
Django Suit Documentation

Release 0.2.25

Kaspars Sprogis (darklow)

Mar 30, 2017

Contents

| | | |
|----------|------------------------------|-----------|
| 1 | About | 3 |
| 2 | Licence | 5 |
| 3 | Resources | 7 |
| 4 | Preview | 9 |
| 5 | Getting Started | 11 |
| 5.1 | Getting Started | 11 |
| 5.2 | Configuration | 13 |
| 6 | Features | 19 |
| 6.1 | Widgets | 19 |
| 6.2 | Sortable | 22 |
| 6.3 | Form tabs | 27 |
| 6.4 | Form includes | 29 |
| 6.5 | List attributes | 30 |
| 6.6 | Wysiwyg editors | 32 |
| 6.7 | JavaScript and CSS | 34 |
| 7 | Support | 37 |
| 7.1 | Examples | 37 |
| 7.2 | Supported apps | 37 |
| 8 | Contributing | 39 |
| 8.1 | Contributing | 39 |

Django Suit - Modern theme for Django admin interface.

CHAPTER 1

About

Django Suit is alternative theme/skin/extension for [Django](#) admin app (administration interface).

CHAPTER 2

Licence

- Django Suit is licensed under Creative Commons Attribution-NonCommercial 3.0 license.
- See licence and pricing: <http://djangosuit.com/pricing/>

CHAPTER 3

Resources

- Home page: <http://djangosuit.com>
- Demo: <http://djangosuit.com/admin/>
- Licence and Pricing: <http://djangosuit.com/pricing/>
- Github: <https://github.com/darklow/django-suit>
- Demo app on Github: <https://github.com/darklow/django-suit-examples>
- Changelog: [Changelog.rst](#)
- Supports: Django 1.4-1.10. Python: 2.6-3.4
- *Supported apps*

CHAPTER 4

Preview

Click on screenshot for live demo:

Getting Started

Installation

1. You can get stable version of Django Suit by using pip or easy_install:

```
pip install django-suit==0.2.25
```

2. You will need to add the 'suit' application to the INSTALLED_APPS setting of your Django project settings.py file.:

```
INSTALLED_APPS = (  
    ...  
    'suit',  
    'django.contrib.admin',  
)
```

Important: 'suit' must be added before 'django.contrib.admin' and if you are using third-party apps with special admin support (like django-cms) you also need to add 'suit' before 'cms'.

3. **For Django < 1.9:** You need to add 'django.core.context_processors.request' to TEMPLATE_CONTEXT_PROCESSORS setting in your Django project settings.py file.:

```
from django.conf.global_settings import TEMPLATE_CONTEXT_PROCESSORS as TCP  
  
TEMPLATE_CONTEXT_PROCESSORS = TCP + (  
    'django.core.context_processors.request',  
)
```

For Django >= 1.9 or with new Django “TEMPLATES“ setting: Make sure you have django.template.context_processors.request in your TEMPLATES OPTIONS

context_processors setting in your Django project settings.py file.:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request', # Make sure_
↪you have this line
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]
```

Note: This is required to handle left side menu. If by some reason you removed original Django Suit menu.html, you can skip this.

Deployment

Deployment with Django Suit should not be different than any other Django application. If you have problems with deployment on production, read [Django docs on wsgi](#) first.

Note: If you deploy your project with Apache or `Debug=False` don't forget to run `./manage.py collectstatic`

Develop branch

[Develop branch](#) is considered as release candidate version. Check [commits](#) and [changelog](#) of develop branch first, before installing develop version. It is quite stable and always tested, but can contain some flaws or behaviour changes too. To install latest develop version use:

```
pip uninstall django-suit
pip install https://github.com/darklow/django-suit/tarball/develop
```

v2-dev branch

[v2-dev branch](#) is a complete rewrite using Bootstrap v4. It is still in development and this documentation doesn't match v2-dev. Read more about v2 [here](#). To install latest v2 version use:

```
pip uninstall django-suit
pip install https://github.com/darklow/django-suit/tarball/v2
```


Configuration

Templates

You must extend `base_site.html` template to customize footer links, copyright text or to add extra JS/CSS files. Example file is available on [github](#).

Copy customized `base_site.html` [template file](#) to your project's main application `templates/admin/` directory and un-comment and edit the blocks you would like to extend.

Alternatively you can copy `base_site.html` to any of template directories, which are defined in `TEMPLATE_DIRS` setting (if any). By default Django looks in every registered application `templates/` dir.

In the same way you can override any of Django Suit admin templates. More about customizing project's templates, you can read in [Django Admin Tutorial](#)

Customization

You can customize Django Suit behaviour by adding `SUIT_CONFIG` configuration variable to your Django project `settings.py` file.

