
Django Extras Documentation

Release 0.2.7.b1

Tim Savage

January 23, 2015

1	Django Extras documentation	1
1.1	Project Status	1
1.2	Getting help	1
1.3	What's in Django Extras	1
1.4	First steps	1
1.5	Other batteries included	1
1.6	Source code	2
2	API Reference	3
2.1	<code>django_extras.contrib.auth</code>	3
2.2	Forms	3
2.3	Middleware	4
2.4	Models	4
2.5	Response objects	5
2.6	Validators	7
3	User authentication extensions	9
3.1	Installation	9
3.2	Convenience decorators	9
3.3	Ownership Mixin Models	10
3.4	Shortcuts	11
	Python Module Index	13

Django Extras documentation

1.1 Project Status

This is a fairly new project, while this package is utilised in production on several projects the status will remain as beta until unit-test coverage has been expanded. Documentation is now fairly complete, in general areas of the code base that have not been documented should be considered unstable.

1.2 Getting help

- Report bugs with Django Extras with the [issue tracker](#).

1.3 What's in Django Extras

Django Extras is a project that provides extensions for [Django](#) to solve common development situations not (or not yet) covered by the core Django framework.

Examples of this include:

- additional decorators
- model mixins to easily assign owners to a model
- additional model and form fields
- greatly expanded collection of default response classes

See [API Reference](#) full reference.

1.4 First steps

If you are new to Django it is recommended you visit the [Django documentation](#) as they have excellent documentation to get you up and running.

1.5 Other batteries included

- *Authentication*

1.6 Source code

Full [source code](#) is available on Git Hub. Have migrated away from BitBucket for Travis CI support.

2.1 `django_extras.contrib.auth`

See *User authentication extensions*.

2.2 Forms

Form API reference.

2.2.1 Form field reference

Field types

`ColorField`

class `ColorField` (`[allow_alpha=False, max_length=40, **options]`)

A `CharField` that checks that the value is a valid CSS color value. `allow_alpha` controls if colors can support alpha values.

Specifies that the default widget is the `JQueryColorPicker`.

`JsonField`

class `JsonField` (`[dump_options={'cls': DjangoJSONEncoder}, load_options={}, **options]`)

A `TextField` that handles serialisation/deserialization of JSON structures for display in a Text input.

2.2.2 Form widget reference

Widget types

HTML5 Input types

`Html5EmailInput`

class `Html5EmailInput`
HTML 5 email input.

Html5NumberInput
class `Html5NumberInput`
HTML 5 number input.

Html5DateInput
class `Html5DateInput`
HTML 5 date input.

Html5DateTimeInput
class `Html5DateTimeInput`
HTML 5 datetime input.

Html5TimeInput
class `Html5TimeInput`
HTML 5 time input.

JQuery enhanced widgets

jQueryColorPicker
class `jQueryColorPicker`
Widget that displays a color picker. This widget assumes jQuery has been included on the page. JavaScript and CSS is included and is defined using Django media definitions.

2.3 Middleware

2.3.1 Timing Middleware

class `TimingMiddleware`
Adds a header to all responses with the total time spent generating a response. This middleware component should be added as early as possible in the list of middleware classes to get best results.

The name of the header returned to the browser is `X-PROCESSING_TIME_MS`, time is in milliseconds.

2.4 Models

Model API reference.

2.4.1 Model field reference

Note: All fields registered with `south` (if used by your project).

Field types

ColorField

```
class ColorField([allow_alpha=False, max_length=40, **options])
```

A CharField that checks that the value is a valid CSS color value. `allow_alpha` controls if colors can support alpha values.

MoneyField

```
class MoneyField([max_digits=40, decimal_places=4, **options])
```

A DecimalField that sets up sensible defaults for monetary values, in addition the MoneyField will return values as instances of the Money type. The Money type is based on Python's decimal object.

Note: The current implementation does not store the currency code with the money value.

PercentField

```
class PercentField(**options)
```

A FloatField that represents a percentage value. Validates provided value to ensure it is within the range 0 to 100.

LatitudeField

```
class LatitudeField(**options)
```

A FloatField that represents a latitude. Validates provided value to ensure it is within the range -90.0 to 90.0

LongitudeField

```
class LongitudeField(**options)
```

A FloatField that represents a longitude. Values are validated to be within the range -180.0 to 180.0.

