.ModelForm):
    """Users never need to enter a description on a gallery."""

    class Meta:
        model = Gallery
        exclude = ['description']

class GalleryAdmin(GalleryAdminDefault):
    form = GalleryAdminForm

admin.site.unregister(Gallery)
admin.site.register(Gallery, GalleryAdmin)
```

This snippet will define a new Gallery admin class based on Photologue's own. The only change we make is to exclude the `description` field from the change form.

We then unregister the default admin for the Gallery model and replace it with our new class.

4.3 Possible uses

The technique outlined above can be used to make many changes to the admin; here are a couple of suggestions.

4.3.1 Custom rich text editors

The description field on the Gallery model (and the caption field on the Photo model) are plain text fields. With the above technique, it's easy to use a rich text editor to manage these fields in the admin. For example, if you have `django-ckeditor` installed:

```
from django import forms
from django.contrib import admin

from ckeditor.widgets import CKEditorWidget
from photologue.admin import GalleryAdmin as GalleryAdminDefault
from photologue.models import Gallery

class GalleryAdminForm(forms.ModelForm):
    """Replace the default description field, with one that uses a custom widget."""

    description = forms.CharField(widget=CKEditorWidget())

    class Meta:
        model = Gallery
        exclude = ['']

class GalleryAdmin(GalleryAdminDefault):
    form = GalleryAdminForm

admin.site.unregister(Gallery)
admin.site.register(Gallery, GalleryAdmin)
```

Customisation: Views and Urls

The photologue views and urls can be tweaked to better suit your project. The technique described on this page is not specific to Photologue - it can be applied to any 3rd party library.

5.1 Create a customisation application

For clarity, it's best to put our customisation code in a new application; let's call it `photologue_custom`; create the application and add it to your `INSTALLED_APPS` setting.

We will also want to customise urls:

1. Create a `urls.py` that will contain our customised urls:

```
from django.conf.urls import *

urlpatterns = patterns('',

    # Nothing to see here... for now.

)
```

2. These custom urls will override the main Photologue urls, so place them just before Photologue in the project's main `urls.py` file:

```
... other code
(r'^photologue/', include('photologue_custom.urls')),
url(r'^photologue/', include('photologue.urls', namespace='photologue')),
... other code
```

Now we're ready to make some changes.

5.2 Changing pagination from our new `urls.py`

The list pages of Photologue (both Gallery and Photo) display 20 objects per page. Let's change this value. Edit our new `urls.py` file, and add:

```
from django.conf.urls import *

from photologue.views import GalleryListView
```

```
urlpatterns = patterns('',
    url(r'^gallerylist/$',
        GalleryListView.as_view(paginate_by=5), name='photologue_custom-gallery-l
    )
```

We've copied the urlpattern for the gallery list view from Photologue itself, and changed it slightly by passing in `paginate_by=5`.

And that's it - now when that page is requested, our customised `urls.py` will be called first, with pagination set to 5 items.

5.3 Values that can be overridden from `urls.py`

5.3.1 GalleryListView

- `paginate_by`: number of items to display per page.

5.3.2 PhotoListView

- `paginate_by`: number of items to display per page.

5.4 Changing `views.py` to create a RESTful api

More substantial customisation can be carried out by writing custom views. For example, it's easy to change a Photologue view to return JSON objects rather than html webpages. For this quick demo, we'll use the [django-braces library](#) to override the view returning a list of all photos.

Add the following code to `views.py` in `photologue_custom`:

```
from photologue.views import PhotoListView

from braces.views import JSONResponseMixin

class PhotoJSONListView(JSONResponseMixin, PhotoListView):

    def render_to_response(self, context, **response_kwargs):
        return self.render_json_object_response(context['object_list'],
                                                **response_kwargs)
```

And call this new view from `urls.py`; here we are replacing the standard Photo list page provided by Photologue:

```
from .views import PhotoJSONListView

urlpatterns = patterns('',

    # Other urls...

    url(r'^photolist/$',
        PhotoJSONListView.as_view(),
        name='photologue_custom-photo-json-list'),
```

