django-tinymce Documentation

Release 2.3.0

Joost Cassee, Aljosa Mohorovic

March 20, 2023

Contents

django-tinymce is a Django application that contains a widget to render a form field as a TinyMCE editor.

Features:

- Use as a form widget or with a view.
- Enhanced support for content languages.
- Integration with the TinyMCE spellchecker.
- Enables predefined link and image lists for dialogs.
- Support for django-staticfiles
- Can compress the TinyMCE Javascript code.
- Integration with django-filebrowser.

The django-tinymce code is licensed under the MIT License. See the `LICENSE.txt` file in the distribution. Note that the TinyMCE editor is distributed under the LGPL v2.1 license.

Starting with django-tinymce v1.5.1 TinyMCE editor is bundled with django-tinymce to enable easy installation and usage. Note that django-tinymce and TinyMCE licenses are compatible (although different) and we have permission to bundle TinyMCE with django-tinymce.

CHAPTER 1

Documentation

1.1 Installation

This section describes how to install the django-tinymce application in your Django project.

1.1.1 Prerequisites

The django-tinymce application requires Django version 1.0 or higher. You will also need TinyMCE version 3.0.1 or higher and a language pack for *every language* you enabled in `settings.LANGUAGES`. If you use the django-filebrowser application in your project, the tinymce application can use it as a browser when including media.

If you want to use the spellchecker plugin using the supplied view (no PHP needed) you must install the PyEnchant package and dictionaries for your project languages. Note that the Enchant needs a dictionary that exactly matches your language codes. For example, a dictionary for code `'en-us'` will not automatically be used for `'en'`. You can check the availability of the Enchant dictionary for the `'en'` language code using the following Python code:

```
import enchant
enchant.dict_exists('en')
```

Note that the documentation will use 'TinyMCE' (capitalized) to refer the editor itself and 'django-tinymce' (lower case) to refer to the Django application.

1.1.2 Installation

1. Install django-tinymce using pip (or any other way to install python package) from PyPI. If you need to use a different way to install django-tinymce you can place the `tinymce` module on your Python path. You can put it into your Django project directory or run `python setup.py install` from a shell.

   ```
   pip install django-tinymce
   ```

2. Add `tinymce` to INSTALLED_APPS in `settings.py` for your project:

```
INSTALLED_APPS = (
    ...
    'tinymce',
    ...
)
```

3. Add `tinymce.urls` to `urls.py` for your project:

```
urlpatterns = patterns('',
    ...
    path('tinymce/', include('tinymce.urls')),
    ...
)
```

1.1.3 Testing

Verify that everything is installed and configured properly:

1. Setup an isolated environment with virtualenv and activate environment:

```
virtualenv --no-site-packages env
. env/bin/activate
```

2. Install required packages:

```
pip install Django django-tinymce
```

3. Setup environment variable `DJANGO_SETTINGS_MODULE`:

```
export DJANGO_SETTINGS_MODULE='tests.settings'
```

4. Create project and change into project directory:

```
django-admin startproject tinymce_test
cd tinymce_test
```

5. Setup test database (it will be created in current folder):

```
python manage.py migrate
```

6. Create superuser (follow the prompts):

```
python manage.py createsuperuser
```

7. Run Django runserver command to verify results:

```
python manage.py runserver
```

8. Open this address in a browser:

```
http://localhost:8000/admin/testapp/testpage/add/
```

If you see TinyMCE instead of standard textarea boxes everything is working fine, otherwise check installation steps.

1.1.4 Configuration

The application can be configured by editing the project's `settings.py` file.

TINYMCE_JS_URL (default: settings.STATIC_URL + 'tinymce/tinymce.min.js') The URL of the TinyMCE javascript file:

```
TINYMCE_JS_URL = os.path.join(STATIC_URL, "path/to/tiny_mce/tiny_mce.js")
```

TINYMCE_JS_ROOT (default: settings.STATIC_ROOT + 'tinymce') The filesystem location of the TinyMCE files. It is used by the compressor (see below):

```
TINYMCE_JS_ROOT = os.path.join(STATIC_ROOT, "path/to/tiny_mce")
```

TINYMCE_DEFAULT_CONFIG The default TinyMCE configuration to use. See the TinyMCE manual for all options. To set the configuration for a specific TinyMCE editor, see the `mce_attrs` parameter for the *widget*. !Important: The `language` attribute should only be set to force TinyMCE to have a different language than Django's current active language.

