Django tags input Documentation

Release 6.0.0

Rick van Hattem

Mar 01, 2023

CONTENTS

Contents:

CHAPTER

ONE

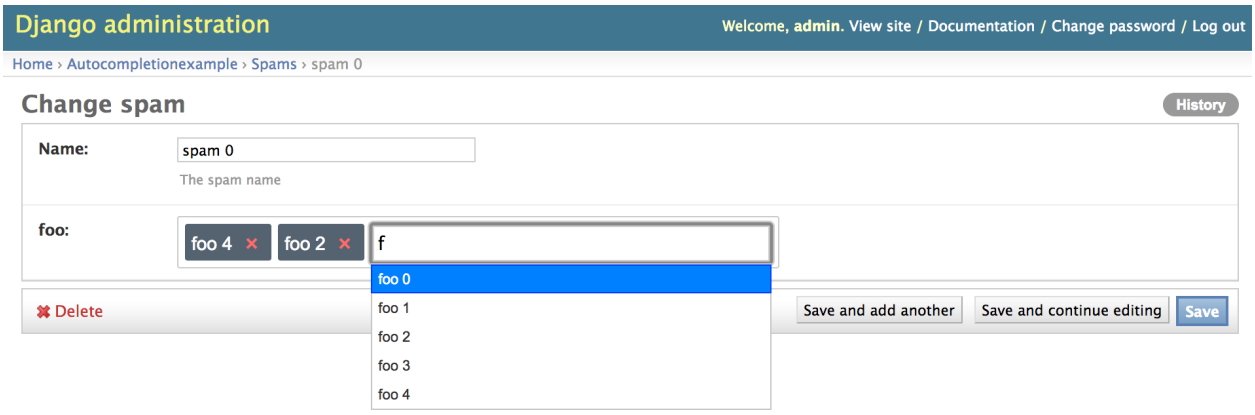
OVERVIEW

Django Tags Input is a module that gives you a modified version of the Xoxco jQuery Tags Input library within Django.

The result is a very pretty interface with tags and autocomplete which can optionally automatically create new items when they are missing.

One of the most useful features of Django Tags Input is that it stores the elements in the order which you input.

So if you insert *B, A, C* into the database, it will return it sorted the way you entered it: *B, A, C*.



CHAPTER

TWO

LINKS

- The source: https://github.com/WoLpH/django-tags-input
- Project page: https://pypi.python.org/pypi/django-tags-input
- Reporting bugs: https://github.com/WoLpH/django-tags-input/issues
- Documentation: http://django-tags-input.readthedocs.org/en/latest/
- My blog: http://w.wol.ph/

CHAPTER

THREE

HOW TO INSTALL

Installing this module only takes a couple of minutes.

Currently Django 2.2, 3.1 and 3.2 are supported and tested in combination with Python 3.7, 3.8, 3.9 and 3.10.

For Django below 2.2 and Python 2.7 up to version 4.6.0 should work.

For Django 1.4, Django 1.5, Django 1.6 and Django 1.7 in combination with Python 2.6 and 2.7. Python 3.2, 3.3 and 3.4. Pypy and Pypy3 version 2.1.0 can be used.

1. Install the module itself

   pip install django-tags-input

   # or

   easy_install django-tags-input

2. Add `tags_input` to your `INSTALLED_APPS` setting in the Django `settings.py`.

   Example:

```
INSTALLED_APPS = (
    # ... your other installed apps
    'tags_input',
)
```

3. Add the mappings to your `settings.py` file:

   Example:

```
TAGS_INPUT_MAPPINGS = {
    'some_app.SomeKeyword': {
        'field': 'some_field',
    },
    'some_app.SomeOtherKeyword': {
        'fields': ('some_field', 'some_other_field'),
    },
    'some_app.SomeSortedKeyword': {
        'field': 'some_field',
        'ordering': [
            'some_field',
            'some_other_field',
        ],
        'filters': {
            'some_field__istartswith': 'a',
        },
```

(continues on next page)

(continued from previous page)

```
        'excludes': {
            'some_field__iexact': 'foobar',
        },
    },
    'some_app.SomeCreateableKeyword': {
        'field': 'some_field',
        'create_missing': True,
    },
}
```

4. Add the `tags_input` urls to your `urls.py`:

   Example:

```
from django.conf import urls

urlpatterns = patterns('',
    url(r'^tags_input/', include('tags_input.urls', namespace='tags_input')),
    # ... other urls ...
)
```