```
# Other urls as required...  
)
```

And that's it! Of course, this is simply a demo and a real RESTful api would be rather more complex.

Customisation: Models

The photologue models can be extended to better suit your project. The technique described on this page is not specific to Photologue - it can be applied to any 3rd party library.

The models within Photologue cannot be directly modified (unlike, for example, Django's own User model). There are a number of reasons behind this decision, including:

- If code within a project modifies directly the Photologue models' fields, it leaves the Photologue schema migrations in an ambiguous state.
- Likewise, model methods can no longer be trusted to behave as intended (as fields on which they depend may have been overridden).

However, it's easy to create new models linked by one-to-one relationships to Photologue's own `Gallery` and `Photo` models.

On this page we will show how you can add tags to the `Gallery` model. For this we will use the popular 3rd party application `django-taggit`.

Note: The `Gallery` and `Photo` models currently have tag fields, however these are based on the abandonware `django-tagging` application. Instead, tagging is being entirely removed from Photologue, as it is a non-core functionality of a gallery application, and is easy to add back in - as this page shows!

6.1 Create a customisation application

For clarity, it's best to put our customisation code in a new application; let's call it `photologue_custom`; create the application and add it to your `INSTALLED_APPS` setting.

6.2 Extending

Within the `photologue_custom` application, we will edit 2 files:

6.2.1 Models.py

```
from django.db import models

from taggit.managers import TaggableManager
```

```
from photologue.models import Gallery

class GalleryExtended(models.Model):

    # Link back to Photologue's Gallery model.
    gallery = models.OneToOneField(Gallery, related_name='extended')

    # This is the important bit - where we add in the tags.
    tags = TaggableManager(blank=True)

    # Boilerplate code to make a prettier display in the admin interface.
    class Meta:
        verbose_name = u'Extra fields'
        verbose_name_plural = u'Extra fields'

    def __str__(self):
        return self.gallery.title
```

6.2.2 Admin.py

```
from django.contrib import admin

from photologue.admin import GalleryAdmin as GalleryAdminDefault
from photologue.models import Gallery
from .models import GalleryExtended

class GalleryExtendedInline(admin.StackedInline):
    model = GalleryExtended
    can_delete = False

class GalleryAdmin(GalleryAdminDefault):

    """Define our new one-to-one model as an inline of Photologue's Gallery model."""

    inlines = [GalleryExtendedInline, ]

admin.site.unregister(Gallery)
admin.site.register(Gallery, GalleryAdmin)
```

The above code is enough to start entering tags in the admin interface. To use/display them in the front end, you will also need to override Photologue's own templates - as the templates are likely to be heavily customised for your specific project, an example is not included here.

Customisation: third-party contributions

Photologue has a ‘contrib’ folder that includes some useful tweaks to the base project. At the moment, we have just one contribution:

7.1 Old-style templates

Replaces the normal templates with the templates that used to come with Photologue 2.X. Use these if you have an existing project that extends these ‘old-style’ templates.

To use these, edit your `TEMPLATE_DIRS` setting:

```
from photologue import PHOTOLOGUE_APP_DIR
TEMPLATE_DIRS = (
    ...
    os.path.join(PHOLOGUE_TEMPLATE_DIR, 'contrib/old_style_templates'),
    ... other folders containing Photologue templates should come after...
)
```

Contributing to Photologue

Contributions are always very welcome. Even if you have never contributed to an open-source project before - please do not hesitate to offer help. Fixes for typos in the documentation, extra unit tests, etc... are welcome. And look in the issues list for anything tagged “easy_win”.

8.1 Example project

Photologue includes an example project under `/example_project/` to get you quickly ready for contributing to the project - do not hesitate to use it! Please refer to `/example_project/README.rst` for installation instructions.

You’ll probably also want to manually install [Sphinx](#) if you’re going to update the documentation.

8.2 Workflow

Photologue is hosted on Github, so if you have not already done so, read the excellent [Github help pages](#). We try to keep the workflow as simple as possible; most pull requests are merged straight into the master branch. Please ensure your pull requests are on separate branches, and please try to only include one new feature per pull request!

Features that will take a while to develop might warrant a separate branch in the project; at present only the ImageKit integration project is run on a separate branch.

8.3 Coding style

No surprises here - just try to [follow the conventions used by Django itself](#).

8.4 Unit tests

Including unit tests with your contributions will earn you bonus points, maybe even a beer. So write plenty of tests, and run them from the `/example_project/` with a `python manage.py test photologue`.

8.5 Documentation

Keeping the documentation up-to-date is very important - so if your code changes how Photologue works, or adds a new feature, please check that the documentation is still accurate, and update it if required.

We use [Sphinx](#) to prepare the documentation; please refer to the excellent docs on that site for help.

Note: The CHANGELOG is part of the documentation, so if your patch needs the end user to do something - e.g. run a South migration - don't forget to update it!

8.6 Translations

Photologue manages string translations with [Transifex](#). The easiest way to help is to add new/updated translations there.

Once you've added translations, give the maintainer a wave and he will pull the updated translations into the master branch, so that you can install Photologue directly from the Github repository (see [Installation](#)) and use your translations straight away. Or you can do nothing - just before a release any new/updated translations get pulled from Transifex and added to the Photologue project.

8.7 New features

In the wiki there is a [wishlist of new features already planned for Photologue](#) - you are welcome to suggest other useful improvements.

If you're interested in developing a new feature, it is recommended that you first discuss it on the [mailing list](#) or open a new ticket in Github, in order to avoid working on a feature that will not get accepted as it is judged to not fit in with the goals of Photologue.

8.7.1 A bit of history

Photologue was started by Justin Driscoll in 2007. He quickly built it into a powerful photo gallery and image processing application, and it became successful.

Justin then moved onto other projects, and no longer had the time required to maintain Photologue - there was only one commit between August 2009 and August 2012, and approximately 70 open tickets on the Google Code project page.

At this point Richard Barran took over as maintainer of the project. First priority was to improve the infrastructure of the project: moving to Github, adding South, Sphinx for documentation, Transifex for translations, Travis for continuous integration, zest.releaser.

The codebase has not changed much so far - and it needs quite a bit of TLC (Tender Loving Care), and new features are waiting to be added. This is where you step in...

8.8 And finally...

Please remember that the maintainer looks after Photologue in his spare time - so it might be a few weeks before your pull request gets looked at... and the pull requests that are nicely formatted, with code, tests and docs included, will always get reviewed first ;-)

Changelog

9.1 3.1 (2014-11-03)

- The 'zip upload' functionality has been moved to a custom admin page.
- Refactor `add_accessor_methods` to be lazily applied (see #110).
- Updated German translation.
- Several improvements to the sample Bootstrap templates.
- Support CACHEDIR.TAG spec issue #89
- Fix issue #99 by adding 10 extra char to photo title(max gallery size up to 999999999 images)
- Sitemap.xml was not aware of Sites (#104).
- In python 3, gallery upload would crash if uploaded file was not a zip file (#106).

9.2 3.0.2 (2014-09-23)

- Updated django-sortedm2m to an official release.
- Updated Spanish translation.
- Updated Bootstrap version used in example project.

9.3 3.0.1 (2014-09-16)

- Missed out some templates from the released package.

9.4 3.0 (2014-09-15)

Upgrade notes:

WARNING: IF YOU'RE USING POSTGRESQL AS A DATABASE & DJANGO 1.7, THE LATEST RELEASE OF DJANGO-SORTEDM2M HAS A BUG. INSTEAD, YOU'LL HAVE TO MANUALLY INSTALL:

`pip install -e git://github.com/richardbarran/django-sortedm2m.git@9a609a1c6b790a40a016e4ceadedbb6dd6b92010#egg=sortedm2m`

THE FOLLOWING CHANGES BREAK BACKWARDS COMPATIBILITY!

- Django 1.7 comes with a new migrations framework which replaces South - if you continue to use Django 1.6, you'll need to add new settings. Please refer in the docs to the installation instructions. If you're upgrading to Django 1.7 - upgrade Photologue first, THEN upgrade Django.
- The Twitter-Bootstrap templates - previously in 'contrib' - become the default; the previous templates are moved to 'contrib'.
