# Django App Helper Documentation

***Release 1.4.1***

**Iacopo Spalletti**

**Jul 03, 2019**

# Contents

A pluggable app allows Django developers to quickly add meta tags and OpenGraph, Twitter, and Google Plus properties to their HTML responses.

CHAPTER 1

# Usage

django meta has two different operating mode:

- *Model support*
- *View support*

## 1.1 Installation

- Install using pip:

```
pip install django-meta
```

- Add `meta` to `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ....
    'meta',
)
```

- Add the *required namespace tags* to your base template for compliancy with metadata protocols.
- Optionally you can install and configure sekizai

## 1.2 Model support

### 1.2.1 Concepts

**django-meta** provides a mixin to handle metadata in your models.

Actual data are evaluated at runtime pulling values from model attributes and methods.

To use it, defines a `_metadata` attribute as a dictionary of tag/value pairs;

- **tag** is the name of the metatag as used by `meta.html` template
- **value** is a string that is evaluated in the following order:
  - model method name called with the meta attribute as argument
  - model method name called with no arguments
  - model attribute name (evaluated at runtime)
  - string literal (if none of the above exists)

If **value** is `False` or it is evaluated as `False` at runtime the tag is skipped.

To use this mixin you must invoke `as_meta()` on the model instance for example in the get_context_data().

#### Request

`as_meta()` accepts the `request` object that is saved locally and is available to methods by using the `get_request` method.

#### Public interface

`ModelMeta.get_meta(request=None)`: returns the metadata attributes definition. Tipically these are set in `_metadata` attribute in the model;

`ModelMeta.as_meta(request=None)`: returns the meta representation of the object suitable for use in the template;

`ModelMeta.get_request()`: returns the `request` object, if given as argument to `as_meta`;

`ModelMeta.get_author()`: returns the author object for the current instance. Default implementation does not return a valid object, this **must** be overidden in the application according to what is an author in the application domain;

`ModelMeta.build_absolute_uri(url)`: create an absolute URL (i.e.: complete with protocol and domain); this is generated from the `request` object, if given as argument to `as_meta`;

### 1.2.2 Usage

1. Configure `django-meta` according to documentation
2. Add meta information to your model:

```
from django.db import models
from meta.models import ModelMeta

class MyModel(ModelMeta, models.Model):
    name = models.CharField(max_length=20)
    abstract = models.TextField()
    image = models.ImageField()
    ...

    _metadata = {
        'title': 'name',
        'description': 'abstract',
        'image': 'get_meta_image',
        ...
    }
```

(continues on next page)

(continued from previous page)

```
    def get_meta_image(self):
        if self.image:
            return self.image.url
```

3. Push metadata in the context using `as_meta` method:

```
class MyView(DetailView):

    ...

    def get_context_data(self, **kwargs):
        context = super(MyView, self).get_context_data(self, **kwargs)
        context['meta'] = self.get_object().as_meta(self.request)
        return context
```

4. For Function Based View can just use `as_meta()` method for pass "meta" context variable:

```
def post(request, id):
    template = 'single_post.html'
    post = Post.objects.get(pk=id)
    context = {}
    context['post'] = post
    context['meta'] = post.as_meta()
    return render(request, template, context)
```

5. Include `meta/meta.html` template in your templates:

```
{% load meta %}

<html>
<head {% meta_namespaces %}>
    {% include "meta/meta.html" %}
</head>
<body>
</body>
</html>
```

#### Note

- For Google+ support you must add `{% render_block 'html_extra' %}` in your template to add object type definition. See relevant Google+ snippets documentation (https://developers.google.com/+/web/snippet/)
- For OpenGraph / Facebook support, edit your `<head>` tag to use `meta_namespaces` templatetags

### 1.2.3 Reference template

See below the basic reference template:

```
{% load meta %}

<html>
<head {% meta_namespaces %}>
    {% include "meta/meta.html" %}
</head>
```

(continues on next page)

(continued from previous page)

```
<body>
{% block content %}
{% endblock content %}
</body>
</html>
```

## 1.3 View support

### 1.3.1 Using the view

You render the meta tags by including a `meta.html` partial template in your view templates. This template will only render meta tags if it can find a `meta` object in the context, so you can safely include it in your base template to have it render on all your pages.

The `meta.html` template expects to find an object called `meta` in the template context which contains any of the following attributes:

- use_og
- use_twitter
- use_facebook
- use_googleplus
- use_title_tag
- title
- description
- keywords
- url
- image
- object_type
- site_name
- twitter_site
- twitter_creator
- twitter_card
- facebook_app_id
- locale
- extra_props
- extra_custom_props

In all cases, if the attribute is not set/empty, the matching metadata/property is not rendered.

#### Meta object

The core of django-meta is the `Meta` class. Although you can prepare the metadata for the template yourself, this class can make things somewhat easier.

To set up a meta object for use in templates, simply instantiate it with the properties you want to use:

```
from meta.views import Meta

meta = Meta(
    title="Sam's awesome ponies",
    description='Awesome page about ponies',
    keywords=['pony', 'ponies', 'awesome'],
    extra_props = {
        'viewport': 'width=device-width, initial-scale=1.0, minimum-scale=1.0'
    }
    'extra_custom_props': [
        ('http-equiv', 'Content-Type', 'text/html; charset=UTF-8'),
    ]
)
```

When the time comes to render the template, simply include the instance as `'meta'` context variable.

`Meta` also accept an (optional) `request` argument to pass the current request, which is used to retrieve the `SITE_ID` if it's not in the settings.

The `Meta` instances have the same properties as the keys listed in the *Using the view* section. For convenience, some of the properties are 'smart', and will modify values you set. These properties are:

- keywords
- url
- image

For brevity, we will only discuss those here.

### 1.3.2 Meta.keywords

When you assign keywords either via the constructor, or by assigning an iterable to the `keywords` property, it will be cleaned up of all duplicates and returned as a `set`. If you have specified the *META_INCLUDE_KEYWORDS*, the resulting set will also include them. If you omit this argument when instantiating the object, or if you assign `None` to the `keywords` property, keywords defined by *META_DEFAULT_KEYWORDS* setting will be used instead.

### 1.3.3 Meta.url

Setting the url behaves differently depending on whether you are passsing a path or a full URL. If your URL starts with `'http'`, it will be used verbatim (not that the actual validity of the url is not checked so `'httpfoo'` will be considered a valid URL). If you use an absolute or relative path, domain and protocol parts would be prepended to the URL. Here's an example:

```
m = Meta(url='/foo/bar')
m.url  # returns 'http://example.com/foo/bar'
```

The actual protocol and domain are dependent on the *META_SITE_PROTOCOL* and *META_SITE_DOMAIN* settings. If you wish to use the Django's sites contrib app to calculate the domain, you can either set the *META_USE_SITES* setting to `True`, or pass the `use_sites` argument to the constructor:

```
m = Meta(url='/foo/bar', use_sites=True)
```

Note that using the sites app will trigger database queries and/or cache hits, and it is therefore disabled by default.

### 1.3.4 Meta.image

The `image` property behaves the same way as `url` property with one notable difference. This property treats absolute and relative paths differently. It will place relative paths under the *META_IMAGE_URL*.

#### View mixin

As a convenience to those who embrace the Django's class-based views, django-meta includes a mixin that can be used with your views. Using the mixin is very simple:

```
from django.views.generic import View

from meta.views import MetadataMixin


class MyView(MetadataMixin, View):
    title = 'Some page'
    description = 'This is an awesome page'
    image = 'img/some_page_thumb.gif'
    url = 'some/page/'

    ....
```

The mixin sports all properties listed in the *Using the view* section with a few additional bells and whistles that make working with them easier. The mixin will return an instance of the `Meta` class (see *Meta object*) as `meta` context variable. This is, in turn, used in the partial template to render the meta tags (see *Rendering meta tags*).

Each of the properties on the mixin can be calculated dynamically by using the `MetadataMixin.get_meta_PROPERTYNAME` methods, where `PROPERTYNAME` is the name of the property you wish the calculate at runtime. Each method will receive a `context` keyword argument containig the request context.

For example, to calculate the description dynamically, you may use the mixin like so:

```
class MyView(MetadataMixin, SingleObjectMixin, View):
    ...

    def get_meta_description(self, context):
        return self.get_object().description
```

There are two more methods that you can overload in your view classes, and those are `get_domain` and `get_protocol`.

#### Reference template

See below the basic reference template:

```
{% load sekizai_tags meta %}

<html {% render_block 'html_extra' %}>
<head {% meta_namespaces %}>
```

(continues on next page)

(continued from previous page)