CHAPTER

FOUR

ADMIN USAGE

```
from django.contrib import admin
import models
from tags_input import admin as tags_input_admin

class YourAdmin(tags_input_admin.TagsInputAdmin):

    #Optionally specify which ManyToMany fields are to be used for tagging
    #Or define a get_tag_fields() method
    tag_fields = ["some_field"]

admin.site.register(models.YourModel, YourAdmin)
```

CHAPTER

FIVE

QUICKSTART

To test the project simply clone the repository, install and run the example:

```
# mkvirtualenv is part of virtualenvwrapper, using a regular virtualenv, pyvenv or␣
↪pipenv is also possible
# Or even without any type of virtualenv at all
mkvirtualenv django-tags-input
git clone https://github.com/WoLpH/django-tags-input.git
# Tested with Django 3.0
pip install django
pip install -e 'django-tags-input[tests]'
cd django-tags-input/example
python manage.py runserver
```

Now you can go to http://localhost:8000/admin/ and login with username and password *admin* and *admin* respectively.

After this you can try adding some extra *Foo* objects through the *Spam* admin here: http://localhost:8000/admin/autocompletionexample/spam/2/

Note that some parts of the example are deliberately broken to test the behaviour in broken environments.

CHAPTER

SIX

TAGS_INPUT PACKAGE

6.1 tags_input.fields module

class tags_input.fields.**AdminTagsInputField**(*queryset*, *verbose_name=None*, **args*, ***kwargs*)

Bases: *TagsInputField*

bound_data(*data*, *initial*)

Return the value that should be shown for this field on render of a bound form, given the submitted POST data for the field and the initial data, if any.

For most fields, this will simply be data; FileFields need to handle it a bit differently.

property choices

clean(*value*)

Validate the given value and return its "cleaned" value as an appropriate Python object. Raise ValidationError for any errors.

default_error_messages = {'invalid_choice': 'Select a valid choice. %s is not one of the available choices.', 'invalid_pk_value': '"%s" is not a valid value for a primary key.', 'list': 'Enter a list of values.'}

default_validators = []

empty_values = [None, '', [], (), {}]

get_bound_field(*form*, *field_name*)

Return a BoundField instance that will be used when accessing the form field in a template.

get_limit_choices_to()

Return `limit_choices_to` for this form field.

If it is a callable, invoke it and return the result.

get_mapping()

has_changed(*initial*, *data*)

Return True if data differs from initial.

hidden_widget

alias of `MultipleHiddenInput`

iterator

alias of `ModelChoiceIterator`

label_from_instance(*obj*)

Convert objects into strings and generate the labels for the choices presented by this object. Subclasses can override this method to customize the display of the choices.

prepare_value(*value*)

property queryset

run_validators(*value*)

to_python(*value*)

Return a string.

valid_value(*value*)

Check to see if the provided value is a valid choice.

validate(*value*)

Validate that the input is in self.choices.

widget

alias of *AdminTagsInputWidget*

widget_attrs(*widget*)

Given a Widget instance (*not* a Widget class), return a dictionary of any HTML attributes that should be added to the Widget, based on this Field.

class tags_input.fields.**TagsInputField**(*queryset*, ***kwargs*)

Bases: ModelMultipleChoiceField

bound_data(*data*, *initial*)

Return the value that should be shown for this field on render of a bound form, given the submitted POST data for the field and the initial data, if any.

For most fields, this will simply be data; FileFields need to handle it a bit differently.

property choices

clean(*value*)

Validate the given value and return its "cleaned" value as an appropriate Python object. Raise ValidationError for any errors.

default_error_messages = {'invalid_choice': 'Select a valid choice. %s is not one of the available choices.', 'invalid_pk_value': '"%s" is not a valid value for a primary key.', 'list': 'Enter a list of values.'}

default_validators = []

empty_values = [None, '', [], (), {}]

get_bound_field(*form*, *field_name*)

Return a BoundField instance that will be used when accessing the form field in a template.

get_limit_choices_to()

Return limit_choices_to for this form field.