- The django-tagging library is no longer maintained by its author. As a consequence, it has been disabled - see the docs for more information (page <https://django-photologue.readthedocs.org/en/latest/pages/customising/settings.html#photologue-enable-tags>)
- Support for Django 1.4 and 1.5 has been dropped (Photologue depends on django-sortedm2m, which has dropped support for 1.4; and Django 1.5 is no longer supported).
- PHOTOLOGUE_USE_CKEDITOR has been removed.
- Many urls have been renamed; photologue urls now go into their own namespace. See the urls.py file for all the changes.

Other changes:

- Support for Amazon S3 to store images (thank you Celia Oakley!).
- List views have changed urls: instead of /page/<n>/, we now have a /?page=<n> pattern. This is a more common style, and allows us to simplify template code e.g. paginators!
- date_taken field not correctly handled during single photo upload (#80).
- Removed deprecated PhotologueSitemap.
- Gallery zip uploads would fail if the title contained unicode characters.
- Gallery-uploads: Do not require title for uploading to existing gallery (#98).
- The Photologue urls used to use names for months; this has been changed to using numbers, which is better for non-English websites (#101).

9.5 2.8.3 (2014-08-28)

- Updated Spanish translation.

9.6 2.8.2 (2014-07-26)

- The latest release of django-sortedm2m is not compatible with older versions of Django, so don't use it (issue #92).

9.7 2.8.1 (2014-07-26)

- Fixed issue #94 (problem with i18n plural forms).
- Updated Slovak translation.

9.8 2.8 (2014-05-04)

Upgrade notes:

1. Photologue now depends on django-sortedm2m and django-model-utils - please refer to installation instructions. These dependencies should be added automatically.
2. Run South migrations.

List of changes:

- Photo and Gallery models now support Django's sites framework.
- Photologue now uses django-sortedm2m to sort photos in a gallery.
- Major rewrite of zip archive uploader: warn users of files that could not be processed, get code to work with Python 3 (issue #71), add extra error handling.
- Renamed field title_slug to slug - this allows us to simplify views.py a bit.
- PHOTOLOGUE_USE_CKEDITOR, PHOTOLOGUE_GALLERY_PAGINATE_BY and PHOTOLOGUE_PHOTO_PAGINATE_BY are deprecated.
- Fixed pagination controls for photo list template.
- Tightened naming rules for Photosize names.
- Fixed a couple of unicode-related bugs.
- Added to the documentation pages describing how to customise the admin and the views.
- Refactored slightly views.py.
- Started work on chainable querysets.

9.9 2.7 (2013-10-27)

Upgrade notes:

1. All settings are now prefixed with PHOTOLOGUE_. Please check that you are not affected by this.

List of changes:

- Fixed issue #56, Gallery pagination is broken.
- Photologue now works with Python 3.
- Added a set of templates that work well with Twitter-Bootstrap 3, and used them for the 'example_project'.
- Fixed issue #64 (allow installation without installing Pillow).
- Optional use of CKEditor.
- Updated/new translations for Polish, Slovak and German.
- Bugfix: allow viewing latest galleries/latest photos pages even if they are empty.
- Started using factory-boy - makes unit tests a bit easier to read.
- Added settings to customise pagination count on list pages.
- Documented all settings.
- All settings are now prefixed with PHOTOLOGUE_.

9.10 2.6.1 (2013-05-19)

List of changes:

- Fixed broken packaging in release 2.6.

9.11 2.6 (2013-05-19)

Upgrade notes:

1. Photologue now relies on Pillow instead of PIL. The easiest way to upgrade is to remove PIL completely, then install the new version of Photologue.
2. Photologue, in line with Django itself, has dropped support for Django 1.3.

List of changes:

- Switched from PIL to Pillow - hopefully this should make installation easier.
- Initial setup of data: removed plinit and replaced it with a South data migration.
- Added feature to allow extending the built-in templates (and documented it!).
- Allow editing of Photo added date (temp way of sorting photos).
- Added an example project to help people wanting to contribute to the project.
- Fixed buggy Travis CI script.
- fixed issue #52, transactions in migration
- fixed issue #51, uniqueness collisions in migration
- Accessing the root url (usually /photologue/ will now redirect you to the gallery list view.
- Photologue requires min. Django 1.4.
- Tidied a data validator on PhotoSizes.

9.12 2.5 (2012-12-13)

- added a sitemap.xml.
- added some templatetags.
- started using Sphinx for managing documentation.
- started using Transifex for managing translations.
- started using Travis CI.
- added 12 new translations and improved some of the existing translations.
- fixed issue #29 (quote URL of resized image properly).
- misc improvements to clarity of unit tests.
- added Django 1.4 timezone support.

9.13 2.4 (2012-08-13)

Upgrade notes:

1. Starting with this version, Photologue uses South to manage the database schema. If you are upgrading an existing Photologue installation, please follow the South instructions at: <http://south.readthedocs.org/en/latest/convertinganapp.html#converting-other-installations-and-servers>
2. Photologue has dropped support for Django 1.2.

List of changes:

- use South to manage schema changes.
- updated installation instructions.
- fixed issue #9 (In Django 1.3, FileField no longer deletes files).
- switched from function-based generic views to class-based views.
- fixed PendingDeprecationWarnings seen when running Django 1.3 - this will make the move to Django 1.5 easier.
- added unit tests.
- fixed bug where GALLERY_SAMPLE_SIZE setting was not being used.
- fixed issue #11 (GalleryUpload with len(title) > 50 causes a crash).
- fixed issue #10 (Increase the size of the name field for photosize).

Indices and tables

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

p

`photologue.sitemaps`, [5](#)

P

photologue.sitemaps (module), [5](#)