If not set, the default value of this setting is:

```
{
    "theme": "silver",
    "height": 500,
    "menubar": False,
    "plugins": "advlist,autolink,lists,link,image,charmap,print,preview,anchor,"
    "searchreplace,visualblocks,code,fullscreen,insertdatetime,media,table,paste,"
    "code,help,wordcount",
    "toolbar": "undo redo | formatselect | "
    "bold italic backcolor | alignleft aligncenter "
    "alignright alignjustify | bullist numlist outdent indent | "
    "removeformat | help",
}
```

TINYMCE_SPELLCHECKER (default: False) Whether to use the spell checker through the supplied view. You must add `spellchecker` to the TinyMCE plugin list yourself, it is not added automatically.

TINYMCE_COMPRESSOR (default: False) Whether to use the TinyMCE compressor, which gzips all Javascript files into a single stream. This makes the overall download size 75% smaller and also reduces the number of requests. The overall initialization time for TinyMCE will be reduced dramatically if you use this option.

TINYMCE_EXTRA_MEDIA (default: None) Extra media to include on the page with the *widget*.

TINYMCE_FILEBROWSER (default: True if 'filebrowser' is in INSTALLED_APPS, else False) Whether to use the django-filebrowser as a custom filebrowser for media inclusion. See the official TinyMCE documentation on custom filebrowsers.

Example:

```
TINYMCE_JS_URL = 'http://debug.example.org/tiny_mce/tiny_mce_src.js'
TINYMCE_DEFAULT_CONFIG = {
    "height": "320px",
    "width": "960px",
    "menubar": "file edit view insert format tools table help",
    "plugins": "advlist autolink lists link image charmap print preview anchor␣
↪searchreplace visualblocks code "
    "fullscreen insertdatetime media table paste code help wordcount spellchecker",
    "toolbar": "undo redo | bold italic underline strikethrough | fontselect␣
↪fontsizeselect formatselect | alignleft "
```

(continues on next page)

(continued from previous page)

```
    "aligncenter alignright alignjustify | outdent indent |  numlist bullist␣
↪checklist | forecolor "
    "backcolor casechange permanentpen formatpainter removeformat | pagebreak |␣
↪charmap emoticons | "
    "fullscreen  preview save print | insertfile image media pageembed template link␣
↪anchor codesample | "
    "a11ycheck ltr rtl | showcomments addcomment code",
    "custom_undo_redo_levels": 10,
    "language": "es_ES",  # To force a specific language instead of the Django␣
↪current language.
}
TINYMCE_SPELLCHECKER = True
TINYMCE_COMPRESSOR = True
TINYMCE_EXTRA_MEDIA = {
    'css': {
        'all': [
            ...
        ],
    },
    'js': [
        ...
    ],
}
```

1.2 Usage

The application can enable TinyMCE for one form field using the `widget` keyword argument of `Field` constructors or for all textareas on a page using a view.

1.2.1 Using the widget

If you use the widget (recommended) you need to add some python code and possibly modify your template.

Python code

The TinyMCE widget can be enabled by setting it as the widget for a formfield. For example, to use a nice big TinyMCE widget for the content field of a flatpage form you could use the following code:

```
from django import forms
from django.contrib.flatpages.models import FlatPage
from tinymce.widgets import TinyMCE

class FlatPageForm(forms.ModelForm):
    ...
    content = forms.CharField(widget=TinyMCE(attrs={'cols': 80, 'rows': 30}))
    ...

    class Meta:
        model = FlatPage
```

The widget accepts the following extra keyword argument:

mce_attrs (default: {}) Extra TinyMCE configuration options. Options from `settings.TINYMCE_DEFAULT_CONFIG` (see *Configuration*) are applied first and can be overridden. Python types are automatically converted to Javascript types, using standard JSON encoding. For example, to disable word wrapping you would include `'nowrap': True`.

The tinymce application adds one TinyMCE configuration option that can be set using `mce_attrs` (it is not useful as a default configuration):

content_language (default: django.utils.translation.get_language_code()) The language of the widget content. Will be used to set the `language`, `directionality` and `spellchecker_languages` configuration options of the TinyMCE editor. It may be different from the interface language, which defaults to the current Django language and can be changed using the `language` configuration option in `mce_attrs`)

Templates

The widget requires a link to the TinyMCE javascript code. The `django.contrib.admin` templates do this for you automatically, so if you are just using tinymce in admin forms then you are done. In your own templates containing a TinyMCE widget you must add the following to the HTML `HEAD` section (assuming you named your form 'form'):

```
<head>
    ...
    {{ form.media }}
</head>
```

See also the section of form media in the Django documentation.

The `HTMLField` model field type

For lazy developers the tinymce application also contains a model field type for storing HTML. It uses the TinyMCE widget to render its form field. In this example, the admin will render the `my_field` field using the TinyMCE widget:

```
from django.db import models
from tinymce import models as tinymce_models

class MyModel(models.Model):
    my_field = tinymce_models.HTMLField()
```

In all other regards, `HTMLField` behaves just like the standard Django `TextField` field type.