JsonField

```
class JsonField([dump_options={'cls': DjangoJSONEncoder}, load_options={}, **options])
```

A TextField that handles serialisation/deserialization of JSON structures into a database field.

2.5 Response objects

2.5.1 HttpResponse subclasses

Django extras includes a number of additional HttpResponse subclasses that handle different types of HTTP responses. These subclasses are defined in `django_extras.http`.

Common HTTP response types

Common response codes, most of these codes are defined in the [W3C Protocol specification](#). This collection also includes status codes that are defined for Web DAV. This list does not include response types that are already defined in Django.

Successful 2xx responses

```
class HttpResponseCreated
    Status code 201

class HttpResponseAccepted
    Status code 202

class HttpResponseNonAuthoritative
    Status code 203

class HttpResponseNoContent
    Status code 204

class HttpResponseResetContent
    Status code 205

class HttpResponsePartialContent
    Status code 206
```

Redirection 3xx responses

All responses within the 3xx class inherit from `HttpResponseRedirect`.

```
class HttpResponseSeeOther
    Status code 303
```

Client Error 4xx responses

```
class HttpResponseUnauthorised
    Status code 401

class HttpResponsePaymentRequired
    Status code 402

class HttpResponseNotAcceptable
    Status code 406

class HttpResponseRequestTimeout
    Status code 408

class HttpResponseConflict
    Status code 409

class HttpResponseLengthRequired
    Status code 411

class HttpResponsePreconditionFailed
    Status code 412

class HttpResponseRequestEntityTooLarge
    Status code 413
```

class `HttpResponseUnsupportedMediaType`
Status code 415

class `HttpResponseExpectationFailed`
Status code 417

class `HttpResponseUnprocessableEntity`
Status code 422

class `HttpResponseLocked`
Status code 423

class `HttpResponseFailedDependency`
Status code 424

class `HttpResponseUpgradeRequired`
Status code 426

Server Error 5xx responses

class `HttpResponseNotImplemented`
Status code 501

class `HttpResponseBadGateway`
Status code 502

class `HttpResponseServiceUnavailable`
Status code 503

class `HttpResponseGatewayTimeout`
Status code 504

class `HttpResponseInsufficientStorage`
status_code = 507

Enhanced response types

class `FileResponse`

The constructor accepts the same `content` property as the default `:class:HttpResponse` class except it is interpreted as a file name or file handle and `content_type`. The response object facilitates streaming the content of the file to the client. There is an optional parameter `include_last_modified` which defaults to `True` that supplies the last modified date of the specified file as an HTTP header.

class `JsonResponse`

Acts just like `:class:HttpResponse` except will encode the first parameter to JSON (using `:class:DjangoJSONEncoder`) and changes the default `content_type` to `application/json`.

2.6 Validators

2.6.1 Additional validators

The `django_extras.core.validators` module contains a collection of callable validators for use with model and form fields. They're used internally but are available for use with your own fields, too. They can be used in addition to, or in lieu of custom `field.clean()` methods.

`validate_color`

validate_color

A `RegexValidator` instance that ensures a value looks like a CSS color value.

`validate_alpha_color`

validate_alpha_color

A `RegexValidator` instance that ensures a value looks like a CSS color value. Supports color definitions with alpha blending.

`validate_json`

validate_json

A `JsonValidator` instance that ensures a value is valid JSON.

User authentication extensions

While Django comes with a built in user authentication system it leaves a couple of common use-cases incomplete. Django Extras fills in the missing pieces.

3.1 Installation

No installation is required, this extension does not require additional database models.

3.2 Convenience decorators

3.2.1 The `staff_required` decorator

`staff_required` (`[include_superuser=True, login_url=None, raise_exception=False]`)

This decorator provides a simple way of restricting access to a particular view to users who have the `is_staff` flag set. Rather than using the `~django.contrib.auth.decorators.user_passes_test()` decorator restricting a view to staff can be written as:

```
from django_extras.contrib.auth.decorators import staff_required
@staff_required
def my_view(request):
    ...
```

By default this decorator also includes users with the `is_superuser` flag set, these users can be excluded with the optional `include_superuser` parameter. Example:

```
from django_extras.contrib.auth.decorators import staff_required
@staff_required(include_superuser=False)
def my_view(request):
    ...
```

As in the `login_required()`, `login_url` defaults to `settings.LOGIN_URL`.

