django-core Documentation

Release 0.0.1

Troy Grosfield

November 16, 2015

Contents

CHAPTER 1

Intallation

Install the app via pip:

```
pip install django-core
```

CHAPTER 2

Examples

TODO

CHAPTER 3

API Reference

3.1 Models

3.1.1 Mixins

class `django_core.db.models.mixins.base.`**`AbstractBaseModel`**(**args*, ***kwargs*)

Base model for other db model to extend. This class contains common model attributes needed by almost all models.

Fields:

- created: created user. The user who created this instance.
- created_dttm: created datetime.
- last_modified: last user to modify this instance
- **last_modified_dttm: updated datetime. Datetime this document was last** updated.

`copy`(*exclude_fields=None*, ***override_fields*)

Returns an unsaved copy of this object with all fields except for any fields that are unique in the DB. Those fields must be explicitly set before saving the instance:

- id
- created_dttm
- last_modified_dttm

NOTE: If a field doesn't except null values, you must explicitly set the value in the override fields or this method will error out since you can't set that fields to be null.

Parameters

- **`exclude_fields`** – fields to exclude from the copy. They will fallback to the field default if one is given or None.
- **`override_fields`** – kwargs with fields to override. The key is the field name, the value is the value to set the copied object to.

  Example:

  >> new_obj = some_obj.copy(my_field='hello world') >> new_obj.my_field 'hello world'

`get_verbose_name`()

Gets the verbose name for an object.

classmethod m2m_changed(**args*, ***kwargs*)
Adding a hook here so it's safe to call the super's m2m_changed.

classmethod post_delete(**args*, ***kwargs*)
Adding a hook here so it's safe to call the super's post_delete.

classmethod post_save(**args*, ***kwargs*)
Adding a hook here so it's safe to call the super's post_save.

save(**args*, ***kwargs*)
Optional kwargs:

- **id_length: the length of characters to use for the id. Default** is 10.

classmethod save_prep(*instance_or_instances*)
Common save functionality for all models. This can be called with a saved or unsaved instance or one or many objects. This is beneficial when additional process needs to happen before a bulk_create which doesn't explicitly call the .save on each instance being saved. This method will go through each object and do necessary presave processing.

This method can be extended by classes and implement this abstact class by simply creating the def save_prep method and making sure to call super class method making sure the save_prep method is properly called from each inheriting class:

All models.CharField will be stripped prior to saving.

Example:

@classmethod def save_prep(cls, instance_or_instances):

Do additional processing for inheriting class super(MyInheritingClass, cls).save_prep(instance_or_instances)

Note: Make sure not to call the save_prep method in the save method of inheriting classes or it will get called twice which likely isn't wanted since this Abstract class explicitly calls the save_prep on save().

All objects are assumed to have the following fields:

- id
- created
- created_dttm
- last_modified
- last_modified_dttm

strip_fields()
Strips whitespace from all text related model fields. This includes CharField and TextFields and all subclasses of those two fields.

class django_core.db.models.mixins.crud.**AbstractSafeDeleteModelMixin**(**args*, ***kwargs*)
Give a model safe delete logic so an indicator can be set to is_deleted and not removed from the database.

class django_core.db.models.mixins.crud.**ReadOnlyModelMixin**(**args*, ***kwargs*)
This is a wrapper class around a model so all methods and fields can be used the same as the extending model, but this doesn't allow the model instance to be saved.

class django_core.db.models.mixins.tokens.**AbstractTokenModel**(**args*, ***kwargs*)
Abstract class for token logic.

save(**args*, ***kwargs*)
Make sure token is added.

classmethod **save_prep**(*instance_or_instances*)
Preprocess the object before the object is saved. This automatically gets called when the save method gets called.

class django_core.db.models.mixins.urls.**AbstractUrlLinkModelMixin**(**args*, ***kwargs*)
Mixin for accessing the links for the models. This requires the object to have already implemented the following methods:

To override the model field used for the absolute url, override the following method:

- get_link_text_field()

to the model and that field will be used for the text.

