
Django Admin Kit Documentation

Release 0.0.2

Rohan Poojary

Sep 02, 2021

Table of Contents

1	Compatibility	3
2	Table of Contents	5
2.1	Getting Started	5
2.2	Documentation	8
2.3	Contents	15
2.4	Index	18
2.5	Python Module Index	18
3	Liscense	19
	Python Module Index	21
	Index	23

Django Admin Kit adds more functionality to admin page. This includes easier **Ajax Bindings**, **Different Widgets** and a feature to **Duplicate models** in admin site.

CHAPTER 1

Compatibility

This is compatible with [Django](#) (version 1.11+) and [Python 3.5+](#)

2.1 Getting Started

2.1.1 Installation

The module can be downloaded using python pip.

Command: `pip install django-admin-kit`

2.1.2 Configuration

The app name `admin_kit` should be put at the top of installed apps in django settings file.

```
# settings.py

INSTALLED_APPS = [
    'admin_kit',

    'django.contrib.admin',
    'django.contrib.auth',
    ...
]
```

This is because, Admin Kit overrides Django *change_form* template. Then register the `admin_kit` app in root urls file with name `admin_kit`

```
# urls.py

from django.conf.urls import url
import admin_kit

urlpatterns = [
```

(continues on next page)

(continued from previous page)

```
...
url(r'^admin_kit/', admin_kit.site.urls, name="admin_kit"),
]
```

Start the server and hit `/admin_kit/ping` url response. You will get a PONG response if configured correctly.


This *ping* url is enabled only in DEBUG Mode

2.1.3 Features

We will have a walk through of different features that Admin Kit provides.

Duplicate Button

This is a default feature that is added right after successful configuration of the app.

button.png 

This button is similar to `Add Another` button, but it initializes the fields with previously filled data. It is also compatible with `django-nested-admin`

To disable this feature set `KIT_DISABLE_DUPLICATE = True` in settings file.

Note: The duplicate button is only on **Inline Admin Models** like `Staked Inline`, `Tabular Inline` or `nested_admin` fields.

Multi Select Field

Admin Kit provides Multi Select field where you can specify choices. It uses `admin_kit.models.MultiSelectField`.

In `models.py` file

```
# models.py

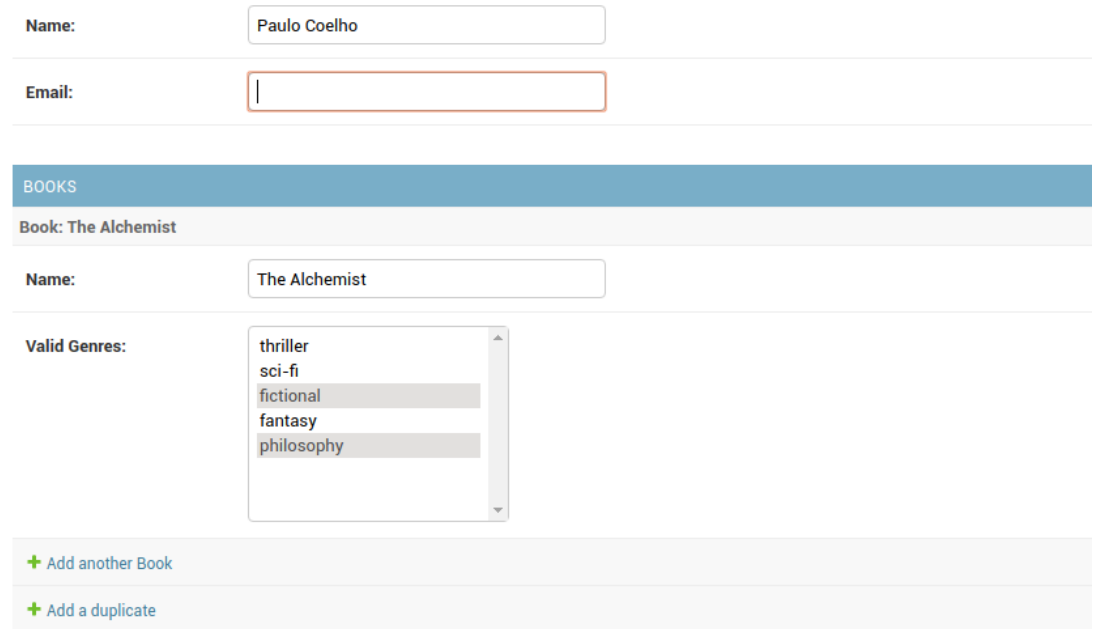
from admin_kit.models import MultiSelectField

class Book(models.Model):
    ...
    GENRES = (
        ('thriller', 'thriller'),
        ('sci-fi', 'sci-fi'),
        ('fictional', 'fictional'),
        ('fantasy', 'fantasy'),
        ('philosophy', 'philosophy')
    )
    ...

    genres = MultiSelectField(verbose_name='Valid Genres', choices=GENRES)
```