If it is a callable, invoke it and return the result.

get_mapping()

has_changed(*initial*, *data*)

Return True if data differs from initial.

hidden_widget

alias of `MultipleHiddenInput`

iterator

alias of `ModelChoiceIterator`

label_from_instance(*obj*)

Convert objects into strings and generate the labels for the choices presented by this object. Subclasses can override this method to customize the display of the choices.

prepare_value(*value*)

property queryset

run_validators(*value*)

to_python(*value*)

Return a string.

valid_value(*value*)

Check to see if the provided value is a valid choice.

validate(*value*)

Validate that the input is in self.choices.

widget

alias of *TagsInputWidget*

widget_attrs(*widget*)

Given a Widget instance (*not* a Widget class), return a dictionary of any HTML attributes that should be added to the Widget, based on this Field.

6.2 tags_input.widgets module

class tags_input.widgets.**AdminTagsInputWidget**(*verbose_name*, *is_stacked*, *attrs=None*, *choices=()*)

Bases: `FilteredSelectMultiple`, *TagsInputWidgetBase*

class Media

Bases: `object`

```
css = {'all':  ('jquery.tagsinput-revisited-2.0.min.css',
'jquery-ui-1.12.1.min.css')}

js = ('jquery-3.2.1.min.js', 'jquery-ui-1.12.1.min.js',
'jquery.tagsinput-revisited-2.0.min.js')
```

add_id_index = False

allow_multiple_selected = True

build_attrs(*base_attrs*, *extra_attrs=None*, ***kwargs*)

Compatibility function for the behavior changes in Django 1.11+

checked_attribute = {'selected': True}

create_option(*name*, *value*, *label*, *selected*, *index*, *subindex=None*, *attrs=None*)

format_value(*value*)

Return selected values as a list.

get_context(*name*, *value*, *attrs*)

id_for_label(*id_*, *index='0'*)

Use an incremented id for each option where the main widget references the zero index.

input_type = 'select'

property is_hidden

is_localized = False

is_required = False

property media

needs_multipart_form = False

optgroups(*name*, *value*, *attrs=None*)

Return a list of optgroups for this widget.

option_inherits_attrs = False

option_template_name = 'django/forms/widgets/select_option.html'

options(*name*, *value*, *attrs=None*)

Yield a flat list of options for this widgets.

render(*name*, *value*, *attrs=None*, *choices=()*, *renderer=None*)

Render the widget as an HTML string.

subwidgets(*name*, *value*, *attrs=None*)

Yield all "subwidgets" of this widget. Used to enable iterating options from a BoundField for choice widgets.

supports_microseconds = True

template_name = 'django/forms/widgets/select.html'

use_required_attribute(*initial*)

Don't render 'required' if the first <option> has a value, as that's invalid HTML.

value_from_datadict(*data*, *files*, *name*)

Given a dictionary of data and this widget's name, return the value of this widget or None if it's not provided.

value_omitted_from_data(*data*, *files*, *name*)

class tags_input.widgets.**TagsInputWidget**(*on_add_tag=None*, *on_remove_tag=None*, *on_change_tag=None*, **args*, ***kwargs*)

Bases: *TagsInputWidgetBase*

class Media

Bases: object

```
        css = {'all':  ('jquery.tagsinput-revisited-2.0.min.css',
        'jquery-ui-1.12.1.min.css')}

        enable_jquery = True

        js = ('jquery-3.2.1.min.js', 'jquery-ui-1.12.1.min.js',
        'jquery.tagsinput-revisited-2.0.min.js')
```

```
add_id_index = False
```

```
allow_multiple_selected = True
```

build_attrs(*base_attrs*, *extra_attrs=None*, ***kwargs*)

Compatibility function for the behavior changes in Django 1.11+

```
checked_attribute = {'selected':  True}
```

create_option(*name*, *value*, *label*, *selected*, *index*, *subindex=None*, *attrs=None*)

format_value(*value*)

Return selected values as a list.

get_context(*name*, *value*, *attrs*)

id_for_label(*id_*, *index='0'*)

Use an incremented id for each option where the main widget references the zero index.

```
input_type = 'select'
```

```
property is_hidden
```

```
is_localized = False
```

```
is_required = False
```

```
property media
```

```
needs_multipart_form = False
```

optgroups(*name*, *value*, *attrs=None*)

Return a list of optgroups for this widget.

```
option_inherits_attrs = False
```

```
option_template_name = 'django/forms/widgets/select_option.html'
```

options(*name*, *value*, *attrs=None*)

Yield a flat list of options for this widgets.

render(*name*, *value*, *attrs=None*, *choices=()*, *renderer=None*)

