**Django Tagging**

A generic tagging application for [Django](http://www.djangoproject.com/) projects, which allows association of a number of tags with any Django model instance and makes retrieval of tags simple.

Contents

* [Installation](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "installation)
  + [Installing an official release](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "installing-an-official-release)
    - [Source distribution](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "source-distribution)
    - [Windows installer](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "windows-installer)
  + [Installing the development version](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "installing-the-development-version)
  + [Using Django Tagging in your applications](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "using-django-tagging-in-your-applications)
* [Settings](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "settings)
  + [FORCE\_LOWERCASE\_TAGS](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "force-lowercase-tags)
  + [MAX\_TAG\_LENGTH](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "max-tag-length)
* [Tags](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tags)
  + [API reference](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "api-reference)
    - [Fields](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "fields)
    - [Manager functions](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "manager-functions)
  + [Basic usage](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "basic-usage)
    - [Tag input](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tag-input)
    - [Tagging objects and retrieving an object's tags](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tagging-objects-and-retrieving-an-object-s-tags)
    - [Retrieving tags used by a particular model](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "retrieving-tags-used-by-a-particular-model)
* [Tagged items](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tagged-items)
  + [API reference](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id1)
    - [Fields](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id2)
    - [Manager functions](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id3)
  + [Basic usage](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id4)
    - [Retrieving tagged objects](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "retrieving-tagged-objects)
* [Utilities](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "utilities)
  + [parse\_tag\_input(input)](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "parse-tag-input-input)
  + [edit\_string\_for\_tags(tags)](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "edit-string-for-tags-tags)
  + [get\_tag\_list(tags)](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "get-tag-list-tags)
  + [calculate\_cloud(tags,](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "calculate-cloud-tags-steps-4-distribution-tagging-utils-logarithmic) steps=4, distribution=tagging.utils.LOGARITHMIC)
* [Model Fields](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "model-fields)
  + [Field types](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "field-types)
    - [TagField](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tagfield)
* [Form fields](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "form-fields)
  + [Field types](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id5)
    - [TagField](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id6)
* [Simplified tagging and retrieval of tags with properties](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "simplified-tagging-and-retrieval-of-tags-with-properties)
* [Generic views](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "generic-views)
  + [tagging.views.tagged\_object\_list](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tagging-views-tagged-object-list)
    - [Example usage](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "example-usage)
* [Template tags](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "template-tags)
  + [Tag reference](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tag-reference)
    - [tags\_for\_model](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tags-for-model)
    - [tag\_cloud\_for\_model](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tag-cloud-for-model)
    - [tags\_for\_object](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tags-for-object)
    - [tagged\_objects](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tagged-objects)

**[Installation](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id7)**

**[Installing an official release](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id8)**

Official releases are made available from <http://code.google.com/p/django-tagging/>

**[Source distribution](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id9)**

Download the .zip distribution file and unpack it. Inside is a script named setup.py. Enter this command:

python setup.py install

...and the package will install automatically.

**[Windows installer](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id10)**

A Windows installer is also made available - download the .exe distribution file and launch it to install the application.

An uninstaller will also be created, accessible through Add/Remove Programs in your Control Panel.

**[Installing the development version](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id11)**

Alternatively, if you'd like to update Django Tagging occasionally to pick up the latest bug fixes and enhancements before they make it into an offical release, perform a [Subversion](http://subversion.tigris.org/) checkout instead. The following command will check the application's development branch out to an tagging-trunk directory:

svn checkout http://django-tagging.googlecode.com/svn/trunk/ tagging-trunk

Add the resulting folder to your [PYTHONPATH](http://docs.python.org/tut/node8.html" \l "SECTION008110000000000000000) or symlink ([junction](http://www.microsoft.com/technet/sysinternals/FileAndDisk/Junction.mspx), if you're on Windows) the tagging directory inside it into a directory which is on your PYTHONPATH, such as your Python installation's site-packages directory.

You can verify that the application is available on your PYTHONPATH by opening a Python interpreter and entering the following commands:

>>> import tagging

>>> tagging.VERSION

(0, 2, 'pre')

When you want to update your copy of the Django Tagging source code, run the command svn update from within the tagging-trunk directory.

Caution!

The development version may contain bugs which are not present in the release version and introduce backwards-incompatible changes.

If you're tracking trunk, keep an eye on the [CHANGELOG](http://django-tagging.googlecode.com/svn/trunk/CHANGELOG.txt) and the [backwards-incompatible changes wiki page](http://code.google.com/p/django-tagging/wiki/BackwardsIncompatibleChanges) before you update your copy of the source code.