In Admin Panel

Change author



Name: Paulo Coelho

Email:

BOOKS

Book: The Alchemist

Name: The Alchemist

Valid Genres:

- thriller
- sci-fi
- fictional
- fantasy
- philosophy

+ Add another Book

+ Add a duplicate

select field.png

Ajax Binding

The core feature of Admin-Kit is the support for easier ajax behaviour. It binds the form-field with user defined view through ajax.

Setting up this behaviour is 2 step process.

- **Step 1: API Creation** Create an `ajax.py` file in the app. And create a class that inherits `admin_kit.ajax.Ajax` and has `run(self, request)` method. This method is executed, which acts as an API.

And register this class using `admin_kit.site.register` method. The first argument is the key through which the model links to class and second is the class itself.

For our example lets fill the choices from an API. Create an `ajax.py` with below code.

```
import admin_kit

class GenresAjax(admin_kit.ajax.Ajax):

    def run(self, request):
        GENRES = (
            ('thriller', 'thriller'),
            ('sci-fi', 'sci-fi'),
            ('fictional', 'fictional'),
            ('fantasy', 'fantasy'),
            ('philosophy', 'philosophy')
        )
        return GENRES

admin_kit.site.register('genres', GenresAjax)
```

Internally, the return type of `run` method is json formatted and acts as an API.

You can get the response by hitting `admin_kit/ajax/genres`. Here `genres` in the url is same as the key name used for registering in `ajax.py` file.

```
[
  - [
    "thriller",
    "thriller"
  ],
  - [
    "sci-fi",
    "sci-fi"
  ],
  - [
    "fictional",
    "fictional"
  ],
  - [
    "fantasy",
    "fantasy"
  ],
  - [
    "philosophy",
    "philosophy"
  ]
]
```

data.png

The data was rendered by Chrome Extension [JSON View](#)

- **Step 2: Model Binding**

In our `models.py` file modify `genres` field with below code

```
genres = MultiSelectField(verbose_name='Valid Genres', ajax_source=
↪ 'genres')
```

And thats it!! you will get the same behaviour, but now the choices are filled from your function. For every change in value, it calls `run` method from your `ajax` class. Thus you can process the return based on the request.

You can also access the user selected values and target the values to a specific field. To learn them please go through our [documentation](#)

2.1.4 Gotchas

- While using `ajax` behaviour make sure the model field is from `admin_kit.fields`. If you try to use `ajax` attributes like `ajax_source` or `kit_config` in fields from `django.models`, you will get an error
- As the project is new, currently it only has `MultiSelectField`. In further releases, newer fields will be integrated.

2.2 Documentation

The documentation consists of mainly five parts.

2.2.1 Model Fields

The documentation for `admin_kit.models` module. Currently, the module provides only one field, new fields will be added in future releases.

MultiSelectField

`MultiSelectField` provides Multi Selecting features. It internally by default stores in a `TextField` and the values are separated by `,`.

Parameters

It accepts all the `django model parameters`. Below are the additional parameters or special behaviour for a parameter.

- **separator** The default is `,`. This value will be used as a delimiter while storing the selected values.
- **max_length** Default is `null`. If this is set, then `varchar(max_length)` will be used for storage by default `TextField` is used for storage.
- **choices** The choices used for storage. This field is optional as the choices can be from an ajax class
- **ajax_source** The key name used while registering inherited `admin_kit.ajax.Ajax` class. The return of its `run` method will be used as choices.
- **ajax_target** This will be of the form `key_name:target_field`. On every change in its input, it hits the inherited `admin_kit.ajax.Ajax` class mapped to the specified `key_name`. If `target_field` is specified then it sets the value of specified model field to api's return value.