```
    {{ meta.og_description }}
    {% include "meta/meta.html" %}
</head>
<body>
{% block content %}
{% endblock content %}
</body>
</html>
```

#### Properties

### 1.3.5 use_og

This key contains a boolean value, and instructs the template to render the OpenGraph properties. These are usually used by FaceBook to get more information about your site's pages.

### 1.3.6 use_twitter

This key contains a boolean value, and instructs the template to render the Twitter properties. These are usually used by Twitter to get more information about your site's pages.

### 1.3.7 use_facebook

This key contains a boolean value, and instructs the template to render the Facebook properties. These are usually used by Facebook to get more information about your site's pages.

### 1.3.8 use_googleplus

This key contains a boolean value, and instructs the template to render the Google+. These are usually used by Google to get more information about your site's pages.

### 1.3.9 use_title_tag

This key contains a boolean value, and instructs the template to render the `<title></title>` tag. In the simple case, you use `<title></title>` tag in the templates where you can override it, but if you want to generate it dynamically in the views, you can set this property to `True`.

### 1.3.10 title

This key is used in the `og:title` OpenGraph property if `use_og` is `True`, `twitter:title` if `use_twitter` is `True`, `itemprop="title"` if `use_googleplus` is `True` or `<title></title>` tag if `use_title_tag` is `True`.

### 1.3.11 description

This key is used to render the `description` meta tag as well as the `og:description` and `twitter:description` property.

### 1.3.12 keywords

This key should be an iterable containing the keywords for the page. It is used to render the `keywords` meta tag.

### 1.3.13 url

This key should be the *full* URL of the page. It is used to render the `og:url`, `twitter:url`, `itemprop=url` property.

### 1.3.14 image

This key should be the *full* URL of an image to be used with the `og:image`, `twitter:image`, `itemprop=mage` property.

### 1.3.15 object_type

This key is used to render the `og:type` property.

### 1.3.16 site_name

This key is used to render the `og:site_name` property.

### 1.3.17 twitter_site

This key is used to render the `twitter:site` property.

### 1.3.18 twitter_creator

This key is used to render the `twitter:creator` property.

### 1.3.19 twitter_card

This key is used to render the `twitter:card` property.

### 1.3.20 facebook_app_id

This key is used to render the `fb:app_id` property.

### 1.3.21 locale

This key is used to render the `og:locale` property.

### 1.3.22 extra_props

A dictionary of extra optional properties:

```
{
    'foo': 'bar',
    'key': 'value'
}

...

<meta name="foo" content="bar">
<meta name="key" content="value">
```

### 1.3.23 extra_custom_props

A list of tuples for rendering custom extra properties:

```
[
    ('key', 'foo', 'bar')
    ('property', 'name', 'value')
]

...