Render the widget as an HTML string.

subwidgets(*name*, *value*, *attrs=None*)

Yield all "subwidgets" of this widget. Used to enable iterating options from a BoundField for choice widgets.

```
supports_microseconds = True
```

```
template_name = 'django/forms/widgets/select.html'
```

use_required_attribute(*initial*)

Don't render 'required' if the first <option> has a value, as that's invalid HTML.

value_from_datadict(*data*, *files*, *name*)

Given a dictionary of data and this widget's name, return the value of this widget or None if it's not provided.

value_omitted_from_data(*data*, *files*, *name*)

class tags_input.widgets.**TagsInputWidgetBase**(*on_add_tag=None*, *on_remove_tag=None*, *on_change_tag=None*, **args*, ***kwargs*)

Bases: SelectMultiple

add_id_index = False

allow_multiple_selected = True

build_attrs(*base_attrs*, *extra_attrs=None*, ***kwargs*)

Compatibility function for the behavior changes in Django 1.11+

checked_attribute = {'selected': True}

create_option(*name*, *value*, *label*, *selected*, *index*, *subindex=None*, *attrs=None*)

format_value(*value*)

Return selected values as a list.

get_context(*name*, *value*, *attrs*)

id_for_label(*id_*, *index='0'*)

Use an incremented id for each option where the main widget references the zero index.

input_type = 'select'

property is_hidden

is_localized = False

is_required = False

property media

needs_multipart_form = False

optgroups(*name*, *value*, *attrs=None*)

Return a list of optgroups for this widget.

option_inherits_attrs = False

option_template_name = 'django/forms/widgets/select_option.html'

options(*name*, *value*, *attrs=None*)

Yield a flat list of options for this widgets.

render(*name*, *value*, *attrs=None*, *choices=()*, *renderer=None*)

Render the widget as an HTML string.

subwidgets(*name*, *value*, *attrs=None*)

Yield all "subwidgets" of this widget. Used to enable iterating options from a BoundField for choice widgets.

supports_microseconds = True

template_name = 'django/forms/widgets/select.html'

use_required_attribute(*initial*)

Don't render 'required' if the first <option> has a value, as that's invalid HTML.

value_from_datadict(*data*, *files*, *name*)

Given a dictionary of data and this widget's name, return the value of this widget or None if it's not provided.

value_omitted_from_data(*data*, *files*, *name*)

6.3 tags_input.admin module

class tags_input.admin.**TagsInputAdmin**(*model*, *admin_site*)

Bases: *TagsInputMixin*, ModelAdmin

action_checkbox(*obj*)

A list_display column containing a checkbox widget.

action_form

alias of ActionForm

actions = []

actions_on_bottom = False

actions_on_top = True

actions_selection_counter = True

add_form_template = None

add_view(*request*, *form_url=''*, *extra_context=None*)

autocomplete_fields = ()

change_form_template = None

change_list_template = None

change_view(*request*, *object_id*, *form_url=''*, *extra_context=None*)

changeform_view(*request*, *object_id=None*, *form_url=''*, *extra_context=None*)

changelist_view(*request*, *extra_context=None*)

The 'change list' admin view for this model.

check(***kwargs*)

checks_class

alias of ModelAdminChecks

construct_change_message(*request*, *form*, *formsets*, *add=False*)

Construct a JSON structure describing changes from a changed object.

date_hierarchy = None

delete_confirmation_template = None

delete_model(*request*, *obj*)

Given a model instance delete it from the database.

delete_queryset(*request*, *queryset*)

Given a queryset, delete it from the database.

delete_selected_confirmation_template = None

delete_view(*request*, *object_id*, *extra_context=None*)

exclude = None

fields = None

fieldsets = None

filter_horizontal = ()

filter_vertical = ()

form

alias of `ModelForm`

formfield_for_choice_field(*db_field*, *request*, ***kwargs*)

Get a form Field for a database Field that has declared choices.

formfield_for_dbfield(*db_field*, *request*, ***kwargs*)

Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

formfield_for_foreignkey(*db_field*, *request*, ***kwargs*)

Get a form Field for a ForeignKey.

formfield_for_manytomany(*db_field*, *request=None*, ***kwargs*)

Get a form Field for a ManyToManyField.

formfield_overrides = {}

get_action(*action*)

Return a given action from a parameter, which can either be a callable, or the name of a method on the ModelAdmin. Return is a tuple of (callable, name, description).