The selected values will be passed as `q[]` query list parameter.

Example

```
# ajax.py

import admin_kit

class SelectedGenresAjax(admin_kit.ajax.Ajax):
    response_type = 'text'

    def run(self, request):
        query = request.GET.getlist('q[]')
        response = ','.join(query)
        return response

admin_kit.site.register('selected-genres', SelectedGenresAjax)
```

This ajax class, joins query list `q[]` values with `,` and returns it in text format.

```
# models.py

from admin_kit.models import MultiSelectField

class Book(models.Model):
    name = models.CharField(max_length=100)
    GENRES = (
```

(continues on next page)

(continued from previous page)

```
( 'thriller', 'thriller'),
( 'sci-fi', 'sci-fi'),
( 'fictional', 'fictional'),
( 'fantasy', 'fantasy'),
( 'philosophy', 'philosophy'
)

genres = MultiSelectField(verbose_name='Valid Genres',
↪choices=GENRES,
                                ajax_target='selected-
↪genres:selectedValues')
selectedValues = models.TextField(verbose_name='Selected Values')
```

Hence for every change in genres field, the selected values will be sent to ajax class mapped to key: selected-genres which is SelectedAjax and its return will be filled to selectedValues field.

As you can notice the target_field of ajax_target parameter need not be from `admin_kit.models` module.

- **ajax_subscribe** This parameter is paired with ajax_source parameter and is set to False. If it is True, then this field is linked to every other field with its ajax_source value same as its linked ajax_target value.

But it wont be linked to fields which have target_field specified in its ajax_target parameter.

Example

```
# ajax.py

class GenresAjax (ajax.Ajax):
    def run(self, request):
        query = request.GET.getlist('q[]')
        response = list(zip(query, query))
        return response
```

This ajax class zips the selected query and returns it back.

```
# models.py

class Book (models.Model):
    ...
    genres = MultiSelectField(verbose_name='Valid Genres',
↪choices=GENRES,
                                ajax_target='genres')
    chosen_fields = MultiSelectField(seperator=',', ajax_source=
↪'genres',
                                ajax_subscribe=True)
```

Here chosen fields will have choices dynamically filled whenever genres field is modified. And the choices for chosen_fields will be from return of the GenresAjax class.

- **kit_config** This defaults to None. Instead of passing ajax_source, ajax_target and ajax_subscribe seperately, one can specify them in a dictionary and can be passed to this parameter.

Example

```
# models.py

class Book(models.Model):
    ...
    genres = MultiSelectField(verbose_name='Valid Genres',
                              ajax_source='genres', ajax_
↪ subscribe=True,
                              ajax_target='selected-
↪ genres:selectedValues')
```

Is equivalent to

```
# models.py

class Book(models.Model):
    ...
    kit_config = {
        'ajax-source': 'genres',
        'ajax-subscribe': True,
        'ajax-target': 'selected-genres:selectedValues'
    }
    genres = MultiSelectField(verbose_name='Valid Genres',
↪ choices=GENRES,
                              kit_config=kit_config)
```

Note: Make sure the *key names* are **hiphen seperated** and not *underscore* seperated.

SelectField

SelectField provides Selecting features. It has similar behaviour as *MultiSelectField*. So below are the valid parameters

Parameters

- max_length
- choices
- ajax_source
- ajax_target
- ajax_subscribe
- kit_config

2.2.2 Form Fields

The documentation for *admin_kit.fields* module.

MultiSelect Form Field

MultiSelectField provides Multi Selecting features. It is same as *models.MultiSelectField*, but is used in Django Admin Forms

Parameters

- **separator** The default is `,`. This value will be used as a delimiter while storing the selected values.
- **choices** The choices used for rendering. This field is optional as the choices can be from an ajax class
- **ajax_source** The key name used while registering inherited `admin_kit.ajax.Ajax` class. The return of its `run` method will be used as choices.
- **ajax_target** This will be of the form `key_name:target_field`. On every change in its input, it hits the inherited `admin_kit.ajax.Ajax` class mapped to the specified `key_name`. If `target_field` is specified then it sets the value of specified model field to api's return value.

The selected values will be passed as `q[]` query list parameter.

- **ajax_subscribe** This parameter is paired with `ajax_source` parameter and is set to `False`. If it is `True`, then this field is linked to every other field with its `ajax_source` value same as its linked `ajax_target` value.