<meta name="foo" content="bar">
<meta property="name" content="value">
```

## 1.4 Settings

django-meta has a few configuration options that allow you to customize it. Two of them are required: `META_SITE_PROTOCOL` and `META_SITE_DOMAIN`. By default, if they are unset, an `ImproperlyConfigured` exception will be raised when dealing with `url` and `image` properties. You can either set them, or overload the `Meta` class' `get_domain` and `get_protocol` methods (see *Meta object* section).

### 1.4.1 META_SITE_PROTOCOL

Defines the protocol used on your site. This should be set to either `'http'` or `'https'`. Default is `None`.

### 1.4.2 META_SITE_DOMAIN

Domain of your site. The `Meta` objects can also be made to use the Django's Sites framework as well (see *Meta object* and *META_USE_SITES* sections). Default is `None`.

### 1.4.3 META_SITE_TYPE

The default `og:type` property to use site-wide. You do not need to set this if you do not intend to use the OpenGraph properties. Default is `None`.

### 1.4.4 META_SITE_NAME

The site name to use in `og:site_name` property. Althoug this can be set per view, we recommend you set it globally. Defalt is `None`.

### 1.4.5 META_INCLUDE_KEYWORDS

Iterable of extra keywords to include in every view. These keywords are appended to whatever keywords you specify for the view, but are not used at all if no keywords are specified for the view. See *META_DEFAULT_KEYWORDS* if you wish to specify keywords to be used when no keywords are supplied. Default is `[]`.

### 1.4.6 META_DEFAULT_KEYWORDS

Iterable of default keywords to use when no keywords are specified for the view. These keywords are not included if you specify keywords for the view. If you need keywords that will always be present, regardless of whether you've specified any other keywords for the view or not, you need to combine this setting with *META_INCLUDE_KEYWORDS* setting. Default is `[]`.

### 1.4.7 META_IMAGE_URL

This setting is used as the base URL for all image assets that you intend to use as `og:image` property in your views. This is django-meta's counterpart of the Django's `STATIC_URL` setting. In fact, Django's `STATIC_URL` setting is a fallback if you do not specify this setting, so make sure either one is configured. Default is to use the `STATIC_URL` setting.

Note that you must add the trailing slash when specifying the URL. Even if you do not intend to use the `og:image` property, you need to define either this setting or the `STATIC_URL` setting or an attribute error will be raised.

### 1.4.8 META_USE_OG_PROPERTIES

This setting tells django-meta whether to render the OpenGraph properties. Default is `False`.

### 1.4.9 META_USE_TWITTER_PROPERTIES

This setting tells django-meta whether to render the Twitter properties. Default is `False`.

### 1.4.10 META_USE_GOOGLEPLUS_PROPERTIES

This setting tells django-meta whether to render the Google properties. Default is `False`.

### 1.4.11 META_USE_TITLE_TAG

This setting tells django-meta whether to render the `<title></title>` tag. Default is `False`.

### 1.4.12 META_USE_SITES

This setting tells django-meta to derive the site's domain using the Django's sites contrib app. If you enable this setting, the *META_SITE_DOMAIN* is not used at all. Default is `False`.

### 1.4.13 META_OG_NAMESPACES

Use this setting to add a list of additional OpenGraph namespaces to be declared in the `<head>` tag.

### 1.4.14 Other settings

The following settings are available to set a default value to the corresponding attribute for both *View support* and *Model support*

- image: `META_DEFAULT_IMAGE` (must be an absolute URL, ignores *META_IMAGE_URL*)
- object_type: `META_SITE_TYPE` (default: first `META_OBJECT_TYPES`)
- og_type: `META_FB_TYPE` (default: first `META_FB_TYPES`)
- og_app_id: `META_FB_APPID` (default: blank)
- og_profile_id: `META_FB_PROFILE_ID` (default: blank)
- fb_pages: `META_FB_PAGES` (default: blank)
- og_publisher: `META_FB_PUBLISHER` (default: blank)
- og_author_url: `META_FB_AUTHOR_URL` (default: blank)
- twitter_type: `META_TWITTER_TYPE` (default: first `META_TWITTER_TYPES`)
- twitter_site: `META_TWITTER_SITE` (default: blank)
- twitter_author: `META_TWITTER_AUTHOR` (default: blank)
- gplus_type: `META_GPLUS_TYPE` (default: first `META_GPLUS_TYPES`)
- gplus_author: `META_GPLUS_AUTHOR` (default: blank)
- gplus_publisher: `META_GPLUS_PUBLISHER` (default: blank)

## 1.5 Rendering meta tags

To render the meta tags, simply add the `meta` dictionary/object to the template context, and add this inside the `<head>` tags:

```
{% include 'meta/meta.html' %}
```

The partial template will not output anything if the context dictionary does not contain a `meta` object, so you can safely include it in your base template.

Additionally, if you want to use facebook or a custom namespace, you should include them in the <head> tag, as follow:

```
{% load meta %}
...
<head {% meta_namespaces %} >
```

This will take care of rendering OpenGraph namespaces in the `<head prefix="...">`.

If you enabled Google+ Support you have to add the following templatetag to the `<html>` tag:

```
{% load meta %}
...
<html {% meta_namespaces_gplus %}>
```

For compatibility with 1.0 and previous version you can keep the sekizai version of the above:

```
{% load sekizai_tags meta %}
...
<html {% render_block 'html_extra' %}>
```

## 1.6 Package documentation

**class** meta.models.**ModelMeta**
Bases: object

Meta information mixin.

**as_meta**(*request=None*)
Method that generates the Meta object (from django-meta)

**build_absolute_uri**(*url*)
Return the full url for the provided url argument

**get_author**()
Retrieve the author object. This is meant to be overridden in the model to return the actual author instance (e.g.: the user object).

**get_author_gplus**()
Sample method to return the author google plus URL

**get_author_name**()
Sample method to return the author full name

**get_author_twitter**()
Sample method to return the author twitter account

**get_author_url**()
Sample method to return the author facebook URL

**get_meta**(*request=None*)
Retrieve the meta data configuration

**get_meta_protocol**()
Current http protocol

**get_request**()
Retrieve request from current instance

**class** meta.views.**Meta**(***kwargs*)
Bases: object

Helper for building context meta object

**get_domain**()

**get_full_url**(*url*)

**get_protocol**()

**image**

**keywords**

**request = None**

**url**

**class** meta.views.**MetadataMixin**(***kwargs*)
Bases: object

Django CBV mixin to prepare metadata for the view context

**context_meta_name = u'meta'**

**custom_namespace = None**

**description = None**

**extra_custom_props = None**

**extra_props = None**

**facebook_app_id = None**

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

**get_domain**()

**get_meta**(*context=None*)

**get_meta_class**()

**get_meta_custom_namespace**(*context=None*)

**get_meta_description**(*context=None*)

**get_meta_extra_custom_props**(*context=None*)

**get_meta_extra_props**(*context=None*)

**get_meta_facebook_app_id**(*context=None*)

**get_meta_gplus_author**(*context=None*)

**get_meta_gplus_publisher**(*context=None*)

**get_meta_gplus_type**(*context=None*)

**get_meta_image**(*context=None*)

**get_meta_keywords**(*context=None*)

**get_meta_locale**(*context=None*)

**get_meta_object_type**(*context=None*)

**get_meta_site_name**(*context=None*)

**get_meta_title**(*context=None*)

**get_meta_twitter_card**(*context=None*)

**get_meta_twitter_creator**(*context=None*)

**get_meta_twitter_site**(*context=None*)

**get_meta_url**(*context=None*)

**get_protocol**()

**gplus_author = None**

**gplus_publisher = None**

**gplus_type = None**

**image = None**

**keywords = []**

**locale = None**

**meta_class**
alias of *Meta*

**object_type = None**

**site_name = None**

**title = None**

**twitter_card = None**

**twitter_creator = None**

**twitter_site = None**

**url = None**

**use_og = False**

**use_sites = False**

**use_title_tag = False**

meta.templatetags.meta.**custom_meta**(*attr*, *name*, *content*)
Generates a custom meta tag:

<meta {attr}="{name}" content="{content}">

**Parameters**

- **attr** – meta attribute name
- **name** – meta name
- **content** – content value

meta.templatetags.meta.**custom_meta_extras**(*extra_custom_props*)
Generates the markup for a list of custom meta tags

Each tuple is passed to :py:func:custom_meta to generate the markup

**Parameters** **extra_custom_props** – list of tuple of additional meta tags

meta.templatetags.meta.**facebook_prop**(*name*, *value*)
Generic Facebook property

**Parameters**

- **name** – property name (without 'fb:' namespace)
- **value** – property value

meta.templatetags.meta.**generic_prop**(*namespace*, *name*, *value*)
Generic property setter that allows to create custom namespaced meta e.g.: fb:profile_id.

meta.templatetags.meta.**googleplus_html_scope**(*value*)
This is meant to be used as attribute to html / body or other tags to define schema.org type

**Parameters** **value** – declared scope

meta.templatetags.meta.**googleplus_prop**(*name*, *value*)
Generic Google+ property

**Parameters**

- **name** – property name
- **value** – property value

meta.templatetags.meta.**googleplus_scope**(*value*)
Alias for googleplus_html_scope

**Parameters** **value** – declared scope

meta.templatetags.meta.**meta**(*name*, *content*)
Generates a meta tag according to the following markup:

<meta name="{name}" content="{content}">

**Parameters**

- **name** – meta name
- **content** – content value

meta.templatetags.meta.**meta_extras**(*extra_props*)
Generates the markup for a list of meta tags

Each key,value paur is passed to :py:func:meta to generate the markup

**Parameters** **extra_props** – dictionary of additional meta tags

meta.templatetags.meta.**meta_list**(*name*, *lst*)
Renders in a single meta a list of values (e.g.: keywords list)

**Parameters**

- **name** – meta name
- **lst** – values

meta.templatetags.meta.**meta_namespaces**(*context*)
Include OG namespaces. To be used in the <head> tag.

meta.templatetags.meta.**meta_namespaces_gplus**(*context*)
Include google+ attributes. To be used in the <html> or <body> tag.

meta.templatetags.meta.**og_prop**(*name*, *value*)
Generic OpenGraph property

**Parameters**

- **name** – property name (without 'og:' namespace)
- **value** – property value

meta.templatetags.meta.**title_prop**(*value*)
Title tag

**Parameters** **value** – title value

meta.templatetags.meta.**twitter_prop**(*name*, *value*)
Generic Twitter property

**Parameters**

- **name** – property name (without 'twitter:' namespace)

- **value** – property value

## 1.7 Development & community

django meta is an open-source project.

You don't need to be an expert developer to make a valuable contribution - all you need is a little knowledge, and a willingness to follow the contribution guidelines.

### 1.7.1 Nephila

django meta is maintained by Iacopo Spalletti at Nephila and is released under a BSD License.

Nephila is an active supporter of Django, django CMS and its community. django meta is intended to help make it easier for developers in the Django ecosystem to work effectively and create high quality applications.

## 1.8 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

### 1.8.1 Types of Contributions

#### Report Bugs

Report bugs at https://github.com/nephila/django-meta/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

#### Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

#### Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

#### Write Documentation

django-meta could always use more documentation, whether as part of the official django-meta docs, in docstrings, or even on the web in blog posts, articles, and such.

#### Submit Feedback

The best way to send feedback is to file an issue at https://github.com/nephila/django-meta/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

### 1.8.2 Get Started!

Ready to contribute? Here's how to set up *django-meta* for local development.

1. Fork the *django-meta* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-meta.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-meta
$ cd django-meta/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 meta_mixin tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

### 1.8.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.

2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/nephila/django-meta/pull_requests and make sure that the tests pass for all supported Python versions.

### 1.8.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_meta_mixin
```