Mirroring the change in Django 1.4 this decorator also accepts the `raise_exception` parameter. If given, the decorator will raise `PermissionDenied`.

3.2.2 The `superuser_required` decorator

`superuser_required` (`[login_url=None, raise_exception=False]`)

This decorator provides a simple way of restricting access to a particular view to users who have the `is_superuser` flag set. Rather than using the `~django.contrib.auth.decorators.user_passes_test()` decorator restricting a view to super users can be written as:

```
from django_extras.contrib.auth.decorators import superuser_required
@superuser_required
def my_view(request):
    ...
```

As in the `login_required()`, `login_url` defaults to `settings.LOGIN_URL`.

Mirroring the change in Django 1.4 this decorator also accepts the `raise_exception` parameter. If given, the decorator will raise `PermissionDenied`.

3.3 Ownership Mixin Models

Two mixin classes are provided that provide a consistent API for assigning ownership of a model instance to a user.

Example:

```
class MyModel(SingleOwnerMixin, models.Model):
    name = models.CharField(max_length=50)
```

Many methods include `include_staff` and `include_superuser` parameters, these are used to treat staff and superuser users as if they are owners of the instance.

Both `SingleOwnerMixin` and `MultipleOwnerMixin` provide the following methods.

class `OwnerMixinBase`

`OwnerMixinBase.owned_by` (`user` [`, include_staff=False, include_superuser=False`])

Returns a boolean value to indicate if the supplied user is a valid owner of a model instance.

`OwnerMixinBase.owner_list` ()

Returns a list of `User` models that are owners of the model instance. A list is returned by both single and multiple versions of the mixin.

3.3.1 `SingleOwnerMixin`

class `SingleOwnerMixin`

This class provides a simple way to assign ownership of a model instance to a single user.

owner

User object as provided by a `ForeignKey` model field.

Note: By default the `related_name` parameter is set to: `'%(app_label)s_%(class)s_owner'`

3.3.2 `MultipleOwnerMixin`

class `MultipleOwnerMixin`

This class provides a simple way to assign ownership of a model instance to multiple users.

owners

RelatedManager object as provided by a ManyToManyField model field.

Note: By default the `related_name` parameter is set to: `'%(app_label)s_%(class)s_owners'`

3.3.3 OwnerMixinManager

class OwnerMixinManager

Manager class used by `SingleOwnerMixin` and `MultipleOwnerMixin` to obtain the instances of a model that has ownership assigned to a particular user.

Fetching owned instances

```
OwnerMixinManager.owned_by(user[, include_staff=False, include_superuser=False])
```

Returns a `QuerySet` filtered by a user or users. The user parameter can be either a single `User` object or primary key, a sequence of `User` objects or primary keys.

Example:

```
# Single user
>>> MyModel.objects.owned_by(request.user)
[<MyModel: Foo>, <MyModel: Bar>]

# Multiple primary keys
>>> MyModel.objects.owned_by([1, 2, 3])
[<MyModel: Foo>, <MyModel: Bar>, <MyModel: Eek>]
```

Note: It is not possible to use the `include_staff` and `include_superuser` parameters when passing a sequence for the `user` parameter. A `TypeError` exception will be raised in this case.

3.4 Shortcuts

```
get_owned_object_or_40x([klass, owner, include_staff=False, include_superuser=False, *args,
                        **kwargs])
```

A convenience method that mirrors the Django shortcut `get_object_or_404`. If the object cannot be loaded a `Http404` exception is raised, if a user cannot be verified as an owner a `PermissionDenied` exception is raised.