get_action_choices(*request*, *default_choices=[('', '---------')]*)

Return a list of choices for use in a form object. Each choice is a tuple (name, description).

get_actions(*request*)

Return a dictionary mapping the names of all actions for this ModelAdmin to a tuple of (callable, name, description) for each action.

get_autocomplete_fields(*request*)

Return a list of ForeignKey and/or ManyToMany fields which should use an autocomplete widget.

get_changeform_initial_data(*request*)

Get the initial form data from the request's GET params.

get_changelist(*request*, ***kwargs*)

Return the ChangeList class for use on the changelist page.

get_changelist_form(*request*, ***kwargs*)

Return a Form class for use in the Formset on the changelist page.

get_changelist_formset(*request*, ***kwargs*)

Return a FormSet class for use on the changelist page if list_editable is used.

get_changelist_instance(*request*)

Return a *ChangeList* instance based on *request*. May raise *IncorrectLookupParameters*.

get_deleted_objects(*objs*, *request*)

Hook for customizing the delete process for the delete view and the "delete selected" action.

get_empty_value_display()

Return the empty_value_display set on ModelAdmin or AdminSite.

get_exclude(*request*, *obj=None*)

Hook for specifying exclude.

get_field_queryset(*db*, *db_field*, *request*)

If the ModelAdmin specifies ordering, the queryset should respect that ordering. Otherwise don't specify the queryset, let the field decide (return None in that case).

get_fields(*request*, *obj=None*)

Hook for specifying fields.

get_fieldsets(*request*, *obj=None*)

Hook for specifying fieldsets.

get_form(*request*, *obj=None*, *change=False*, ***kwargs*)

Return a Form class for use in the admin add view. This is used by add_view and change_view.

get_formsets_with_inlines(*request*, *obj=None*)

Yield formsets and the corresponding inlines.

get_inline_formsets(*request*, *formsets*, *inline_instances*, *obj=None*)

get_inline_instances(*request*, *obj=None*)

get_inlines(*request*, *obj*)

Hook for specifying custom inlines.

get_list_display(*request*)

Return a sequence containing the fields to be displayed on the changelist.

get_list_display_links(*request*, *list_display*)

Return a sequence containing the fields to be displayed as links on the changelist. The list_display parameter is the list of fields returned by get_list_display().

get_list_filter(*request*)

Return a sequence containing the fields to be displayed as filters in the right sidebar of the changelist page.

get_list_select_related(*request*)

Return a list of fields to add to the select_related() part of the changelist items query.

get_model_perms(*request*)

Return a dict of all perms for this model. This dict has the keys `add`, `change`, `delete`, and `view` mapping to the True/False for each of those actions.

get_object(*request*, *object_id*, *from_field=None*)

Return an instance matching the field and value provided, the primary key is used if no field is provided. Return `None` if no match is found or the object_id fails validation.

get_ordering(*request*)

Hook for specifying field ordering.

get_paginator(*request*, *queryset*, *per_page*, *orphans=0*, *allow_empty_first_page=True*)

get_prepopulated_fields(*request*, *obj=None*)

Hook for specifying custom prepopulated fields.

get_preserved_filters(*request*)

Return the preserved filters querystring.

get_queryset(*request*)

Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

get_readonly_fields(*request*, *obj=None*)

Hook for specifying custom readonly fields.

get_search_fields(*request*)

Return a sequence containing the fields to be searched whenever somebody submits a search query.

get_search_results(*request*, *queryset*, *search_term*)

Return a tuple containing a queryset to implement the search and a boolean indicating if the results may contain duplicates.

get_sortable_by(*request*)

Hook for specifying which fields can be sorted in the changelist.

get_tag_fields()

Get a list fo fields on this model that could be potentially tagged.

By default reads self.tag_fields if it exists of returns None for default behavious.

get_urls()

get_view_on_site_url(*obj=None*)

has_add_permission(*request*)

Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

has_change_permission(*request*, *obj=None*)

Return True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to change the *obj* model instance. If *obj* is None, this should return True if the given request has permission to change *any* object of the given type.

has_delete_permission(*request*, *obj=None*)

Return True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to delete the *obj* model instance. If *obj* is None, this should return True if the given request has permission to delete *any* object of the given type.

has_module_permission(*request*)

Return True if the given request has any permission in the given app label.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to view the module on the admin index page and access the module's index page. Overriding it does not restrict access to the add, change or delete views. Use *ModelAdmin.has_(add|change|delete)_permission* for that.

has_view_or_change_permission(*request*, *obj=None*)

has_view_permission(*request*, *obj=None*)

Return True if the given request has permission to view the given Django model instance. The default implementation doesn't examine the *obj* parameter.