- get_absolute_url - returns the absolute link to the object.
- get_edit_url - returns the link to edit the object
- get_delete_url - return the link to delete the object.

get_absolute_url_link(*text=None*, *cls=None*, *icon_class=None*, ***attrs*)
Gets the html link for the object.

get_delete_url_link(*text=None*, *cls=None*, *icon_class=None*, ***attrs*)
Gets the html delete link for the object.

get_edit_url_link(*text=None*, *cls=None*, *icon_class=None*, ***attrs*)
Gets the html edit link for the object.

get_link_text_field()
This returns the field name to use for the absolute url.

3.1.2 Fields

3.1.3 Managers

class django_core.db.models.managers.**SlugManager**
Manager mixin for slugs.

get_next_slug(*slug*, ***kwargs*)
Gets the next available slug.

Parameters

- **slug** – the slug to slugify
- **kwargs** – additional filter criteria to check for when looking for a unique slug.

Example:

if the value "my-slug" is already taken, this method will append "-n" to the end of the slug until the next available slug is found.

is_slug_available(*slug*, ***kwargs*)
Checks to see if a slug is available. If the slug is already being used this method returns False. Otherwise, return True.

class django_core.db.models.managers.**TokenManager**
Manager Mixin for tokens.

get_available_tokens(*count=10*, *token_length=15*, ***kwargs*)
Gets a list of available tokens.

Parameters

- **count** – the number of tokens to return.
- **token_length** – the length of the tokens. The higher the number the easier it will be to return a list. If token_length == 1 there's a strong probability that the enough tokens will exist in the db.

get_by_token(*token*, ***kwargs*)
Get by token.

get_next_token(*length=15*, ***kwargs*)
Gets the next available token.

Parameters

- **length** – length of the token
- **kwargs** – additional filter criteria to check for when looking for a unique token.

class django_core.db.models.managers.**UserManager**
Manager Mixin for models that have a user.

3.2 Forms

3.2.1 Fields

class django_core.forms.fields.**CharFieldStripped**(*max_length=None*, *min_length=None*, **args*, ***kwargs*)
Wrapper around CharField that strips whitespace from the CharField when validating so .strip() doesn't have to be called every time you validate the field's data.

class django_core.forms.fields.**CommaSeparatedIntegerListField**(*max_list_length=None*, *max_list_length_error_msg=None*, **args*, ***kwargs*)
Comma Separated Integer list field.

class django_core.forms.fields.**CommaSeparatedListField**(*max_list_length=None*, *max_list_length_error_msg=None*, **args*, ***kwargs*)
Form field that takes a string and converts into a list of strings.

class django_core.forms.fields.**MultipleDecimalField**(*num_inputs=2*, *value_suffix=u'‘*, **args*, ***kwargs*)
A field with multiple decimal fields that should be converted to single line.

Example response values:

•"5px 0 5px 4px"

clean(*value*)
Validates that the input can be converted to a list of decimals.

to_python(*value*)
Validates that the input can be converted to a list of decimals.

widget
alias of MultipleDecimalInputWidget

3.2.2 Widgets

class django_core.forms.widgets.**ChoiceAndCharInputWidget**(*choices=None*, *attrs=None*, *widgets=None*, *widget_css_class=u'choice-and-char-widget'*, ***kwargs*)

Renders choice field and char field next to each other.

class django_core.forms.widgets.**CommaSeparatedListWidget**(**args*, ***kwargs*)

Widget for rendering a comma separated list using a text field.

class django_core.forms.widgets.**ExtendedMultiWidget**(*widget_css_class=None*, ***kwargs*)

Wrapper around the MultiWidget that allow for putting a custom css class around multiple widgets.

class django_core.forms.widgets.**Html5DateInput**(*date_format=u'%Y-%m-%d'*, **args*, ***kwargs*)

Renders an HTML5 date widget.