**[Using Django Tagging in your applications](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id12)**

Once you've installed Django Tagging and want to use it in your Django applications, do the following:

1. Put 'tagging' in your INSTALLED\_APPS setting.
2. Run the command manage.py syncdb.

The syncdb command creates the necessary database tables and creates permission objects for all installed apps that need them.

That's it!

**[Settings](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id13)**

Some of Django Tagging's behaviour may be configured by adding the appropriate settings to your project's settings file.

The following settings are available:

**[FORCE\_LOWERCASE\_TAGS](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id14)**

Default: False

A boolean that turns on/off forcing of all tag names to lowercase before they are saved to the database.

**[MAX\_TAG\_LENGTH](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id15)**

Default: 50

An integer which specifies the maxiumum length which any tag is allowed to have. This is used for validation in the django.contrib.admin application and in any newforms forms automatically generated using ModelForm.

**[Tags](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id16)**

Tags are represented by the Tag model, which lives in the tagging.models module.

**[API reference](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id17)**

**[Fields](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id18)**

Tag objects have the following fields:

* name -- The name of the tag. This is a unique value.

**[Manager functions](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id19)**

The Tag model has a custom manager which has the following helper functions:

* update\_tags(obj, tag\_names) -- updates tags associated with an object.

tag\_names is a string containing tag names with which obj should be tagged.

If tag\_names is None or '', the object's tags will be cleared.

* add\_tag(obj, tag\_name) -- associates a tag with an an object.

tag\_name is a string containing a tag name with which obj should be tagged.

* get\_for\_object(obj) -- returns a QuerySet containing all Tag objects associated with obj.
* usage\_for\_model(Model, counts=False, min\_count=None, filters=None) -- returns a list of Tag objects associated with instances of Model.

If counts is True, a count attribute will be added to each tag, indicating how many times it has been associated with instances of Model.

If min\_count is given, only tags which have a count greater than or equal to min\_count will be returned. Passing a value for min\_count implies counts=True.

To limit the tags (and counts, if specified) returned to those used by a subset of the model's instances, pass a dictionary of field lookups to be applied to Model as the filters argument.

* related\_for\_model(tags, Model, counts=False, min\_count=None) -- returns a list of tags related to a given list of tags - that is, other tags used by items which have all the given tags.

If counts is True, a count attribute will be added to each tag, indicating the number of items which have it in addition to the given list of tags.

If min\_count is given, only tags which have a count greater than or equal to min\_count will be returned. Passing a value for min\_count implies counts=True.

* cloud\_for\_model(Model, steps=4, distribution=LOGARITHMIC, filters=None, min\_count=None) -- returns a list of the distinct Tag objects associated with instances of Model, each having a count attribute as above and an additional font\_size attribute, for use in creation of a tag cloud (a type of weighted list).

steps defines the number of font sizes available - font\_size may be an integer between 1 and steps, inclusive.

distribution defines the type of font size distribution algorithm which will be used - logarithmic or linear. It must be either tagging.utils.LOGARITHMIC or tagging.utils.LINEAR.

To limit the tags displayed in the cloud to those associated with a subset of the Model's instances, pass a dictionary of field lookups to be applied to the given Model as the filters argument.

To limit the tags displayed in the cloud to those with a count greater than or equal to min\_count, pass a value for the min\_count argument.

**[Basic usage](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id20)**

**[Tag input](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id21)**

Tag input from users is treated as follows:

* If the tag input doesn't contain any commas or double quotes, it is simply treated as a space-delimited list of tag names.
* If the tag input does contain either of these characters, we parse the input like so:
  + Groups of characters which appear between double quotes take precedence as multi-word tags (so double quoted tag names may contain commas). An unclosed double quote will be ignored.
  + For the remaining input, if there are any unquoted commas in the input, the remainder will be treated as comma-delimited. Otherwise, it will be treated as space-delimited.