```
SUIT_CONFIG = { 'PARAM': VALUE, 'PARAM2': VALUE2 ...
}
```

Default values are the ones specified in examples.

Full example

Configuration sample you can use as a start:

```
# Django Suit configuration example
SUIT_CONFIG = {
    # header
    # 'ADMIN_NAME': 'Django Suit',
    # 'HEADER_DATE_FORMAT': 'l, j. F Y',
    # 'HEADER_TIME_FORMAT': 'H:i',

    # forms
    # 'SHOW_REQUIRED_ASTERISK': True, # Default True
    # 'CONFIRM_UNSAVED_CHANGES': True, # Default True

    # menu
    # 'SEARCH_URL': '/admin/auth/user/',
    # 'MENU_ICONS': {
    #     'sites': 'icon-leaf',
    #     'auth': 'icon-lock',
    # },
    # 'MENU_OPEN_FIRST_CHILD': True, # Default True
    # 'MENU_EXCLUDE': ('auth.group',),
    # 'MENU': (
    #     'sites',
    #     {'app': 'auth', 'icon': 'icon-lock', 'models': ('user', 'group')},
    #     {'label': 'Settings', 'icon': 'icon-cog', 'models': ('auth.user', 'auth.group
    → ')},
    #     {'label': 'Support', 'icon': 'icon-question-sign', 'url': '/support/'},
```

```
# ),  
  
# misc  
# 'LIST_PER_PAGE': 15  
}
```

Header

Header related parameters

ADMIN_NAME

Admin name that will appear in header <title> tags and in footer:

```
SUIT_CONFIG = {  
    'ADMIN_NAME': 'Django Suit'  
}
```

HEADER_DATE_FORMAT

HEADER_TIME_FORMAT

Header date and time formats. Formatting according to Django date templatefilter format. Default: as specified in example:

```
SUIT_CONFIG = {  
    'HEADER_DATE_FORMAT': 'l, j. F Y', # Saturday, 16th March 2013  
    'HEADER_TIME_FORMAT': 'H:i',      # 18:42  
}
```

Forms

SHOW_REQUIRED_ASTERISK

Automatically adds asterisk symbol * to the end of every required field label:

```
SUIT_CONFIG = {  
    'SHOW_REQUIRED_ASTERISK': True  
}
```

CONFIRM_UNSAVED_CHANGES

Alert will be shown, when you'll try to leave page, without saving changed form first:

```
SUIT_CONFIG = {  
    'CONFIRM_UNSAVED_CHANGES': True  
}
```

Menu

SEARCH_URL

We have big plans for this field in the future, by making it global search field. However right now this field works only as a search redirect to any other urls of your admin:

```
SUIT_CONFIG = {
    'SEARCH_URL': '/admin/user',

    # Parameter also accepts url name
    'SEARCH_URL': 'admin:auth_user_changelist',

    # Set to empty string if you want to hide search from menu
    'SEARCH_URL': ''
}
```

MENU_OPEN_FIRST_CHILD

Automatically replaces app's (parent link) url with url of first model's url (child):

```
SUIT_CONFIG = {
    'MENU_OPEN_FIRST_CHILD': True
}
```

MENU_ICONS

Set app icons. Use any of Twitter Bootstrap [icon classes](#) or add your own. Twitter Bootstrap icons are provided by [Glyphicons](#). This parameter is useful, if you don't use MENU parameter (see below) and just want to set icons for default apps:

```
SUIT_CONFIG = {
    'MENU_ICONS': {
        'sites': 'icon-leaf',
        'auth': 'icon-lock',
    }
}
```

MENU_EXCLUDE

Exclude any of apps or models. You can exclude whole app or just one model from app:

```
SUIT_CONFIG = {
    'MENU_EXCLUDE': ('auth.group', 'auth'),
}
```

Note: This parameter excludes app/model only from menu, it doesn't protect from accessing it by url or from app list. Use django user permissions to securely protect app/model.

MENU_ORDER

`MENU_ORDER` parameter is deprecated - use `MENU` instead.

MENU

Most powerful of menu parameters - one parameter to rule them all :) You can rename, reorder, cross link, exclude apps and models, and even define custom menu items and child links.

Following keys are available for each app and model level links:

- **App:** app, label, url, icon, permissions, blank
- **Model:** model, label, url, permissions, blank
- **Use –** as separator between apps

url parameter can be:

- Absolute url like `/custom/`
- Named url like `admin:index`
- Model name like `auth.user` to make link to model changelist
- If `MENU_OPEN_FIRST_CHILD=True` and models for app exists, you can skip url key
- If you add key `'blank': True` links will open in new window

permissions are verified using `user.has_perms()` method.

A custom application can contain a `models` list (or tuple) to customize the application models list. The `models` list can contain model references and model definitions. The model reference is a string referencing to the model through the application label and model name. The model name may be globbed to reference all models in the application like `'auth.*'`.

Here is full example of `MENU` from simple existing app reorder to defining custom menu items:

```
SUIT_CONFIG = {
    'MENU': (

        # Keep original label and models
        'sites',

        # Rename app and set icon
        {'app': 'auth', 'label': 'Authorization', 'icon': 'icon-lock'},

        # Reorder app models
        {'app': 'auth', 'models': ('user', 'group')},

        # Custom app, with models
        {'label': 'Settings', 'icon': 'icon-cog', 'models': ('auth.user', 'auth.group
↪')},

        # Cross-linked models with custom name; Hide default icon
        {'label': 'Custom', 'icon': None, 'models': (
            'auth.group',
            {'model': 'auth.user', 'label': 'Staff'}
        )},

        # Custom app, no models (child links)
        {'label': 'Users', 'url': 'auth.user', 'icon': 'icon-user'},
```

```
    # Separator
    '-',

    # Custom app and model with permissions
    {'label': 'Secure', 'permissions': 'auth.add_user', 'models': [
        {'label': 'custom-child', 'permissions': ('auth.add_user', 'auth.add_group
↪')}
    ]},
)
}
```

List

LIST_PER_PAGE

Set `change_list` view `list_per_page` parameter globally for whole admin. You can still override this parameter in any `ModelAdmin` class:

```
SUIT_CONFIG = {
    'LIST_PER_PAGE': 20
}
```


Widgets

There are handy widgets included in Django Suit.

Widgets

Easiest way to change widget for specific fields is define your own `ModelForm` class with `Meta` class and widgets. Following this logic, you can override almost any widget in your form.

For example if you want to add some additional CSS class to input you can do following:

```
from django.forms import ModelForm, TextInput
from django.contrib.admin import ModelAdmin
from .models import Country

class CountryForm(ModelForm):
    class Meta:
        widgets = {
            'name': TextInput(attrs={'class': 'input-mini'})
        }

class CountryAdmin(ModelAdmin):
    form = CountryForm
    ...

admin.site.register(Country, CountryAdmin)
```

Note: If you define some custom fields for your form, you must specify `model = YourModel` parameter for class `Meta`.

NumberInput

HTML5 number input `type="number"`:

```
from django.forms import ModelForm
from suit.widgets import NumberInput

class OrderForm(ModelForm):
    class Meta:
        widgets = {
            'count': NumberInput,

            # Optionally you specify attrs too
            'count': NumberInput(attrs={'class': 'input-mini'})
        }
```

Note: The same result you can achieve with `HTML5Input(input_type='number')` widget, this is just a shortcut.

HTML5Input

Specify `input_type` and use any of [HTML5 input types](#). You can see some HTML5 input examples in Django Suit Demo app in [KitchenSink forms](#):

```
from django.forms import ModelForm
from suit.widgets import HTML5Input

class ProductForm(ModelForm):
    class Meta:
        widgets = {
            'color': HTML5Input(input_type='color'),
        }
```

Note: Not all major browsers support all the new input types. However, you can already start using them; If they are not supported, they will behave as regular text fields. Also not all types are supported by [Twitter Bootstrap](#).

EnclosedInput

Supports [Twitter Bootstrap prepended/appended form inputs](#). `EnclosedInput` widget accepts two arguments `prepend=` and `append=`. Values can be `text`, `icon-class` or `html` (starts with `html` tag). You can also use both `prepend=` and `append=` together:

```
from django.forms import ModelForm
from suit.widgets import EnclosedInput

class ProductForm(ModelForm):
    class Meta:
        widgets = {

            # Appended by text
```



```

'discount': EnclosedInput (append='%'),
'size': EnclosedInput (append='m<sup>2</sup>'),

# By icons
'email': EnclosedInput (append='icon-envelope'),
'user': EnclosedInput (prepend='icon-user'),

# You can also use prepended and appended together
'price': EnclosedInput (prepend='$', append='.00'),

# Use HTML for append/prepend (See Twitter Bootstrap docs of supported_
↪tags)
'url': EnclosedInput (prepend='icon-home', append='<input type="button"
↪class="btn" value="Open link">'),

}

```

Preview:

The preview shows two examples of the EnclosedInput widget. The first example is labeled 'Area:' and features a globe icon on the left, a text input field in the center, and the unit 'km²' on the right. The second example is labeled 'Population:' and features a person icon on the left, a text input field in the center, and a 'Search' button on the right.

AutosizedTextarea

AutosizedTextarea enables automatic height based on user input for textarea elements. Uses jQuery Autosize plugin. All the required javascript will be automatically attached. Usage:

```

from django.forms import ModelForm
from suit.widgets import AutosizedTextarea

class ProductForm (ModelForm):
    class Meta:
        widgets = {
            'description': AutosizedTextarea,

            # You can also specify html attributes
            'description': AutosizedTextarea (attrs={'rows': 3, 'class': 'input-xlarge
↪'}),
        }

```

For demo - see countries description field or kitchensink tabluar inlines

Note: If you want to change height, when vertical scrollbar appears, use CSS max-height attribute. By default: max-height: 150px

Date/Time widgets

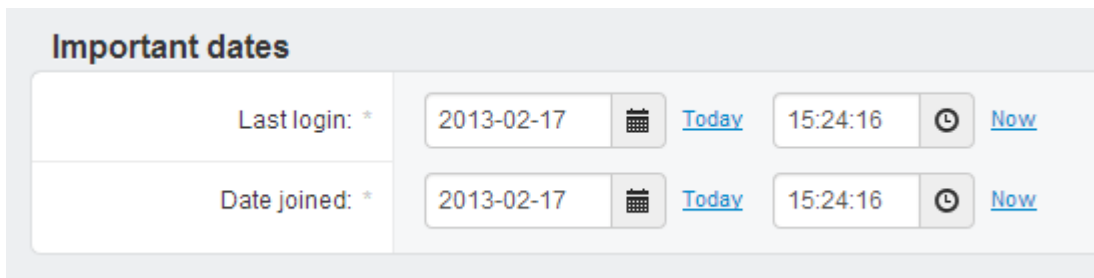
SuitDateWidget, SuitTimeWidget and SuitSplitDateTimeWidget extends original admin widgets by

adding some additional output styling only. Widgets still uses same original JavaScript for calendar and time. You can see example in Demo app: User changeform:

```
from django.forms import ModelForm
from suit.widgets import SuitDateWidget, SuitTimeWidget, SuitSplitDateTimeWidget

class UserChangeForm(ModelForm):
    class Meta:
        model = User
        widgets = {
            'last_login': SuitSplitDateTimeWidget,
            'date_joined': SuitSplitDateTimeWidget,
        }
```

Preview:



The preview shows a form titled "Important dates" with two rows. The first row is for "Last login: *" and the second for "Date joined: *". Each row contains a date field (2013-02-17), a calendar icon, a "Today" link, a time field (15:24:16), a clock icon, and a "Now" link.

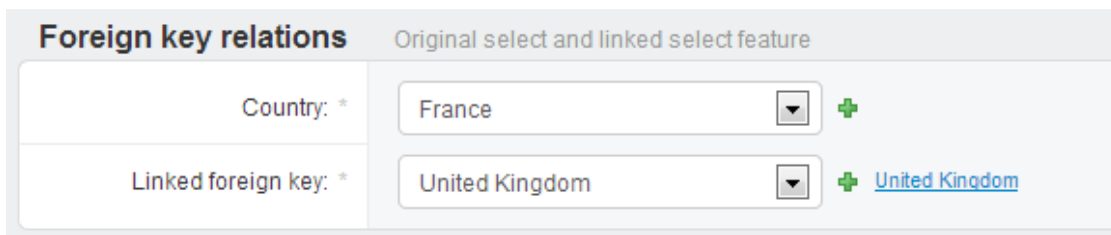
LinkedSelect

Adds link to related item right after select for foreign key fields:

```
from django.forms import ModelForm
from suit.widgets import LinkedSelect

class CountryForm(UserChangeForm):
    class Meta:
        widgets = {
            'continent': LinkedSelect
        }
```

Preview:



The preview shows a form titled "Foreign key relations" with a subtitle "Original select and linked select feature". It has two rows. The first row is for "Country: *" with a dropdown menu showing "France" and a plus sign. The second row is for "Linked foreign key: *" with a dropdown menu showing "United Kingdom" and a plus sign followed by a link to "United Kingdom".

Sortable

Sortable are handy admin tools for ordering different lists.

Sortable

Currently Django Suit supports these types of sortables:

1. Sortable for change list
2. Sortable for `django-mptt` tree list
3. Sortable for Tabular, Stacked, GenericTabular, GenericStacked inlines

Limitations

Since sortables are based on JavaScript solution, there are known limitations:

1. They don't work with pagination.
2. You won't be able to use different list order other than by sortable parameter.

Under the hood

Widgets add sortable parameter to `list_editable` fields as simple number inputs. Afterwards JavaScript utils replaces these editable inputs with arrows. Order is not saved instantly, but only when user saves form, which is very handy - user can do sorting first and afterwards save it or cancel if changed his mind.

Change list sortable

To use change list sortable you must do following:

1. In your `models.py` file add integer property for sortable to you model:

```
from django.db import models

class Continent(models.Model):
    ...
    order = models.PositiveIntegerField()
```

2. In your `admin.py` extend `SortableModelAdmin` class and specify sortable name:

```
from suit.admin import SortableModelAdmin

class ContinentAdmin(SortableModelAdmin):
    ...
    sortable = 'order'
```

That's it, you should see similar picture to example below in your admin now.

Note: If you want sortable arrows to appear in different column than last, you can do this by adding sortable field to `list_editable` in desired order, for example: `list_editable=('name', 'order', 'something')`. If you set arrows as first column, you must also define `list_display_links` - because arrows can't be displayed also as links.

Example

| <input type="checkbox"/> | Name | Order |
|--------------------------|---------------|-------|
| <input type="checkbox"/> | North America | ↑ ↓ |
| <input type="checkbox"/> | Australia | ↑ ↓ |
| <input type="checkbox"/> | South America | ↑ ↓ |
| <input type="checkbox"/> | Europe | ↑ ↓ |

Resources

- [Live example](#)
- [Github source](#)

django-mptt tree sortable

To use sortable on `django-mptt` tree, you must follow the same instructions as for change list sortable. Combining with documentation on `django-mptt`, final code should look like this:

1. Prepare your model in `models.py`

```
from django.db import models
from mptt.fields import TreeForeignKey
from mptt.models import MPTTModel

class Category(MPTTModel):
    name = models.CharField(max_length=64)
    parent = TreeForeignKey('self', null=True, blank=True,
                           related_name='children')

    # Sortable property
    order = models.PositiveIntegerField()

    class MPTTMeta:
        order_insertion_by = ['order']

    # It is required to rebuild tree after save, when using order for mptt-tree
    def save(self, *args, **kwargs):
        super(Category, self).save(*args, **kwargs)
        Category.objects.rebuild()

    def __unicode__(self):
        return self.name
```

2. Prepare admin class in `admin.py`:

```
from suit.admin import SortableModelAdmin
from mptt.admin import MPTTModelAdmin
from .models import Category
```

```

class CategoryAdmin(MPTTModelAdmin, SortableModelAdmin):
    mptt_level_indent = 20
    list_display = ('name', 'slug', 'is_active')
    list_editable = ('is_active',)