class django_core.forms.widgets.**Html5DateTimeInput**(*date_format=u'%Y-%m-%d %H:%M:%S'*, **args*, ***kwargs*)

Renders an HTML5 datetime widget.

class django_core.forms.widgets.**MultipleDecimalInputWidget**(*attrs=None*, *widgets=None*, *num_inputs=4*, *value_suffix=u''*, *widget_css_class=u'horizontal-widget multi-decimal-widget'*, ***kwargs*)

Renders an input with 4 decimal fields.

get_widget_css_class(*attrs*)

Gets the class for the widget.

class django_core.forms.widgets.**ReadonlyWidget**(*attrs=None*)

This renders a readonly field that can also override the default display value.

3.2.3 Mixins

class django_core.forms.mixins.paging.**PagingFormMixin**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*)

Form mixin that includes paging page number and page size.

class django_core.forms.mixins.query.**QueryFormMixin**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*)

Form Mixin for free text query field.

3.3 Middleware

class `django_core.middleware.browser.`**`IECompatibleMiddleware`**

Configures how windows internet explorer renders the webpage by setting the user agenet compatability mode to edge.

Internet Explorer uses a browser and document mode to determine how to render a web page. Without the X-UA-Compatible header, Internet Explorer will attempt to pick the rendering mode based a number of different criteria. This may result in a web page running in IE8 or IE9 rendering as if it were in IE7. Setting the X-UA-Compatible header ensures that Internet Explorer always renders the page as the latest version of the browser it is being viewed in.

See: http://www.alistapart.com/articles/beyonddoctype See: http://msdn.microsoft.com/en-us/library/cc288325%28v=vs.85%29.aspx

3.4 Template Tags

`django_core.templatetags.collection_tags.`**`attr`**(*obj*, *attr*)

Does the same thing as getattr.

getattr(obj, attr, '')

`django_core.templatetags.collection_tags.`**`getitem`**(*d*, *key*)

Ability to access a dictionary keys based on a dynamic key:

Usage:

my_vals = {'hello': 'world', 'testing': 'again'}

{{ my_vals|get:'hello' }}

would return "world"

`django_core.templatetags.collection_tags.`**`jsondumps`**(*obj*)

Turns a json object into a string.

`django_core.templatetags.collection_tags.`**`make_iterable`**(*obj*)

Make an object iterable.

```
>>> make_iterable(obj='hello')
('hello',)
>>> make_iterable(obj=None)
()
```

`django_core.templatetags.math_tags.`**`absolute`**(*value*)

Get the absolute value for "value". This template tag is a wrapper for pythons "abs(...)" method.

Usage:

```
>>> absolute(-5)
5
```

`django_core.templatetags.math_tags.`**`divide`**(*numerator*, *denominator*)

Divides two values from each other.

Usage:

```
>>> absolute(-5)
5
```

django_core.templatetags.math_tags.**multiply**(*value*, *multiplier*)
Multiplies two values together.

Usage:

```
>>> multiply(5, 2)
10
```

django_core.templatetags.math_tags.**substract**(*value*, *subtract_by*)
Get the absolute value for "value". This template tag is a wrapper for pythons "abs(...)" method.

Usage:

```
>>> absolute(5, 2)
3
```

django_core.templatetags.url_tags.**delete_url_link**(*obj*, ***kwargs*)
This method assumes that the "get_delete_url_link" method has been implemented on the obj.

django_core.templatetags.url_tags.**edit_url_link**(*obj*, ***kwargs*)
This method assumes that the "get_delete_url_link" method has been implemented on the obj.

django_core.templatetags.url_tags.**get_absolute_url_link**(*obj*, *text=None*)
Gets the absolute url html link for the object.

Usage:

{{ obj|get_absolute_url_link:"Some Text" }}

Would return:

u'{{ TEXT THAT WAS PASSED IN }}'

django_core.templatetags.url_tags.**get_delete_url_link**(*obj*, *text=None*)
Gets the absolute url html link for the object.

django_core.templatetags.url_tags.**get_edit_url_link**(*obj*, *text=None*)
Gets the absolute url html link for the object.