If overridden by the user in subclasses, it should return True if the given request has permission to view the *obj* model instance. If *obj* is None, it should return True if the request has permission to view any object of the given type.

history_view(*request*, *object_id*, *extra_context=None*)

The 'history' admin view for this model.

inlines = []

list_display = ('__str__',)

list_display_links = ()

list_editable = ()

list_filter = ()

list_max_show_all = 200

list_per_page = 100

list_select_related = False

log_addition(*request*, *object*, *message*)

Log that an object has been successfully added.

The default implementation creates an admin LogEntry object.

log_change(*request*, *object*, *message*)

Log that an object has been successfully changed.

The default implementation creates an admin LogEntry object.

log_deletion(*request*, *object*, *object_repr*)

Log that an object will be deleted. Note that this method must be called before the deletion.

The default implementation creates an admin LogEntry object.

lookup_allowed(*lookup*, *value*)

property media

message_user(*request*, *message*, *level=20*, *extra_tags=''*, *fail_silently=False*)

Send a message to the user. The default implementation posts a message using the django.contrib.messages backend.

Exposes almost the same API as messages.add_message(), but accepts the positional arguments in a different order to maintain backwards compatibility. For convenience, it accepts the *level* argument as a string rather than the usual level number.

object_history_template = None

ordering = None

paginator

alias of `Paginator`

popup_response_template = None

prepopulated_fields = {}

preserve_filters = True

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

render_change_form(*request*, *context*, *add=False*, *change=False*, *form_url=''*, *obj=None*)

render_delete_form(*request*, *context*)

response_action(*request*, *queryset*)

Handle an admin action. This is called if a request is POSTed to the changelist; it returns an HttpResponse if the action was handled, and None otherwise.

response_add(*request*, *obj*, *post_url_continue=None*)

Determine the HttpResponse for the add_view stage.

response_change(*request*, *obj*)

Determine the HttpResponse for the change_view stage.

response_delete(*request*, *obj_display*, *obj_id*)

Determine the HttpResponse for the delete_view stage.

response_post_save_add(*request*, *obj*)

Figure out where to redirect after the 'Save' button has been pressed when adding a new object.

response_post_save_change(*request*, *obj*)

Figure out where to redirect after the 'Save' button has been pressed when editing an existing object.

save_as = False

save_as_continue = True

save_form(*request*, *form*, *change*)

Given a ModelForm return an unsaved instance. `change` is True if the object is being changed, and False if it's being added.

save_formset(*request*, *form*, *formset*, *change*)

Given an inline formset save it to the database.

save_model(*request*, *obj*, *form*, *change*)

Given a model instance save it to the database.

save_on_top = False

save_related(*request*, *form*, *formsets*, *change*)

Given the `HttpRequest`, the parent `ModelForm` instance, the list of inline formsets and a boolean value based on whether the parent is being added or changed, save the related objects to the database. Note that at this point save_form() and save_model() have already been called.

search_fields = ()

show_full_result_count = True

sortable_by = None

to_field_allowed(*request*, *to_field*)

Return True if the model associated with this admin should be allowed to be referenced by the specified field.

property urls

view_on_site = True

class tags_input.admin.**TagsInputMixin**

Bases: `object`

formfield_for_manytomany(*db_field*, *request=None*, ***kwargs*)

Get a form Field for a ManyToManyField.

get_tag_fields()

Get a list fo fields on this model that could be potentially tagged.