## 1.9 History

### 1.9.1 1.4.1 (2018-01-21)

- Added Django 2.0 support
- Fixed RTD builds
- Fixed MetadataMixin.use_use_title_tag typo
- Add request to Meta arguments

### 1.9.2 1.4.0 (2017-08-12)

- Add Django 1.11 support
- Drop python 2.6/ Django<1.8
- Wrap meta.html content in spaceless templatetag to suppress redundant newlines
- Fix issue in Django 1.10

### 1.9.3 1.3.2 (2016-10-26)

- Fix error if custom_meta_extras is empty
- Fix twitter properties
- Fix error with META_DEFAULT_IMAGE path

### 1.9.4 1.3.1 (2016-08-01)

- Add support for G+ publisher tag

### 1.9.5 1.3 (2016-06-06)

- Added support for fb_pages attribute
- Properly implement META_DEFAULT_IMAGE for view-based mixins
- Fixed error in facebook_prop templatetag

- Removed dependency of sites framework

### 1.9.6 1.2 (2016-04-09)

- Fix issue when emulating sekizai

### 1.9.7 1.1 (2016-04-08)

- Sekizai is not required anymore

### 1.9.8 1.0 (2016-03-29)

- Merge with django-meta-mixin
- Reorganized documentation
- Remove deprecated `make_full_url` method
- Add _retrieve_data interface for generic attribute data generation

### 1.9.9 0.3.2 (2016-02-09)

- Use autoescape off in template for Django 1.9

### 1.9.10 0.3.1 (2015-06-27)

- Bump for re-upload

### 1.9.11 0.3.0 (2015-06-27)

- Add support for more twitter attributes
- Add support for more facebook attributes
- Official support for Django 1.4->1.8
- Official support for Python 2.6, 2.7, 3.2, 3.3, 3.4

### 1.9.12 0.2.1 (2014-12-15)

- Add support for more attributes
- Add templatetag to handle generic attributes

### 1.9.13 0.2.0 (2014-05-28)

- Code cleanup
- Change maintainership information
- Official Python 3 support

### 1.9.14 0.1.0 (2014-01-20)

- Support for Twitter meta data (leifdenby)
- Fixes to OpenGraph tags (leifdenby)
- Support Google Plus tags (Iacopo Spalletti)

### 1.9.15 0.0.3 (2013-11-12)

- Keywords are now order-preserving
- Keywords are no longer a set(), but a normal list

### 1.9.16 0.0.2 (2013-04-12)

- Fixed keywords not being included in metadata
- Fixed get_meta_class not being used in the mixin

### 1.9.17 0.0.1 (2013-04-04)

- Initial version

CHAPTER 2

# Apps using django-meta / extensions

- djangocms-blog: https://github.com/nephila/djangocms-blog
- djangocms-page-meta: https://github.com/nephila/djangocms-page-meta
- django-knocker: https://github.com/nephila/django-knocker
- wagtail-metadata-mixin: https://github.com/bashu/wagtail-metadata-mixin

Open a pull request to add yours here

## 2.1 Indices and tables

- genindex
- modindex
- search

# Python Module Index

## m

# Index

## I



## K



## L



## M



## O



## R



## S



## T



## U