But it wont be linked to fields which have `target_field` specified in its `ajax_target` parameter.

- **kit_config** This defaults to `None`. Instead of passing `ajax_source`, `ajax_target` and `ajax_subscribe` seperately, one can specify them in a dictionary and can be passed to this parameter.

Example

```
# admin.py

...
from admin_kit.fields import MultiSelectField

class BookForm(forms.ModelForm):
    model = Book
    selected_fields = MultiSelectField(ajax_source='genres', ajax_
↳ subscribe=True)
    fields = ('name', 'genres')

class BookAdmin(nested_admin.NestedStackedInline):
    model = Book
    form = BookForm
```

is equivalent to

```
# admin.py

...
from admin_kit.fields import MultiSelectField

class BookForm(forms.ModelForm):
    model = Book
    kit_config = {
        'ajax-source': 'genres',
        'ajax-subscribe': True
    }
    selected_fields = MultiSelectField(kit_config=kit_config)
```

(continues on next page)

(continued from previous page)

```
fields = ('name', 'genres')

class BookAdmin(nested_admin.NestedStackedInline):
    model = Book
    form = BookForm
```

Note: Make sure the key names are **hiphen seperated**.

Select Form Field

SelectField provides Selecting features. It is similar to *MultiSelectField*, but provides a single value to select.

Parameters

- choices
- ajax_source
- ajax_target
- ajax_subscribe
- kit_config

2.2.3 Form Widgets

The documentation for *admin_kit.widgets* module.

SelectMultipleWidget

MultiSelectField is the widget used for Multi Select. It inherits Django *SelectMultipleWidget* class.

The widget doesnt take any new parameters. It just adds the initial value of that widget to `data-kit-config` attribute.

SelectWidget

SelectWidget is the widget used for Select. It inherits Django *SelectWidget* class. This widget is similar to *MultiSelectField*.

2.2.4 Ajax Module

The documentation for *admin_kit.ajax* module. The module has only one class *Ajax*. So we will go through its attributes , methods and their functionality.

Attributes

- **response_type**
The response type of the ajax class. Its defaults to `json`, where its jsonifies the output python object. It also accepts `text` where it converts the output to a string and is sent as the response.
- **unique**
It defaults to `False`, if its `True` then the key will be unique to a class. Hence different Ajax classes with the same key can be registered.

Methods

- **run** (*self, request*)
The main method that will be executed for generating response for an Ajax request. This is method should be overided by the child class.

Note: The remainder methods are **internal**. So should be overridden only if necessary.

- **format_response** (*self, output*)
This method formats the return value of `run` method based on `response_type` attribute. If it is `json`, then it converts the output to json, else it renders it in `text` format.
- **route** (*self, request*)
This is the core function, that calls `run` method and then passes the output to `format_response` method and returns it. This method is executed when the `admin_kit site` figures out the `ajax_class` based on the request.
- **classmethod generate_key** (*cls, key*)
Generates the key based on the configuration of Ajax Class. If the `unique` attribute is set , it prepends the key with the slug form of its class name.

This method is called in the `register` function for key and `ajax_class` mapping.

Example

```
# ajax.py

from admin_kit import ajax

class TestAjax(ajax.Ajax):
    ...

class UniqueTestAjax(ajax.Ajax):
    unique = True
    ...

TestAjax.generate_key('key')
# `key`

UniqueTestAjax.generate_key('key')
# `unique-test-ajax-key`
```

To access this Ajax class in models, Its slugged key name has to be used. In the above Example to map to `UniqueTestAjax` class, `unique-test-ajax-key` key should be used in models file.

2.2.5 Sites Module

The documentation of `admin_kit.sites` module. The module has only class `AdminKitSite` which is the root site of the app.

This site object is aliased to `admin_kit.site`. So it can be accessed through the same

Site Methods

- **register** (*key*, *ajax_class*)

key :: str This is the key that will be used in models for binding

ajax_class :: class The ajax class that inherits `admin_kit.ajax.Ajax`

This method is used to bind an `ajax_class` to a key.

Note: If unique attribute of `ajax_class` is True, remember to prepend its slugname to the key.