    # Specify name of sortable property
    sortable = 'order'

admin.site.register(Category, CategoryAdmin)

```

Note: MPTTModelAdmin must be specified “before” SortableModelAdmin in extend syntax as shown in example.

Example

| <input type="checkbox"/> | Name | Slug | Is active | Order |
|--------------------------|-------------------|-------------------|-------------------------------------|-------|
| <input type="checkbox"/> | Services | services | <input checked="" type="checkbox"/> | ↑ ↓ |
| <input type="checkbox"/> | Catalog | catalog | <input checked="" type="checkbox"/> | ↑ ↓ |
| <input type="checkbox"/> | What's new | whats-new | <input checked="" type="checkbox"/> | ↑ ↓ |
| <input type="checkbox"/> | Products | products | <input checked="" type="checkbox"/> | ↑ ↓ |
| <input type="checkbox"/> | Even nicer things | even-nicer-things | <input type="checkbox"/> | ↑ ↓ |
| <input type="checkbox"/> | Nice gadgets | nice-gadgets | <input checked="" type="checkbox"/> | ↑ ↓ |

Resources

- [Live example](#)
- [Github source](#)
- [django-mptt documentation](#)

Tabular inlines sortable

1. In `models.py` your model for inlines, should have integer property for sortable, same way as described in all previous sortable examples:

```

from django.db import models

class Country(models.Model):
    ...
    order = models.PositiveIntegerField()

```

2. In `admin.py` inline class must extend `SortableModelAdmin` class and specify sortable name:

```

from django.contrib.admin import ModelAdmin
from suit.admin import SortableTabularInline

class CountryInline(SortableTabularInline):
    model = Country
    sortable = 'order'

class ContinentAdmin(ModelAdmin):
    inlines = (CountryInline,)

```

That's it, you should see similar picture to example below in your admin now.

Example

| Name * | Code ☺ * | Population | Order * | Delete? |
|--|---------------------------------|----------------------|---------|--------------------------|
| <input type="text" value="Antarctica"/> | <input type="text" value="AQ"/> | <input type="text"/> | ↑ ↓ | <input type="checkbox"/> |
| <input type="text" value="French Southern Territories"/> | <input type="text" value="TF"/> | <input type="text"/> | ↑ ↓ | <input type="checkbox"/> |
| <input type="text" value="South Georgia and the South Sandwic"/> | <input type="text" value="GS"/> | <input type="text"/> | ↑ ↓ | <input type="checkbox"/> |
| <input type="text"/> | <input type="text"/> | <input type="text"/> | ↑ ↓ | |

[Add another Country](#)

Resources

- [Live example](#)
- [Live example #2](#)
- [Github source](#)

Stacked and Generic inlines sortable

Implementation of sortables for Stacked and Generic inlines is the same as mentioned above for Tabular inlines. You just have to use appropriate base class instead of `SortableTabularInline`:

```

# For Stacked inlines
from suit.admin import SortableStackedInline

# For Generic inlines
from suit.admin import SortableTabularStackedInline
from suit.admin import SortableGenericStackedInline

```

Example

Resources

- [Live example](#)
- [Github source](#)

Form tabs

Form tabs help you organize form fieldsets and inlines into tabs.

Form tabs

Form tabs help you organize form fieldsets and inlines into tabs. Before reading further, take a look on the tabs in the [demo app](#).

Under the hood: Tabs are based on mostly CSS/JS solution, therefore integration of tabs is simple and non intrusive - all your form handling will work the same as before.

To organize form into tabs you must:

1. Add `suit_form_tabs` parameter to your `ModelAdmin` class:

```
# (TAB_NAME, TAB_TITLE)
suit_form_tabs = (('general', 'General'), ('advanced', 'Advanced Settings'))
```

2. Add `'suit-tab suit-tab-TAB_NAME'` to fieldset classes, where `TAB_NAME` matches tab name you want to show fieldset in.
3. To use with inlines, specify same css classes in `suit_classes` parameter for inline classes

Example

```
from django.contrib import admin
from .models import Country

class CityInline(admin.TabularInline):
    model = City
    suit_classes = 'suit-tab suit-tab-cities'

class CountryAdmin(admin.ModelAdmin):
    inlines = (CityInline,)

    fieldsets = [
        (None, {
            'classes': ('suit-tab', 'suit-tab-general',),
            'fields': ['name', 'continent',]
        }),
        ('Statistics', {
            'classes': ('suit-tab', 'suit-tab-general',),
            'fields': ['area', 'population',]),
        ('Architecture', {
            'classes': ('suit-tab', 'suit-tab-cities',),
            'fields': ['architecture',]),
    ]

    suit_form_tabs = (('general', 'General'), ('cities', 'Cities'),
                    ('info', 'Info on tabs'))

    # Read about form includes in next section
    suit_form_includes = (
        ('admin/examples/country/custom_include.html', 'middle', 'cities'),
        ('admin/examples/country/tab_info.html', '', 'info'),
    )

admin.site.register(Country, CountryAdmin)
```

Same way you can organize any HTML into tabs, just wrap it in previously mentioned CSS classes:

```
<div class="suit-tab suit-tab-TAB_NAME">...</div>
```


Preview

Form Includes

Django Suit provides handy shortcut to include templates into forms.

See *Form Includes*

Form includes

Django Suit provides handy shortcut to include templates into forms.

Form Includes

Django Suit provides handy shortcut to include templates into forms for several positions (top, middle and bottom).

Under the hood: Suit includes are nothing but a shortcut. The same can be achieved by extending `change_form.html` and hooking into particular blocks. Suit includes can be used in combination with or without *Form tabs*.

Each `suit_form_includes` item can contain 3 parameters:

1. Path to template (Required)
2. Position: (Optional)
 - `top` - above fieldsets:
 - `middle` - between fieldsets and inlines
 - `bottom` - after inlines (Default)
3. Specify `TAB_NAME` if using in combination with *Form tabs* (Optional)

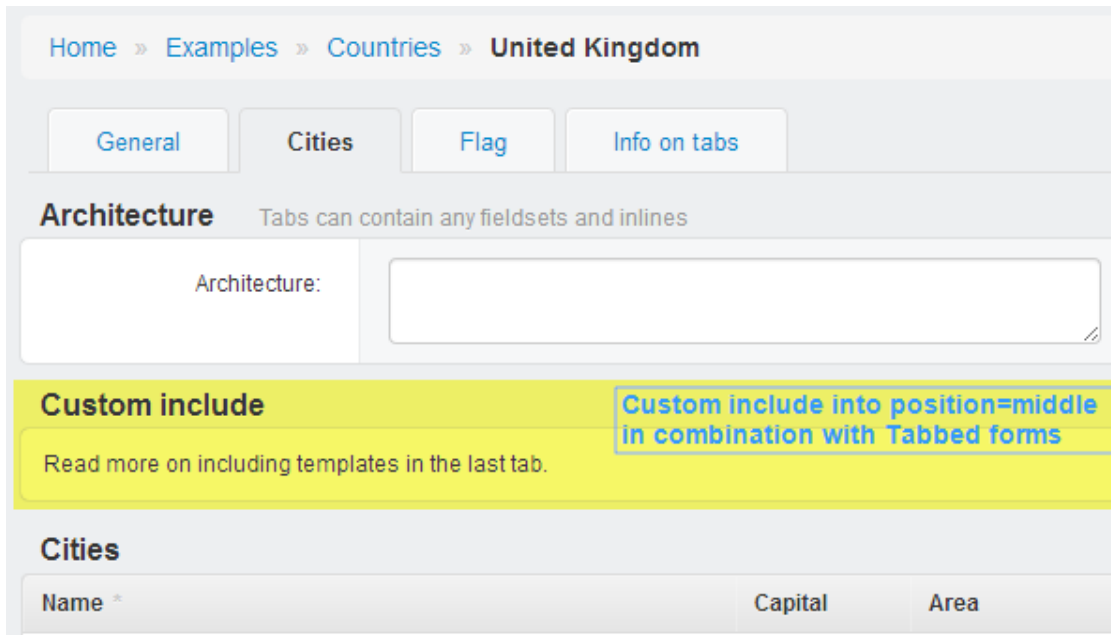
Example

```
from django.contrib import admin
from .models import Country

class CountryAdmin(admin.ModelAdmin):
    ...
    suit_form_includes = (
```

```
(
    ('admin/examples/country/custom_include.html', 'middle', 'cities'),
    ('admin/examples/country/tab_info.html', '', 'info'),
    ('admin/examples/country/disclaimer.html'),
)
```

Preview



List attributes

Using few callable helpers you can customize change list row and cell attributes based on object instance variables.

List attributes

Using few callable helpers you can customize change list row and cell attributes based on object instance variables.

List header columns

As soon as you add `suit` to your apps, all your changelist table header columns will have specific field CSS class present.

For example for column `country` list header tag will look like this `<th class="country-column">`. Which means you can change it's appearance using CSS.

```
.country-column .text, .country-column a {
    text-align: center;
}
```

List row attributes

To add html attributes like `class` or `data` to list rows, you must define `suit_row_attributes` callable (function). Callable receives object instance and the request as arguments and must return `dict` with attributes for current row.

Example:

```
from django.contrib.admin import ModelAdmin

class CountryAdmin(ModelAdmin):
    ...

    def suit_row_attributes(self, obj, request):
        return {'class': 'type-%s' % obj.type}

    # Or bit more advanced example
    def suit_row_attributes(self, obj, request):
        css_class = {
            1: 'success',
            0: 'warning',
            -1: 'error',
        }.get(obj.status)
        if css_class:
            return {'class': css_class, 'data': obj.name}
```

Note: Twitter bootstrap already provides handy CSS classes for table row styling: `error`, `warning`, `info` and `success`

Preview:

| <input type="checkbox"/> | Name | Countries |
|--------------------------|---------------|-----------|
| <input type="checkbox"/> | North America | 36 |
| <input type="checkbox"/> | Antarctica | 3 |
| <input type="checkbox"/> | Australia | 25 |
| <input type="checkbox"/> | South America | 14 |
| <input type="checkbox"/> | Europe | 49 |
| <input type="checkbox"/> | Africa | 57 |
| <input type="checkbox"/> | Asia | 53 |

List cell attributes

To add html attributes like `class` or `data` to list cells, you must define `suit_cell_attributes` callable (function). Callable receives object instance and column name as arguments and must return `dict` with attributes for current cell.

Example:

```
from django.contrib.admin import ModelAdmin
```

```
class CountryAdmin(ModelAdmin):
    ...

    def suit_cell_attributes(self, obj, column):
        if column == 'countries':
            return {'class': 'text-center'}
        elif column == 'name' and obj.status == -1:
            return {'class': 'text-error'}
```

Note: Twitter bootstrap already provides handy CSS classes for table cell alignment: `text-left`, `text-center`, `text-right`

Wysiwyg editors

How to use wysiwyg editors in Django Suit.

WYSIWYG editors

If you use third party wysiwyg editor app, you should follow its manual.

If you would like to use save horizontal space, you should use *full-width fieldset class* with your wysiwyg editor.

We tested many existing wysiwyg apps, but many of them were missing tests, had implementation or other flaws, so we decided to create our own. We are maintaining two simple wysiwyg apps for Imperavi Redactor and CKEditor.

Right now they doesn't support any uploads or anything with Django urls related, however it is planned to improve and update these packages in future.

These packages are independent and Django Suit isn't requirement.

Imperavi Redactor

- Package on github: [django-suit-redactor](#)
- Editor homepage: <http://imperavi.com/redactor/>
- License: Creative Commons Attribution-**NonCommercial** 3.0 license. For commercial use please buy license [here](#).
- Demo: <http://djangosuit.com/admin/examples/wysiwygeditor/add/>

Install:

1. `pip install django-suit-redactor`
2. Add `suit_redactor` to `INSTALLED_APPS`
3. Editor options for `editor_options` parameter can be found in [Redactor Docs](#)

Usage:

```
from django.forms import ModelForm
from django.contrib.admin import ModelAdmin
from suit_redactor.widgets import RedactorWidget
```

```

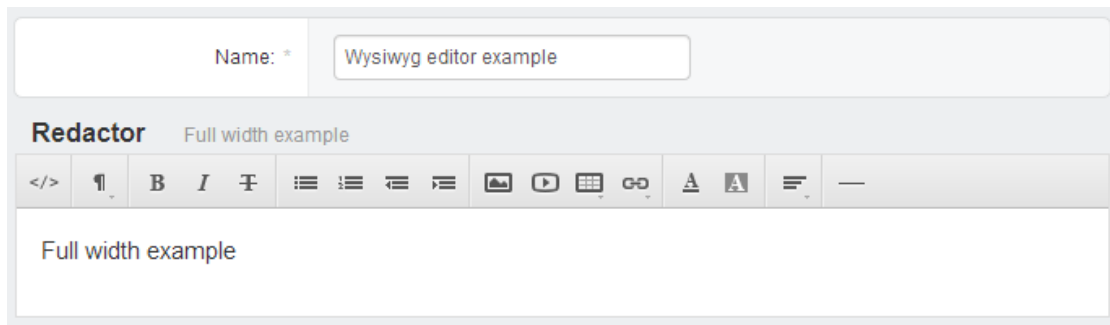
class PageForm(ModelForm):
    class Meta:
        widgets = {
            'name': RedactorWidget(editor_options={'lang': 'en'})
        }

class PageAdmin(ModelAdmin):
    form = PageForm
    fieldsets = [
        ('Body', {'classes': ('full-width',), 'fields': ('body',)})
    ]
    ...

admin.site.register(Page, PageAdmin)

```

Preview:



CKEditor 4.x

- Package on github: [django-suit-ckeditor](#)
- Editor homepage: <http://ckeditor.com>
- License: GPL, LGPL and MPL Open Source licenses
- Demo: <http://djangosuit.com/admin/examples/wysiwygeditor/add/>

Install:

1. `pip install django-suit-ckeditor`
2. Add `suit_ckeditor` to `INSTALLED_APPS`
3. Editor options for `editor_options` parameter can be found in [CKEditor Docs](#)

Usage for CKEditor is the same as for Imperavi Redactor, just change `RedactorWidget` to `CKEditorWidget`:

```

from django.forms import ModelForm
from django.contrib.admin import ModelAdmin
from suit_ckeditor.widgets import CKEditorWidget

class PageForm(ModelForm):
    class Meta:
        widgets = {
            'name': CKEditorWidget(editor_options={'startupFocus': True})
        }

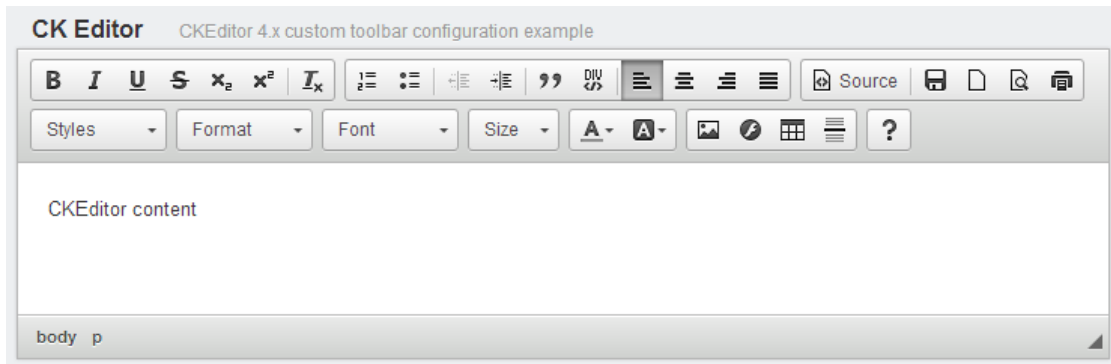
class PageAdmin(ModelAdmin):

```

```
form = PageForm
fieldsets = [
    ('Body', {'classes': ('full-width',), 'fields': ('body',)})
]
...

admin.site.register(Page, PageAdmin)
```

Preview:



Other wysiwyg apps

Also you can integrate WYSIWYG editor using any other third party apps.

Few third party apps we tested for Django Suit:

- [django-tinymce](#) - TinyMCE editor. Tested and works great
- [django-ckeditor](#) - CK Editor. Works great, but no support for CK Editor 4.x yet
- [django-cked](#) - CK Editor. Quite new and unfortunately not so stable. Supports CK Editor 4.x
- [django-redactorjs](#) - Imperavi Redactor.
- [django-redactor](#) - Imperavi Redactor. No Pypi package yet
- See also [WYSIWYG Editors](#) grid on DjangoPackages.com

JavaScript and CSS

JavaScript & CSS

JavaScript goodies

Inlines hook

When working with Django inlines and JavaScript, very often we want to hook/attach to event - when new inline row is added. Django Suit gives us such chance.

Use `JavaScriptSuit.after_inline.register` to register/attach your function to new inline addition event.