By default reads self.tag_fields if it exists of returns None for default behavious.

class tags_input.admin.**TagsInputStackedInline**(*parent_model*, *admin_site*)

Bases: *TagsInputMixin*, `StackedInline`

autocomplete_fields = ()

can_delete = True

check(***kwargs*)

checks_class

alias of `InlineModelAdminChecks`

classes = None

exclude = None

extra = 3

fields = None

fieldsets = None

filter_horizontal = ()

filter_vertical = ()

fk_name = None

form

alias of ModelForm

formfield_for_choice_field(*db_field*, *request*, ***kwargs*)

Get a form Field for a database Field that has declared choices.

formfield_for_dbfield(*db_field*, *request*, ***kwargs*)

Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

formfield_for_foreignkey(*db_field*, *request*, ***kwargs*)

Get a form Field for a ForeignKey.

formfield_for_manytomany(*db_field*, *request=None*, ***kwargs*)

Get a form Field for a ManyToManyField.

formfield_overrides = {}

formset

alias of BaseInlineFormSet

get_autocomplete_fields(*request*)

Return a list of ForeignKey and/or ManyToMany fields which should use an autocomplete widget.

get_empty_value_display()

Return the empty_value_display set on ModelAdmin or AdminSite.

get_exclude(*request*, *obj=None*)

Hook for specifying exclude.

get_extra(*request*, *obj=None*, ***kwargs*)

Hook for customizing the number of extra inline forms.

get_field_queryset(*db*, *db_field*, *request*)

If the ModelAdmin specifies ordering, the queryset should respect that ordering. Otherwise don't specify the queryset, let the field decide (return None in that case).

get_fields(*request*, *obj=None*)

Hook for specifying fields.

get_fieldsets(*request*, *obj=None*)

Hook for specifying fieldsets.

get_formset(*request*, *obj=None*, ***kwargs*)

Return a BaseInlineFormSet class for use in admin add/change views.

get_inlines(*request*, *obj*)

Hook for specifying custom inlines.

get_max_num(*request*, *obj=None*, ***kwargs*)

Hook for customizing the max number of extra inline forms.

get_min_num(*request*, *obj=None*, ***kwargs*)

Hook for customizing the min number of inline forms.

get_ordering(*request*)

Hook for specifying field ordering.

get_prepopulated_fields(*request*, *obj=None*)

Hook for specifying custom prepopulated fields.

get_queryset(*request*)

Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

get_readonly_fields(*request*, *obj=None*)

Hook for specifying custom readonly fields.

get_sortable_by(*request*)

Hook for specifying which fields can be sorted in the changelist.

get_tag_fields()

Get a list fo fields on this model that could be potentially tagged.

By default reads self.tag_fields if it exists of returns None for default behavious.

get_view_on_site_url(*obj=None*)

has_add_permission(*request*, *obj*)

Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

has_change_permission(*request*, *obj=None*)

Return True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to change the *obj* model instance. If *obj* is None, this should return True if the given request has permission to change *any* object of the given type.

has_delete_permission(*request*, *obj=None*)

Return True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to delete the *obj* model instance. If *obj* is None, this should return True if the given request has permission to delete *any* object of the given type.

has_module_permission(*request*)

Return True if the given request has any permission in the given app label.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to view the module on the admin index page and access the module's index page. Overriding it does not restrict access to the add, change or delete views. Use *ModelAdmin.has_(add|change|delete)_permission* for that.

has_view_or_change_permission(*request*, *obj=None*)

has_view_permission(*request*, *obj=None*)

Return True if the given request has permission to view the given Django model instance. The default implementation doesn't examine the *obj* parameter.

If overridden by the user in subclasses, it should return True if the given request has permission to view the *obj* model instance. If *obj* is None, it should return True if the request has permission to view any object of the given type.

lookup_allowed(*lookup*, *value*)

max_num = None

property media

min_num = None

model = None

ordering = None

prepopulated_fields = {}

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

show_change_link = False

show_full_result_count = True

sortable_by = None

template = 'admin/edit_inline/stacked.html'

to_field_allowed(*request*, *to_field*)

Return True if the model associated with this admin should be allowed to be referenced by the specified field.

verbose_name = None

verbose_name_plural = None

view_on_site = True

class tags_input.admin.**TagsInputTabularInline**(*parent_model*, *admin_site*)

Bases: *TagsInputMixin*, TabularInline

autocomplete_fields = ()

can_delete = True

check(***kwargs*)

checks_class

alias of InlineModelAdminChecks

classes = None

exclude = None

extra = 3

fields = None

fieldsets = None

filter_horizontal = ()

filter_vertical = ()

fk_name = None

form

alias of `ModelForm`

formfield_for_choice_field(*db_field*, *request*, ***kwargs*)

Get a form Field for a database Field that has declared choices.

formfield_for_dbfield(*db_field*, *request*, ***kwargs*)

Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

formfield_for_foreignkey(*db_field*, *request*, ***kwargs*)

Get a form Field for a ForeignKey.