2.3 Contents

2.3.1 Models

Admin Kit Models module

class `admin_kit.models.BaseField` (*kit_config=None*, *ajax_source=None*, *identifier=None*, *source_router=None*, *args, **kwargs)

The Base model field of Admin-Kit models. This inherits Django's `models.Field` class.

deconstruct ()

Deconstructs the field to a tuple of 4 elements. This is used to recreate the same object.

formfield (*form_class=None*, *choices_form_class=None*, **kwargs)

Returns the form object to be used for rendering.

from_db_value (*value*, *args, **kwargs)

Returns value from the database. inherited models should override this

validate (*value*, *model_instance*)

To validate the value of a model instance. Inherited models should override this

class `admin_kit.models.MultiSelectField` (*separator=''*, *args, **kwargs)

The Multiselect model field of Admin-Kit, which allows users to create multi select ajax fields.

db_type (*connection*)

Sets `db_type` to either `varchar` or `longtext` depending on `max_length`

deconstruct ()

Deconstructs MultiSelect Field

formfield (*form_class=None*, *choices_form_class=None*, **kwargs)

Sets form to be used for rendering

get_prep_value (*value*)

Converts value to a string

to_python (*value*)
Converts the string value to a list

value_to_string (*obj*)
Return a string value of this field from the passed obj. This is used by the serialization framework.

class admin_kit.models.**SelectField** (**args, **kwargs*)
The Select model field of Admin-Kit, which allows users to create select ajax fields.

db_type (*connection*)
Sets *db_type* to either varchar or longtext depending on *max_length*

deconstruct ()
Deconstructs SelectField

formfield (*form_class=None, choices_form_class=None, **kwargs*)
Sets form to be used for rendering

value_to_string (*obj*)
Return a string value of this field from the passed obj. This is used by the serialization framework.

2.3.2 Fields

Admin Kit Fields module

class admin_kit.fields.**BaseField** (*kit_config=None, ajax_source=None, default_value=None, identifier=None, source_router=None, *args, **kwargs*)

The Base Field for form fields

widget_attrs (*widget*)
This will add data-kit-config attribute to the widget

class admin_kit.fields.**MultiSelectField** (*separator=', ', choices=(), *args, **kwargs*)
This field is used to create MultiSelect Form fields.

widget
alias of *admin_kit.widgets.SelectMultipleWidget*

class admin_kit.fields.**SelectField** (*choices=(), *args, **kwargs*)
This field is used to create MultiSelect Form fields.

widget
alias of *admin_kit.widgets.SelectWidget*

2.3.3 Widgets

Admin Kit Widgets module

class admin_kit.widgets.**SelectMultipleWidget** (*attrs=None, choices=()*)
MultiSelect Widget which inherits Django's SelectMultiple widget

class Media
This class adds css required for admin_kit's widget

get_context (*name, value, attrs*)
Adds appropriate attributes to widget context

class admin_kit.widgets.**SelectWidget** (*attrs=None, choices=()*)
MultiSelect Widget which inherits Django's Select widget

get_context (*name, value, attrs*)
Adds appropriate attributes to widget context

2.3.4 Site

Admin Kit Sites module

class `admin_kit.sites.AdminKitSite` (*name='admin_kit'*)
The main AdminKitSite that routes and process url requests.

ajax (*request, key*)
Calls route method

get_urls ()
Returns the list of urls of admin_kit

js_config (*request*)
Renders the config.js file which configures global variables

ping (*request*)
Ping method is used to ping admin_kit ajax

register (*key, ajax_class*)
Registers the `ajax_class` for ajax behaviour

key :: str This is the *key* that will be used in models for binding

ajax_class :: class The ajax class that inherits `admin_kit.ajax.Ajax`

urls
The actual property used by django for routing requests

`admin_kit.site.register` (*key, ajax_class*)
Registers the `ajax_class` for ajax behaviour. This is same as `admin_kit.sites.AdminKitSite.register` method

key :: str This is the *key* that will be used in models for binding

ajax_class :: class The ajax class that inherits `admin_kit.ajax.Ajax`

2.3.5 Ajax

Admin Kit Ajax module

class `admin_kit.ajax.Ajax`
This is the base class for Ajax functionality.

response_type [str] The response type of the API. By default its set to `json`, It also accepts `text`.

unique [bool] If True, the *key* is prepended with class name slug, Thus making it unique per class.

format_response (*output*)
Formats the response type based on `response_type` attribute.

classmethod generate_key (*key*)
A class method that generates key, that maps to the function

If `unique` attribute is true, then it appends hipphen seperated class name to actual *key*

Example:

```
>>> import DummyAjaxClass
>>> DummyAjaxClass.generateKey('the_key')
the_key
>>> DummyAjaxClass.unique = True
>>> DummyAjaxClass.generateKey('the_key')
dummy-ajax-class-the_key
```

route (*request*, ***kwargs*)

For a given request it executes the `run` method of the `module_cls` and returns the response

run (*request*, ***kwargs*)

This method should be overridden by the child class.