Usage:

{{ obj|get_edit_url_link:"Some Text" }}

Would return:

u'{{ TEXT THAT WAS PASSED IN }}'

3.5 Views

class django_core.views.request.**ApiFormView**(***kwargs*)
Form view for Api's to leverage forms and correctly validate query string data.

form_invalid(*form*, *context=None*, ***kwargs*)
This will return the request with form errors as well as any additional context.

get_form_kwargs()
Add the 'data' to the form args so you can validate the form data on a get request.

class django_core.views.response.**JSONHybridCreateView**(***kwargs*)
Hybrid view that handles regular create requests as well as json create requests.

class django_core.views.response.**JSONHybridProcessFormViewMixin**
Hybrid mixin that handles form processing.

Fields:

json_template_name: if provided, this template will be used to render an html template response that will be returned in the response data.

Example:

class MyView(JSONHybridProcessFormViewMixin, CreateView): json_template_name = 'path/to/json_template.html'

def get_json_context_data(self, **kwargs): context = super(MyView, self).get_json_context_data(**kwargs) context['my_json_template_var'] = 'hello world' return context

Successful JSON response (form is valid):

{ 'html': ' Some rendererd html response "hello world" '

}

class django_core.views.response.**JSONHybridUpdateView**(***kwargs*)
Hybrid view that handles regular update requests as well as json update requests.

class django_core.views.response.**JSONResponse**(*content*, *status=200*, ***kwargs*)
Returns a HttpResponse that has content that's json encoded. Returns a status of 200.

Response content sample:

```
{
    activity: "activity html",
    additional_content_key1: additional_content_value1
}
```

Parameters **content** – a dictionary of content that should be returned with the response.

Returns HttpResponse with json encoded activity content.

class django_core.views.response.**JSONResponseMixin**
Mixin For returning a json response.

get_json_response(*content*, ***kwargs*)
Returns a json response object.

3.5.1 Mixins

class django_core.views.mixins.auth.**CreatorRequiredViewMixin**
Mixin that requires the self.object be created by the authenticated user.

class django_core.views.mixins.auth.**LoginRequiredViewMixin**
Use this with CBVs to ensure user is logged in.

class django_core.views.mixins.auth.**StaffRequiredViewMixin**
Require a logged in Staff member.

class django_core.views.mixins.auth.**SuperuserRequiredViewMixin**
Require a logged in user to be a superuser.

class django_core.views.mixins.csrf.**CsrfExemptViewMixin**
Mixin for the csrf_exempt decorator.

class django_core.views.mixins.paging.**PagingViewMixin**
View mixin for views that deal with paging.

get_paging()
Gets the paging values passed through the query string params.

- "p" for "page number" and
- "ps" for "page size".

Returns tuple with the page being the first part and the page size being the second part.

class django_core.views.mixins.query.**QueryStringAliasViewMixin**
Mixin to let you map GET query string keys to form keys.

This allows you to use alias keys in your forms so you can keeps shorter urls or rename params in the GET query string that will map nicely with django's forms.

This only applies to the request's GET method. If a short key is used it will be mapped and a new "initial" dict will be returned for the form with the correct initial mapping. If a short key doesn't exist, the key will be used as-is.

Example:

Consuming view implements the following attribute:

```
query_key_mapper = {'t': 'title'}
```

and a url query string is:

```
?t=hello&foo=bar
```

This will result in an initial dict for the form being returned as:

```
{
    'title': 'hello',
    'foo': 'bar'
}
```

get_query_key_mapper()
Returns a dictionary of the query params trying to be mapped.

map_query_string()
Maps the GET query string params the the query_key_mapper dict and updates the request's GET QueryDict with the mapped keys.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

d

A

C

D

E

F

G

H

I

J

L

M

P

Q

R

S

T

U

W