1.2.2 Using the view

If you cannot or will not change the widget on a form you can also use the `tinymce-js` named view to convert some or all textfields on a page to TinyMCE editors. On the template of the page, add the following lines to the `HEAD` element:

```
<script src="{{ STATIC_URL }}js/tiny_mce/tiny_mce.js"></script>
<script src="{% url "tinymce-js" "NAME" %}"></script>
```

The use of `STATIC_URL` needs the `django.core.context_processors.static` context processors.

You may want to use``{% static %}`` instead like:

```
<script src="{% static "js/tiny_mce/tiny_mce.js" %}"></script>
<script src="{% url "tinymce-js" "NAME" %}"></script>
```

Be careful that some STATICFILES_STORAGE will modify your tiny_mce.js file name and your file will fail to load.

The NAME argument allows you to create multiple TinyMCE configurations. Now create a template containing the Javascript initialization code. It should be placed in the template path as NAME/tinymce_textareas.js or tinymce/NAME_textareas.js.

Example:

```
tinyMCE.init({
    mode: "textareas",
    theme: "silver",
    plugins: "spellchecker,directionality,paste,searchreplace",
    language: "{{ language }}",
    directionality: "{{ directionality }}",
    spellchecker_languages : "{{ spellchecker_languages }}",
    spellchecker_rpc_url : "{{ spellchecker_rpc_url }}"
});
```

This example also shows the variables you can use in the template. The language variables are based on the current Django language. If the content language is different from the interface language use the tinymce-js-lang view which takes a language (LANG_CODE) argument:

```
<script src="{% url "tinymce-js-lang" "NAME","LANG_CODE" %}"></script>
```

1.2.3 External link and image lists

The TinyMCE link and image dialogs can be enhanced with a predefined list of links and images. These entries are filled using a variable loaded from an external Javascript location. The tinymce application can serve these lists for you.

Creating external link and image views

To use a predefined link list, add the external_link_list_url option to the mce_attrs keyword argument to the widget (or the template if you use the view). The value is a URL that points to a view that fills a list of 2-tuples (*name*, *URL*) and calls tinymce.views.render_to_link_list. For example:

Create the widget:

```
from django import forms
from django.urls import reverse
from tinymce.widgets import TinyMCE

class SomeForm(forms.Form):
    somefield = forms.CharField(widget=TinyMCE(mce_attrs={'external_link_list_url':␣
↪reverse('someviewname')}))
```

Create the view:

```
from tinymce.views import render_to_link_list

def someview(request):
    objects = ...
    link_list = [(unicode(obj), obj.get_absolute_url()) for obj in objects]
    return render_to_link_list(link_list)
```

Finally, include the view in your URLconf.

Image lists work exactly the same way, just use the TinyMCE `external_image_list_url` configuration option and call `tinymce.views.render_to_image_list` from your view.

The `flatpages_link_list` view

As an example, the tinymce application contains a predefined view that lists all `django.contrib.flatpages` objects: `tinymce.views.flatpages_link_list`. If you want to use a TinyMCE widget for the flatpages `content` field with a predefined list of other flatpages in the link dialog you could use something like this:

```
from django import forms
from django.contrib.flatpages.admin import FlatPageAdmin
from django.contrib.flatpages.models import FlatPage
from django.urls import reverse
from tinymce.widgets import TinyMCE

class TinyMCEFlatPageAdmin(FlatPageAdmin):
    def formfield_for_dbfield(self, db_field, **kwargs):
        if db_field.name == 'content':
            return db_field.formfield(widget=TinyMCE(
                attrs={'cols': 80, 'rows': 30},
                mce_attrs={'external_link_list_url': reverse('tinymce-linklist')},
            ))
        return super().formfield_for_dbfield(db_field, **kwargs)

somesite.register(FlatPage, TinyMCEFlatPageAdmin)
```

If you want to enable this for the default admin site (`django.contrib.admin.site`) you will need to unregister the original ModelAdmin class for flatpages first:

```
from django.contrib import admin

admin.site.unregister(FlatPage)
admin.site.register(FlatPage, TinyMCEFlatPageAdmin)
```

The source contains a test project that includes this flatpages model admin.

1. Checkout django-tinymce: `git clone https://github.com/jazzband/django-tinymce.git`
2. Go to the test project: `cd django-tinymce/tests`
3. Copy the `tiny_mce` directory from the TinyMCE distribution into `media/js`
4. Run `python manage.py migrate`
5. Run `python manage.py createsuperuser`
6. Run `python manage.py runserver`
7. Connect to http://localhost:8000/admin/ and login with the above-created user.

1.3 Changelog

This document describes changes between each past release.