Examples:

| **Tag input** | **Resulting tag names** | **Notes** |
| --- | --- | --- |
| apple ball cat | [apple], [ball], [cat] | No commas or quotes, so space delimited |
| apple, ball cat | [apple], [ball cat] | Comma present, so comma delimited |
| "apple, ball" cat dog | [apple, ball], [cat], [dog] | All commas are quoted, so space delimited |
| "apple, ball", cat dog | [apple, ball], [cat dog] | Contains an unquoted comma, so comma delimited |
| apple "ball cat" dog | [apple], [ball cat], [dog] | No commas, so space delimited |
| "apple" "ball dog | [apple], [ball], [dog] | Unclosed double quote is ignored |

**[Tagging objects and retrieving an object's tags](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id22)**

Objects may be tagged using the update\_tags helper function:

>>> from shop.apps.products.models import Widget

>>> from tagging.models import Tag

>>> widget = Widget.objects.get(pk=1)

>>> Tag.objects.update\_tags(widget, 'house thing')

Retrieve tags for an object using the get\_for\_object helper function:

>>> Tag.objects.get\_for\_object(widget)

[<Tag: house>, <Tag: thing>]

Tags are created, associated and unassociated accordingly when you use update\_tags and add\_tags:

>>> Tag.objects.update\_tags(widget, 'house monkey')

>>> Tag.objects.get\_for\_object(widget)

[<Tag: house>, <Tag: monkey>]

>>> Tag.objects.add\_tag(widget, 'tiles')

>>> Tag.objects.get\_for\_object(widget)

[<Tag: house>, <Tag: monkey>, <Tag: tiles>]

Clear an object's tags by passing None or '' to update\_tags:

>>> Tag.objects.update\_tags(widget, None)

>>> Tag.objects.get\_for\_object(widget)

[]

**[Retrieving tags used by a particular model](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id23)**

To retrieve all tags used for a particular model, use the get\_for\_model helper function:

>>> widget1 = Widget.objects.get(pk=1)

>>> Tag.objects.update\_tags(widget1, 'house thing')

>>> widget2 = Widget.objects.get(pk=2)

>>> Tag.objects.update\_tags(widget2, 'cheese toast house')

>>> Tag.objects.usage\_for\_model(Widget)

[<Tag: cheese>, <Tag: house>, <Tag: thing>, <Tag: toast>]

To get a count of how many times each tag was used for a particular model, pass in True for the counts argument:

>>> tags = Tag.objects.usage\_for\_model(Widget, counts=True)

>>> [(tag.name, tag.count) for tag in tags]

[('cheese', 1), ('house', 2), ('thing', 1), ('toast', 1)]

To get counts and limit the tags returned to those with counts above a certain size, pass in a min\_count argument:

>>> tags = Tag.objects.usage\_for\_model(Widget, min\_count=2)

>>> [(tag.name, tag.count) for tag in tags]

[('house', 2)]

You can also specify a dictionary of [field lookups](http://www.djangoproject.com/documentation/db-api/" \l "field-lookups) to be used to restrict the tags and counts returned based on a subset of the model's instances. For example, the following would retrieve all tags used on Widgets created by a user named Alan which have a size greater than 99:

>>> Tag.objects.usage\_for\_model(Widget, filters=dict(size\_\_gt=99, user\_\_username='Alan'))

**[Tagged items](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id24)**

The relationship between a Tag and an object is represented by the TaggedItem model, which lives in the tagging.models module.

**[API reference](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id25)**

**[Fields](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id26)**

TaggedItem objects have the following fields:

* tag -- The Tag an object is associated with.
* content\_type -- The ContentType of the associated model instance.
* object\_id -- The id of the associated object.
* object -- The associated object itself, accessible via the Generic Relations API.

**[Manager functions](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id27)**

The TaggedItem model has a custom manager which has the following helper functions:

* get\_by\_model(Model, tag) -- If tag is an instance of a Tag, returns a QuerySet containing all instances of Model which are tagged with it.

If tag is a list of tags, returns a QuerySet containing all instances of Model which are tagged with every tag in the list.

* get\_intersection\_by\_model(Model, tags) -- Returns a QuerySet containing all instances of Model which are tagged with every tag in the list.

get\_by\_model will call this function behind the scenes when you pass it a list, so it's recommended that you use get\_by\_model instead of calling this function directly.

* get\_union\_by\_model(Model, tags) -- Returns a QuerySet containing all instances of Model which are tagged with any tag in the list.
* get\_related(obj, Model, num=None) - Returns instances of Model which share tags with the model instance obj, ordered by the number of shared tags in descending order.

If num is given, a maximum of num instances will be returned.