```
$(function () {
    Suit.after_inline.register('my_unique_func_name', function(inline_prefix, row){
        // Your code here
        console.info(inline_prefix)
        console.info(row)
    });
});
```

jQuery

Django Suit provides jQuery (currently v1.8.3) and it is using custom namespace to avoid collisions between different apps which also provide jQuery.

To use jQuery from Django Suit package:

```
// Use Suit.$ instead of $
Suit.$('.my-selector').addClass('my-class');

// Or for larger code you can wrap it in following way:
(function ($) {
    // Here you can use regular $ sign
    $('.my-selector').addClass('my-class');
})(Suit.$);

// On document ready example:
(function ($) {
    $(function () {
        $('.my-selector').addClass('my-class');
    });
})(Suit.$);
```

CSS goodies

Original collapse and wide fieldset classes are also supported by Django Suit. Usage:

```
from django.contrib.admin import ModelAdmin

class CountryAdmin(admin.ModelAdmin):
    ...
    fieldsets = [
        (None, {'fields': ['name', 'description']}),

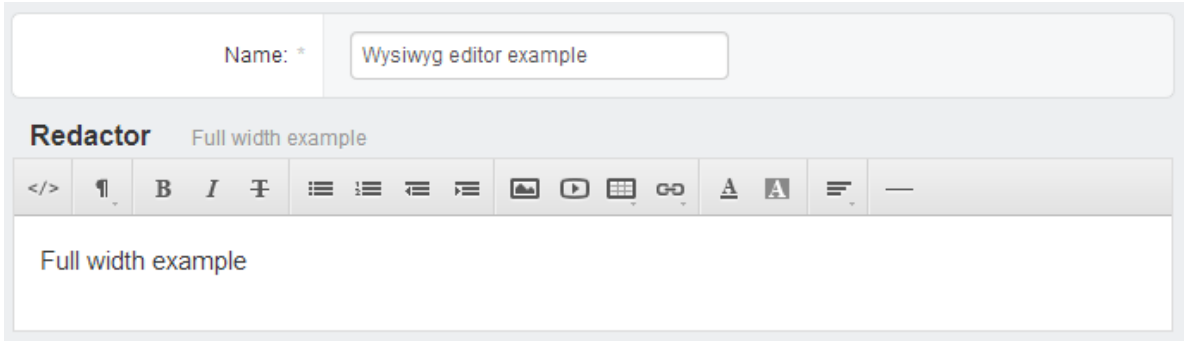
        ('Advanced settings', {
            'classes': ('collapse',), # Specify fieldset classes here
            'fields': ['hidden_checkbox', 'hidden_choice']}),
    ]
```

- collapse CSS class makes fieldset collapsable:

Advanced settings ([Show](#))

- wide CSS class makes fieldset labels wider

- `full-width` CSS class hides field label and makes field controls in full width (useful for wysiwyg editors). Because label will be hidden, this class is intended to use one field per fieldset and fieldset title will be used as field title.



- [Github](#): Use `django-suit` github issues, if you have any problems using Django Suit.

Examples

Besides documentation examples, Django Suit [demo application](#) source code is also available on separate github repository: [django-suit-examples](#). If you see anything in demo application, you can always go to this repository and see implementation and code in one of `models.py` or `admin.py` files

Supported apps

Besides Django admin, Django Suit supports following third-party apps:

- `django-cms` (v2.3.5 - v2.4.3) - [Example Read notes](#)
- `django-filer` (since v0.9.4) - [Example](#)
- `django-mptt` - [Example](#)
- `django-reversion` - [Example](#)
- `django-import-export` - [Example](#)
- Suggest other popular apps you would like to be supported [here](#)

Important: If you are using third-party apps with special admin support (like `django-cms`) you also need to add `'suit'` before `'cms'` in the list of `INSTALLED_APPS` in your `settings.py` file.

Contributing

To contribute to Django Suit:

```
# Clone forked Django Suit repo
git clone git@github.com:YOUR_USERNAME/django-suit.git suit

# Install Django Suit from local fork in editable mode
pip install -e suit
```

If you wish to participate in development discussions, join our IRC channel #django-suit on `irc.freenode.net`.

Tests

When contributing don't forget to test your code by running:

```
./manage.py test suit
```

CSS/LESS

Contributing on specifically UI/CSS features/fixes have more requirements:

- `lessc` compiler - <http://lesscss.org/>
- `watchdog` package - `pip install watchdog`
- `django-suit-examples` - it may be a good idea to add `examples` app to your project

While editing `.less` files, run following script, which automatically watches `.less` files for changes and compiles them to `suit.css`:

```
python suit/watch_less.py suit/static/suit/less/suit.less
```

Related packages

Related packages you can contribute to:

- [django-suit-redactor](#) - Imperavi Redactor (WYSIWYG editor) integration app
- [django-suit-ckeditor](#) - CKEditor (WYSIWYG editor) integration app
- [django-suit-rq](#) - Django-RQ (Queuing library) integration app
- [django-suit-examples](#) - demo app