1.3.1 3.6.1 (2023-03-20)

- Fixed a regression by reverting usage of staticfiles to find tinymce location (#420, #430).

1.3.2 3.6.0 (2023-03-18)

- Upgrade embedded tinyMCE from 5.10.1 to 5.10.7
- Replace obsolete mode and elements by selector and target (#417)
- Detect non-installed tinyMCE soon in init_tinymce.js
- Stop installing the tests directory (#355)
- Add support for translatable strings in tinyMCE config
- Use staticfiles storage API to find tinymce location (#420)

1.3.3 3.5.0 (2022-08-27)

- Support new non-jQuery formset:added event triggered on Django 4.1
- Replace an obsolete call to tinyMCE.editors (#391)
- Confirm support for Django 4.0 and 4.1
- Drop support for Django 3.0, 3.1 and Python 3.6
- Add Python 3.10 support

1.3.4 3.4.0 (2021-11-25)

- Upgrade to TinyMCE 5.10.1
- Confirmed support for Django 3.2
- Repair the spellchecker plugin functionality.

1.3.5 3.3.0 (2021-03-24)

- Add support for Django 3.1
- Improve detection of dynamically added formsets
- Update configuration documentation

1.3.6 3.2.0 (2020-12-10)

- Remove support for universal builds
- Add compatibility of django-filebrowser with tinymce 5
- Load the CHANGELOG in the documentation front page
- Fix en_US language loading
- Speed up tests by removing the loading of a database
- Add verbosity option to tests

- Assume TinyMCE files are utf-8 encoded

1.3.7 3.1.0 (2020-09-29)

- Add support for language configuration
- Upgrade to TinyMCE 5.5.0
- Remove the jQuery dependency and fix multiples errors around that

  Note: As a consequence, `TINYMCE_INCLUDE_JQUERY` setting has been removed.

- Move to the Jazzband organization

1.3.8 3.0.2 (2020-04-22)

- Update the default config.

1.3.9 3.0.0 (2020-04-10)

- Upgrade to TinyMCE 5
- Fix compressor

1.3.10 2.9.0 (2020-04-10)

- Upgrade test matrix to Python 3.7 and Django 2.1, 2.2
- Add support for TinyMCE FileBrowser 4.0
- Remove support for South

1.3.11 2.8.0 (2019-01-15)

- Use the attrs set on instantiation as well as the attrs passed to render (#237)

1.3.12 2.7.0 (2017-12-19)

- Drop support for Django 1.7, 1.8, 1.9 and 1.10.
- Django 1.11 is still supported but issues a deprecation warning.
- Add support for Django 2.0
- Added INCLUDE_JQUERY setting to decide whether TinyMCE.media should include a jQuery release (#190).

1.3.13 2.6.0 (2017-01-23)

- Avoid deprecation warning with django.core.urlresolvers (#188)
- Fixed a client-side validation issue when the TinyMCE widget has the HTML required attribute set (#187).
- Fixed a crash when no languages are activated (#175).

1.3.14 2.5.0 (2017-01-23)

- Added compatibility for Django 1.11.
- Dropped support for Django 1.6.

1.3.15 2.4.0 (2016-08-31)

- Added compatibility for Django 1.10.
- Fix JQuery Problem with grappelli
- Fix Python 3 compatibility (#170)
- Improve documentation (#163, #171)
- Cleaned Imports (#182)
- Fix TinyMCE Widget for ModelTranslation tabs (#174)
- Fix JSON mimetype (#186)

1.3.16 2.3.0 (2016-03-10)

- Added tests (#149)
- Improved Python3 support

1.3.17 2.2.0 (2015-12-23)

- TinyMCE compressor now use staticfiles to get the file content (and to find files that are in multiple static directory.) (#142)

1.3.18 2.1.0 (2015-12-23)

- Rewrite URL files to let it works with Django 1.9 (#147, #148)
- Add a CONTRIBUTORS file.

1.3.19 2.0.6 (2015-11-12)

- Make sure jQuery is loaded both in the admin and for non-admin forms. (#141)

1.3.20 2.0.5 (2015-09-09)

- Use static finders for development mode. (#131)

1.3.21 2.0.4 (2015-08-07)

- Fix non-admin jQuery.

1.3.22 2.0.3 (2015-08-06)

- Handle non-admin jQuery. (#108)

1.3.23 2.0.2 (2015-07-26)

- Add Python3 support.

1.3.24 2.0.1 (2015-07-24)

- Fix missing CHANGELOG.

1.3.25 2.0.0 (2015-07-23)

- Starts supporting Django 1.8

Older Changelog entries can be found on https://github.com/jazzband/django-tinymce/blob/3.1.0/docs/history.rst