**[Basic usage](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id28)**

**[Retrieving tagged objects](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id29)**

Objects may be retrieved based on their tags using the get\_by\_model manager method:

>>> from shop.apps.products.models import Widget

>>> from tagging.models import Tag

>>> house\_tag = Tag.objects.get(name='house')

>>> TaggedItem.objects.get\_by\_model(Widget, house\_tag)

[<Widget: pk=1>, <Widget: pk=2>]

Passing a list of tags to get\_by\_model returns an intersection of objects which have those tags, i.e. tag1 AND tag2 ... AND tagN:

>>> thing\_tag = Tag.objects.get(name='thing')

>>> TaggedItem.objects.get\_by\_model(Widget, [house\_tag, thing\_tag])

[<Widget: pk=1>]

Functions which take tags are flexible when it comes to tag input:

>>> TaggedItem.objects.get\_by\_model(Widget, Tag.objects.filter(name\_\_in=['house', 'thing']))

[<Widget: pk=1>]

>>> TaggedItem.objects.get\_by\_model(Widget, 'house thing')

[<Widget: pk=1>]

>>> TaggedItem.objects.get\_by\_model(Widget, ['house', 'thing'])

[<Widget: pk=1>]

**[Utilities](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id30)**

Tag-related utility functions are defined in the tagging.utils module:

**[parse\_tag\_input(input)](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id31)**

Parses tag input, with multiple word input being activated and delineated by commas and double quotes. Quotes take precedence, so they may contain commas.

Returns a sorted list of unique tag names.

See [tag input](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "tag-input) for more details.

**[edit\_string\_for\_tags(tags)](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id32)**

Given list of Tag instances, creates a string representation of the list suitable for editing by the user, such that submitting the given string representation back without changing it will give the same list of tags.

Tag names which contain commas will be double quoted.

If any tag name which isn't being quoted contains whitespace, the resulting string of tag names will be comma-delimited, otherwise it will be space-delimited.

**[get\_tag\_list(tags)](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id33)**

Utility function for accepting tag input in a flexible manner.

If a Tag object is given, it will be returned in a list as its single occupant.

If given, the tag names in the following will be used to create a Tag QuerySet:

* A string, which may contain multiple tag names.
* A list or tuple of strings corresponding to tag names.
* A list or tuple of integers corresponding to tag ids.

If given, the following will be returned as-is:

* A list or tuple of Tag objects.
* A Tag QuerySet.

**[calculate\_cloud(tags,](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id34) steps=4, distribution=tagging.utils.LOGARITHMIC)**

Add a font\_size attribute to each tag according to the frequency of its use, as indicated by its count attribute.

steps defines the range of font sizes - font\_size will be an integer between 1 and steps (inclusive).

distribution defines the type of font size distribution algorithm which will be used - logarithmic or linear. It must be one of tagging.utils.LOGARITHMIC or tagging.utils.LINEAR.

**[Model Fields](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id35)**

The tagging.fields module contains fields which make it easy to integrate tagging into your models and into the django.contrib.admin application.

**[Field types](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id36)**

**[TagField](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id37)**

A CharField that actually works as a relationship to tags "under the hood".

Using this example model:

class Link(models.Model):

...

tags = TagField()

Setting tags:

>>> l = Link.objects.get(...)

>>> l.tags = 'tag1 tag2 tag3'

Getting tags for an instance:

>>> l.tags

'tag1 tag2 tag3'

Getting tags for a model - i.e. all tags used by all instances of the model:

>>> Link.tags

'tag1 tag2 tag3 tag4 tag5'

This field will also validate that it has been given a valid list of tag names, separated by a single comma, a single space or a comma followed by a space, using the isTagList validator from tagging.validators.

**[Form fields](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id38)**

The tagging.forms module contains a Field for use with Django's [newforms library](http://www.djangoproject.com/documentation/newforms/) which takes care of validating tag name input when used in your forms.

**[Field types](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id39)**

**[TagField](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id40)**

A form Field which is displayed as a single-line text input, which validates that the input it receives is a valid list of tag names.

When you generate a form for one of your models automatically, using the ModelForm class provided by newforms, any tagging.fields.TagField fields in your model will automatically be represented by a tagging.forms.TagField in the generated form.

**[Simplified tagging and retrieval of tags with properties](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id41)**

If you're not using TagField, a useful method for simplifying tagging and retrieval of tags for your models is to set up a property:

from django.db import models

from tagging.models import Tag

class MyModel(models.Model):

name = models.CharField(maxlength=100)

tag\_list = models.CharField(maxlength=255)

def save(self):

super(MyModel, self).save()

self.tags = self.tag\_list

def \_get\_tags(self):

return Tag.objects.get\_for\_object(self)

def \_set\_tags(self, tag\_list):

Tag.objects.update\_tags(self, tag\_list)

tags = property(\_get\_tags, \_set\_tags)

def \_\_unicode\_\_(self):

return self.name

Once you've set this up, you can access and set tags in a fairly natural way:

>>> obj = MyModel.objects.get(pk=1)

>>> obj.tags = 'foo bar'

>>> obj.tags

[<Tag: bar>, <Tag: foo>]

Remember that obj.tags will return a QuerySet, so you can perform further filtering on it, should you need to.