As with the other extensions `include_staff` and `include_superuser` flags are provided. `*args` and `**kwargs` work in the same way as `get_object_or_404`.

d

`django_extras.contrib.auth`, 9
`django_extras.core.validators`, 7
`django_extras.db.models.fields`, 4
`django_extras.forms.fields`, 3
`django_extras.forms.widgets`, 3
`django_extras.http`, 5
`django_extras.middleware`, 4
`django_extras.middleware.timing`, 4

C

ColorField (class in django_extras.db.models), 5
 ColorField (class in django_extras.forms.fields), 3

D

django_extras.contrib.auth (module), 9
 django_extras.core.validators (module), 7
 django_extras.db.models.fields (module), 4
 django_extras.forms.fields (module), 3
 django_extras.forms.widgets (module), 3
 django_extras.http (module), 5
 django_extras.middleware (module), 4
 django_extras.middleware.timing (module), 4

F

FileResponse (class in django_extras.http), 7

G

get_owned_object_or_40x() (in module
 django_extras.contrib.auth.shortcuts), 11

H

Html5DateInput (class in django_extras.forms.widgets),
 4
 Html5DateTimeInput (class in
 django_extras.forms.widgets), 4
 Html5EmailInput (class in django_extras.forms.widgets),
 3
 Html5NumberInput (class in
 django_extras.forms.widgets), 4
 Html5TextInput (class in django_extras.forms.widgets),
 4
 HttpResponseAccepted (class in django_extras.http), 6
 HttpResponseBadGateway (class in django_extras.http),
 7
 HttpResponseConflict (class in django_extras.http), 6
 HttpResponseCreated (class in django_extras.http), 6
 HttpResponseExpectationFailed (class in
 django_extras.http), 7

HttpResponseFailedDependency (class in
 django_extras.http), 7
 HttpResponseGatewayTimeout (class in
 django_extras.http), 7
 HttpResponseInsufficientStorage (class in
 django_extras.http), 7
 HttpResponseLengthRequired (class in
 django_extras.http), 6
 HttpResponseLocked (class in django_extras.http), 7
 HttpResponseNoContent (class in django_extras.http), 6
 HttpResponseNonAuthoritative (class in
 django_extras.http), 6
 HttpResponseNotAcceptable (class in
 django_extras.http), 6
 HttpResponseNotImplemented (class in
 django_extras.http), 7
 HttpResponsePartialContent (class in django_extras.http),
 6
 HttpResponsePaymentRequired (class in
 django_extras.http), 6
 HttpResponsePreconditionFailed (class in
 django_extras.http), 6
 HttpResponseRequestEntityTooLarge (class in
 django_extras.http), 6
 HttpResponseRequestTimeout (class in
 django_extras.http), 6
 HttpResponseResetContent (class in django_extras.http),
 6
 HttpResponseSeeOther (class in django_extras.http), 6
 HttpResponseServiceUnavailable (class in
 django_extras.http), 7
 HttpResponseUnAuthorised (class in django_extras.http),
 6
 HttpResponseUnprocessableEntity (class in
 django_extras.http), 7
 HttpResponseUnsupportedMediaType (class in
 django_extras.http), 6
 HttpResponseUpgradeRequired (class in
 django_extras.http), 7

J

jQueryColorPicker (class in django_extras.forms.widgets), 4
JsonField (class in django_extras.db.models), 5
JsonField (class in django_extras.forms.fields), 3
JsonResponse (class in django_extras.http), 7

L

LatitudeField (class in django_extras.db.models), 5
LongitudeField (class in django_extras.db.models), 5

M

MoneyField (class in django_extras.db.models), 5
MultipleOwnerMixin (class in django_extras.contrib.auth.models), 10

O

owned_by() (OwnerMixinBase method), 10
owned_by() (OwnerMixinManager method), 11
owner (SingleOwnerMixin attribute), 10
owner_list() (OwnerMixinBase method), 10
OwnerMixinBase (class in django_extras.contrib.auth.models), 10
OwnerMixinManager (class in django_extras.contrib.auth.models), 11
owners (MultipleOwnerMixin attribute), 10

P

PercentField (class in django_extras.db.models), 5

S

SingleOwnerMixin (class in django_extras.contrib.auth.models), 10
staff_required() (in module django_extras.contrib.auth.decorators), 9
superuser_required() (in module django_extras.contrib.auth.decorators), 10

T

TimingMiddleware (class in django_extras.middleware.timing), 4

V

validate_alpha_color (in module django_extras.core.validators), 8
validate_color (in module django_extras.core.validators), 8
validate_json (in module django_extras.core.validators), 8