2.4 Index

2.5 Python Module Index

CHAPTER 3

License

The django code is licensed under [The MIT License](#). View the *license* file to under the root directory for complete license and copyright information.

a

`admin_kit.ajax`, [17](#)
`admin_kit.fields`, [16](#)
`admin_kit.models`, [15](#)
`admin_kit.sites`, [17](#)
`admin_kit.widgets`, [16](#)

A

`admin_kit.ajax` (module), 17
`admin_kit.fields` (module), 16
`admin_kit.models` (module), 15
`admin_kit.sites` (module), 17
`admin_kit.widgets` (module), 16
`AdminKitSite` (class in `admin_kit.sites`), 17
`Ajax` (class in `admin_kit.ajax`), 17
`ajax()` (`admin_kit.sites.AdminKitSite` method), 17

B

`BaseField` (class in `admin_kit.fields`), 16
`BaseField` (class in `admin_kit.models`), 15

D

`db_type()` (`admin_kit.models.MultiSelectField` method), 15
`db_type()` (`admin_kit.models.SelectField` method), 16
`deconstruct()` (`admin_kit.models.BaseField` method), 15
`deconstruct()` (`admin_kit.models.MultiSelectField` method), 15
`deconstruct()` (`admin_kit.models.SelectField` method), 16

F

`format_response()` (`admin_kit.ajax.Ajax` method), 17
`format_response()` (built-in function), 14
`formfield()` (`admin_kit.models.BaseField` method), 15
`formfield()` (`admin_kit.models.MultiSelectField` method), 15
`formfield()` (`admin_kit.models.SelectField` method), 16
`from_db_value()` (`admin_kit.models.BaseField` method), 15

G

`generate_key()`, 14

`generate_key()` (`admin_kit.ajax.Ajax` class method), 17
`get_context()` (`admin_kit.widgets.SelectMultipleWidget` method), 16
`get_context()` (`admin_kit.widgets.SelectWidget` method), 16
`get_prep_value()` (`admin_kit.models.MultiSelectField` method), 15
`get_urls()` (`admin_kit.sites.AdminKitSite` method), 17

J

`js_config()` (`admin_kit.sites.AdminKitSite` method), 17

M

`MultiSelectField` (class in `admin_kit.fields`), 16
`MultiSelectField` (class in `admin_kit.models`), 15

P

`ping()` (`admin_kit.sites.AdminKitSite` method), 17

R

`register()` (`admin_kit.sites.admin_kit.site` method), 17
`register()` (`admin_kit.sites.AdminKitSite` method), 17
`register()` (built-in function), 15
`response_type`, 14
`route()` (`admin_kit.ajax.Ajax` method), 18
`route()` (built-in function), 14
`run()` (`admin_kit.ajax.Ajax` method), 18
`run()` (built-in function), 14

S

`SelectField` (class in `admin_kit.fields`), 16
`SelectField` (class in `admin_kit.models`), 16

SelectMultipleWidget (class in *admin_kit.widgets*), [16](#)

SelectMultipleWidget.Media (class in *admin_kit.widgets*), [16](#)

SelectWidget (class in *admin_kit.widgets*), [16](#)

T

to_python() (*admin_kit.models.MultiSelectField* method), [15](#)

U

unique, [14](#)

urls (*admin_kit.sites.AdminKitSite* attribute), [17](#)

V

validate() (*admin_kit.models.BaseField* method), [15](#)

value_to_string() (*admin_kit.models.MultiSelectField* method), [16](#)

value_to_string() (*admin_kit.models.SelectField* method), [16](#)

W

widget (*admin_kit.fields.MultiSelectField* attribute), [16](#)

widget (*admin_kit.fields.SelectField* attribute), [16](#)

widget_attrs() (*admin_kit.fields.BaseField* method), [16](#)