formfield_for_manytomany(*db_field*, *request=None*, ***kwargs*)

Get a form Field for a ManyToManyField.

formfield_overrides = {}

formset

alias of `BaseInlineFormSet`

get_autocomplete_fields(*request*)

Return a list of ForeignKey and/or ManyToMany fields which should use an autocomplete widget.

get_empty_value_display()

Return the empty_value_display set on ModelAdmin or AdminSite.

get_exclude(*request*, *obj=None*)

Hook for specifying exclude.

get_extra(*request*, *obj=None*, ***kwargs*)

Hook for customizing the number of extra inline forms.

get_field_queryset(*db*, *db_field*, *request*)

If the ModelAdmin specifies ordering, the queryset should respect that ordering. Otherwise don't specify the queryset, let the field decide (return None in that case).

get_fields(*request*, *obj=None*)

Hook for specifying fields.

get_fieldsets(*request*, *obj=None*)

Hook for specifying fieldsets.

get_formset(*request*, *obj=None*, ***kwargs*)

Return a BaseInlineFormSet class for use in admin add/change views.

get_inlines(*request*, *obj*)

Hook for specifying custom inlines.

get_max_num(*request*, *obj=None*, ***kwargs*)

Hook for customizing the max number of extra inline forms.

get_min_num(*request*, *obj=None*, ***kwargs*)

Hook for customizing the min number of inline forms.

get_ordering(*request*)

Hook for specifying field ordering.

get_prepopulated_fields(*request*, *obj=None*)

Hook for specifying custom prepopulated fields.

get_queryset(*request*)

Return a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

get_readonly_fields(*request*, *obj=None*)

Hook for specifying custom readonly fields.

get_sortable_by(*request*)

Hook for specifying which fields can be sorted in the changelist.

get_tag_fields()

Get a list fo fields on this model that could be potentially tagged.

By default reads self.tag_fields if it exists of returns None for default behavious.

get_view_on_site_url(*obj=None*)

has_add_permission(*request*, *obj*)

Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

has_change_permission(*request*, *obj=None*)

Return True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to change the *obj* model instance. If *obj* is None, this should return True if the given request has permission to change *any* object of the given type.

has_delete_permission(*request*, *obj=None*)

Return True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to delete the *obj* model instance. If *obj* is None, this should return True if the given request has permission to delete *any* object of the given type.

has_module_permission(*request*)

Return True if the given request has any permission in the given app label.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to view the module on the admin index page and access the module's index

page. Overriding it does not restrict access to the add, change or delete views. Use *ModelAdmin.has_(add|change|delete)_permission* for that.

has_view_or_change_permission(*request*, *obj=None*)

has_view_permission(*request*, *obj=None*)

Return True if the given request has permission to view the given Django model instance. The default implementation doesn't examine the *obj* parameter.

If overridden by the user in subclasses, it should return True if the given request has permission to view the *obj* model instance. If *obj* is None, it should return True if the request has permission to view any object of the given type.

lookup_allowed(*lookup*, *value*)

max_num = None

property media

min_num = None

model = None

ordering = None

prepopulated_fields = {}

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

show_change_link = False

show_full_result_count = True

sortable_by = None

template = 'admin/edit_inline/tabular.html'

to_field_allowed(*request*, *to_field*)

Return True if the model associated with this admin should be allowed to be referenced by the specified field.

verbose_name = None

verbose_name_plural = None

view_on_site = True

6.4 tags_input.urls module

6.5 tags_input.utils module

tags_input.utils.**filter_func**(*fields*, *separator*, *values*)

tags_input.utils.**get_mapping**(*model_or_queryset*)

Get the mapping for a given model or queryset

tags_input.utils.**get_mappings**()

Get all mappings from the settings

To use the Django Tags Input module the *TAGS_INPUT_SETTINGS* must be defined.

tags_input.utils.**join_func**(*fields*, *separator*, *values*)

tags_input.utils.**split_func**(*fields*, *separator*, *value*)

6.6 tags_input.views module

tags_input.views.**autocomplete**(*request*, *app*, *model*, *fields*)

tags_input.views.**get_model**(*app*, *model*)

6.7 tags_input.exceptions module

exception tags_input.exceptions.**ConfigurationError**

Bases: Exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception tags_input.exceptions.**MappingUndefined**

Bases: *TagsInputError*

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception tags_input.exceptions.**TagsInputError**

Bases: Exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W