**[Generic views](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id42)**

The tagging.views module contains views to handle simple cases of common display logic related to tagging.

**[tagging.views.tagged\_object\_list](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id43)**

**Description:**

A view that displays a list of objects for a given model which have a given tag. This is a thin wrapper around the django.views.generic.list\_detail.object\_list view, which takes a model and a tag as its arguments (in addition to the other optional arguments supported by object\_list), building the appropriate QuerySet for you instead of expecting one to be passed in.

**Required arguments:**

* model: The Django model class of the object that will be listed.
* tag: The tag which objects of the given model must have in order to be listed.

**Optional arguments:**

Please refer to the [object\_list documentation](http://www.djangoproject.com/documentation/generic_views/" \l "django-views-generic-list-detail-object-list) for additional optional arguments which may be given.

* related\_tags: If True, a related\_tags context variable will also contain tags related to the given tag for the given model.
* related\_tag\_counts: If True and related\_tags is True, each related tag will have a count attribute indicating the number of items which have it in addition to the given tag.

**Template context:**

Please refer to the [object\_list documentation](http://www.djangoproject.com/documentation/generic_views/" \l "django-views-generic-list-detail-object-list) for additional template context variables which may be provided.

* tag: The Tag instance for the given tag.

**[Example usage](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id44)**

The following sample URLconf demonstrates using this generic view to list items of a particular model class which have a given tag:

from django.conf.urls.defaults import \*

from tagging.views import tagged\_object\_list

from shop.apps.products.models import Widget

urlpatterns = patterns('',

url(r'^widgets/tag/(?P<tag>[^/]+)/$',

tagged\_object\_list,

dict(model=Widget, paginate\_by=10, allow\_empty=True,

template\_object\_name='widget'),

name='widget\_tag\_detail'),

)

**[Template tags](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id45)**

The tagging.templatetags.tagging\_tags module defines a number of template tags which may be used to work with tags.

**[Tag reference](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id46)**

**[tags\_for\_model](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id47)**

Retrieves a list of Tag objects associated with a given model and stores them in a context variable.

Usage:

{% tags\_for\_model [model] as [varname] %}

The model is specified in [appname].[modelname] format.

Extended usage:

{% tags\_for\_model [model] as [varname] with counts %}

If specified - by providing extra with counts arguments - adds a count attribute to each tag containing the number of instances of the given model which have been tagged with it.

Examples:

{% tags\_for\_model products.Widget as widget\_tags %}

{% tags\_for\_model products.Widget as widget\_tags with counts %}

**[tag\_cloud\_for\_model](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id48)**

Retrieves a list of Tag objects for a given model, with tag cloud attributes set, and stores them in a context variable.

Usage:

{% tag\_cloud\_for\_model [model] as [varname] %}

The model is specified in [appname].[modelname] format.

Extended usage:

{% tag\_cloud\_for\_model [model] as [varname] with [options] %}

Extra options can be provided after an optional with argument, with each option being specified in [name]=[value] format. Valid extra options are:

steps

Integer. Defines the range of font sizes.

min\_count

Integer. Defines the minimum number of times a tag must have been used to appear in the cloud.

distribution

One of linear or log. Defines the font-size distribution algorithm to use when generating the tag cloud.

Examples:

{% tag\_cloud\_for\_model products.Widget as widget\_tags %}

{% tag\_cloud\_for\_model products.Widget as widget\_tags with steps=9 min\_count=3 distribution=log %}

**[tags\_for\_object](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id49)**

Retrieves a list of Tag objects associated with an object and stores them in a context variable.

Usage:

{% tags\_for\_object [object] as [varname] %}

Example:

{% tags\_for\_object foo\_object as tag\_list %}

**[tagged\_objects](file:///C:\Users\p7164437\AppData\Local\Temp\tagging-0.2-overview.html" \l "id50)**

Retrieves a list of instances of a given model which are tagged with a given Tag and stores them in a context variable.

Usage:

{% tagged\_objects [tag] in [model] as [varname] %}

The model is specified in [appname].[modelname] format.

The tag must be an instance of a Tag, not the name of a tag.

Example:

{% tagged\_objects comedy\_tag in tv.Show as comedies %}