django-statici18n

Release 2.1.0

May 17, 2021

Contents

A Django app compiling i18n JavaScript catalogs to static files.

CHAPTER 1

Overview

When dealing with internationalization in JavaScript code, Django provides the JSONCatalog view which sends out a JavaScript code library with functions that mimic the gettext interface, plus an array of translation strings.

At first glance, it works well and everything is fine. But, because JSONCatalog view is generating JavaScript catalog dynamically on each and every request, it's adding an overhead that can be an issue with site growth.

That's what `django-statici18n` is for:

> Collecting JavaScript catalogs from each of your Django apps (and any other place you specify) into a single location that can easily be served in production.

The main website for `django-statici18n` is github.com/zyegfryed/django-statici18n where you can also file tickets.

CHAPTER 2

Supported Django Versions

`django-statici18n` works with all the Django versions officially supported by the Django project. At this time of writing, these are the 2.2 (LTS), 3.1 and 3.2 series.

CHAPTER 3

Installation

1. Use your favorite Python packaging tool to install `django-statici18n` from PyPI, e.g.:

```
pip install django-statici18n
```

2. Add `'statici18n'` to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = [
    # ...
    'statici18n',
]
```

3. Once you have translated and compiled your messages, use the `compilejsi18n` management command:

```
python manage.py compilejsi18n
```

4. Add the django.core.context_processors.i18n context processor to the `context_processors` section for your backend in the `TEMPLATES` setting - it should have already been set by Django:

```
TEMPLATES = [
  {
    # ...
    'OPTIONS': {
      'context_processors': {
        # ...
        'django.template.context_processors.i18n',
      },
    },
  },
]
```

5. Edit your template(s) and replace the dynamically generated script by the statically generated one:

```
<script src="{{ STATIC_URL }}jsi18n/{{ LANGUAGE_CODE }}/djangojs.js"></
↪script>
```

Note: By default, the generated catalogs are stored to `STATIC_ROOT/jsi18n`. You can modify the output path and more options by tweaking `django-statici18n` settings.

(Optional)

The following step assumes you're using django.contrib.staticfiles.

5. Edit your template(s) and use the provided template tag:

```
{% load statici18n %}
<script src="{% statici18n LANGUAGE_CODE %}"></script>
```

6. Or inline the JavaScript directly in your template:

```
{% load statici18n %}
<script>{% inlinei18n LANGUAGE_CODE %}</script>
```

CHAPTER 4

Documentation

4.1 Management commands

4.1.1 compilejsi18n

Collect JavaScript catalog files in a single location.

Some commonly used options are:

-l LOCALE or --locale=LOCALE The locale to process. Default is to process all but if for some reason I18N features are disabled, only *settings.LANGUAGE_CODE* will be processed.

-d DOMAIN or --domain=DOMAIN Override the gettext domain. By default, the command uses the `djangojs` gettext domain.

-p PACKAGES or -packages=PACKAGES A list of packages to check for translations. Default is `'django.conf'`. Use multiple times to add more.

-o OUPUT_DIR or --output=OUTPUT_DIR Output directory to store generated catalogs. Defaults to the joining path of *STATICI18N_ROOT* and *STATICI18N_OUTPUT_DIR*.

-f OUTPUT_FORMAT or --format=OUTPUT_FORMAT

Format of the output catalog. Options are:

- `js`,
- `json`.

Defaults to `js`.

-n NAMESPACE or --namespace=NAMESPACE The final gettext will be put with window.SpecialBlock.gettext rather than the window.gettext. This is useful for pluggable modules which need Javascript i18n.

Defaults to `None`.

For a full list of options, refer to the `compilejsi18n` management command help by running:

```
$ python manage.py compilejsi18n --help
```

Note: Missing directories will be created on-the-fly by the command when invoked.

4.2 Template tags

4.2.1 statici18n

templatetags.statici18n.**statici18n**(*locale*)

Builds the full JavaScript catalog URL for the given locale by joining the *STATICI18N_OUTPUT_DIR* and *STATICI18N_FILENAME_FUNCTION* settings:

```
{% load statici18n %}
<script src="{% statici18n LANGUAGE_CODE %}"></script>
```

This is especially useful when using a non-local storage backend to deploy files to a CDN or when using CachedStaticFilesStorage storage to serve files.

Note: Behind the scenes, it's a thin wrapper around the static template tag. Therefore, ensure that django.contrib.staticfiles is configured before proceeding. See *How to configure static files with django-statici18n?* for more information.

4.3 Settings

django.conf.settings.**STATICI18N_DOMAIN**

Default 'djangojs'

The gettext domain to use when generating static files.

Can be overrided with the -d/--domain option of compilejsi18n command.

Usually you don't want to do that, as JavaScript messages go to the djangojs domain. But this might be needed if you deliver your JavaScript source from Django templates.

django.conf.settings.**STATICI18N_PACKAGES**

Default ('django.conf')

A list of packages to check for translations.

Can be overrided with the -p/--package option of *compilejsi18n* command.

Each string in packages should be in Python dotted-package syntax (the same format as the strings in INSTALLED_APPS) and should refer to a package that contains a locale directory. If you specify multiple packages, all those catalogs are merged into one catalog. This is useful if you have JavaScript that uses strings from different applications.

django.conf.settings.**STATICI18N_ROOT**

Default STATIC_ROOT

Controls the file path that catalog files will be written into.

django.conf.settings.**STATICI18N_OUTPUT_DIR**

Default `'jsi18n'`

Controls the directory inside *STATICI18N_ROOT* that generated files will be written into.

django.conf.settings.**STATICI18N_FILENAME_FUNCTION**

Default `'statici18n.utils.default_filename'`

The dotted path to the function that creates the filename.

The function receives two parameters:

- `locale`: a string representation of the locale currently processed
- `domain`: a string representation of the gettext domain used to check for translations

By default, the function returns the path `'<locale>/<domain>.js'`.

The final filename is resulted by joining *STATICI18N_ROOT*, *STATICI18N_OUTPUT_DIR* and *STATICI18N_FILENAME_FUNCTION*.

For example, with default settings in place and `STATIC_ROOT = 'static'`, the JavaScript catalog generated for the `en_GB` locale is: `'static/jsi18n/en_GB/djangojs.js'`.

Use the legacy function `statici18n.utils.legacy_filename` to generate a filename with the language code derived from the `django.utils.translation.trans_real import to_language`.

django.conf.settings.**STATICI18N_NAMESPACE**

Default `None`

Javascript identifier to use as namespace. This is useful when we want to have separate translations for the global and the namespaced contexts. The final gettext will be put under *window.<namespace>.gettext* rather than the *window.gettext*. Useful for pluggable modules that need JS i18n.

4.4 Troubleshooting

4.4.1 Files are not served during development

By default `django-statici18n` doesn't rely on `django.contrib.staticfiles`, so you have to serve the generated catalogs files with the Django dev server. For example:

```
# urls.py
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = patterns('',
    # ... the rest of your URLconf goes here ...
) + static(settings.STATIC_URL, document_root=settings.STATICI18N_ROOT)
```

However, when using the *statici18n* template tag you should first integrate `django-static18n` with `django.contrib.staticfiles`. See *How to configure static files with django-statici18n?* for more information.

Note: Even if the setup looks a bit more tedious at first sight, using the *statici18n* template tag is the recommended way and it will make your life easier in the long run.

4.4.2 Catalog is empty

`django-statici18n` requires that the locale paths are available in the settings. So just add `LOCALE_PATHS=('/path/to/your/locale/directory',)` to the settings file.

For more information on how Django discovers translations, refer to the official documentation.

4.5 FAQ

4.5.1 How to configure static files with `django-statici18n`?

Due to the modularity of `django.contrib.staticfiles` it's easy to use the storage facility provided by tweaking some settings.

There's two solution leveraging the `STATICFILES_FINDERS` setting:

- using a dedicated application, or,
- using a dedicated directory to hold the catalog files.

In the next sections, we'll detail with examples how to use both solutions. Choose the one that best fits your needs and/or taste.

See static files management for more information.

Once setup is in place, run the `compilejsi18n` command to compile/update the Javascript catalog files followed by the `collectstatic` command to generate the static files:

```
# compile/update Javascript catalog files...
$ python manage.py compilejsi18n

# then, collect static files.
$ python manage.py collectstatic
```

Using a placeholder app

You need to have the `AppDirectoriesFinder` finder enabled (the default).

Create a minimal app with a `static` subdirectory. For example, let's create an app named **ìi18n** to hold the generated catalogs:

```
cd /path/to/your/django/project
mkdir -p i18n/static
touch i18n/__init__.py i18n/models.py
```

Your project layout should then looks like the following:

```
example_project
|-- app
|   |-- __init__.py
|   |-- admin.py
|   |-- locale
|   |-- models.py
|   |-- static
|   |-- templates
|   |-- tests.py
|   `-- views.py
|-- i18n                    <-- Your dedicated app
|   |-- __init__.py
|   |-- models.py           <-- A placeholder file to enable app loading
|   `-- static              <-- The output directory of catalog files
|       `-- jsi18n
|-- manage.py
|-- project
|   |-- __init__.py
|   |-- locale
|   |-- settings.py
|   |-- templates
|   `-- urls.py
`-- public
    `-- static              <-- The output directory of collected
        `-- jsi18n              static files for deployment
```

Then update your settings accordingly. Following the previous example:

```
# project/settings.py

# ... the rest of your settings here ...

INSTALLED_APPS = (
    'django.contrib.staticfiles',
    # ...
    'statici18n',
    'i18n',
)

STATIC_ROOT = os.path.join(BASE_DIR, "public", "static")
STATICI18N_ROOT = os.path.join(BASE_DIR, "i18n", "static")
```

Using a placeholder directory

This approach extends the `STATICFILES_DIRS` setting. You need to have the `FileSystemFinder` finder enabled (the default).

Following is an example project layout:

```
example_project
|-- app
|   |-- __init__.py
|   |-- admin.py
|   |-- locale
|   |-- models.py
|   |-- tests.py
```

(continues on next page)

(continued from previous page)

```
|   `-- views.py
|-- manage.py
|-- project
|   |-- __init__.py
|   |-- locale
|   |-- settings.py
|   |-- static              <-- Directory holding catalog files
|   |   `-- jsi18n
|   |-- templates
|   `-- urls.py
`-- public
    `-- static              <-- The output directory of collected
                                static files for deployment
```

Then update your settings accordingly. Following the previous example:

```
# project/settings.py

# ... the rest of your settings here ...

INSTALLED_APPS = (
    'django.contrib.staticfiles',
    # ...
    'statici18n',
)

STATIC_ROOT = os.path.join(BASE_DIR, "public", "static")
STATICI18N_ROOT = os.path.join(BASE_DIR, "project", "static")
STATICFILES_DIRS += (STATICI18N_ROOT,)
```

4.5.2 Can I use the generated catalog with RequireJS?

Yes. You just need some boilerplate configuration to export the object reference, like the following:

```
# settings.py
STATICI18N_ROOT = os.path.join(BASE_DIR, "project", "static")
STATICFILES_DIRS += (STATICI18N_ROOT,)

# app.js
require.config({
        baseUrl: "static/js",
        paths: {
                "jsi18n": "../jsi18n/{{ LANGUAGE_CODE }}/djangojs",
        },
        shim: {
                "jsi18n":
                {
                        exports: 'django'
                },
        }
})

// Usage
require(["jquery", "jsi18n"], function($, jsi18n) {
    console.log(jsi18n.gettext('Internationalization is fun !'));
```

(continues on next page)

(continued from previous page)

```
        // > "L'internationalisation, c'est cool !"
})
```

4.6 Changelog

4.6.1 v2.1.0 (2021 May 17)

- [PR#50] Fix test under Django 3.2 (thanks @mbakke)
- Add Django 3.2 support
- Drop Django 3.0 support
- Add Python 3.9 support

4.6.2 v2.0.1 (2020 Oct 18)

- Switch to codecov as coverage service
- Remove no longer used compatibility code
- Simplify tooling, rely only on tox
- Remove six dependency
- Fix Django and django-appconf minimum version
- Fix Python versions suppport
- Add project URL

4.6.3 v2.0.0 (2020 Sep 18)

- [PR#49] Change *force_text* to *force_str* (thanks @bullfest)
- Add Django 3.1 support
- Drop Django 1.8 support
- Drop Django 1.9 support
- Drop Django 1.10 support
- Drop Django 1.11 support
- Drop Django 2.0 support
- Drop Django 2.1 support
- Fix linting issues
- Use Python 3.8 as default version
- Fix travis matrix definition
- Upgrade Sphinx configuration

4.6.4 v1.9.0 (2020 Jan 11)

- [PR#48] Update django-appconf, thanks @zetaab
- Add Django 2.2 support
- Add Django 3.0 support
- Drop Python 3.3 support (reached EOL)
- Drop Python 3.4 support (reached EOL)

4.6.5 v1.8.3 (2019 Mar 03)

- [PR#44] Fix python3 compatibility about inlinei18n (thanks @outloudvi)
- Add Django 2.1 support
- Fix warning for static templatetag in Django 2.1
- Fix deprecation warning on collections module
- Compress tox matrix definition
- Change travis to use tox under the hood

4.6.6 v1.8.2 (2018 Jun 29)

This is a maintenance release due to CI issues.

- Add Python 3.7 classifier
- Fix Travis matrix definition
- Remove Python 3.7 support for Travis

4.6.7 v1.8.1 (2018 Jun 29)

- Fix #42: regression issue with packages (thanks @classifaddict)
- Update to latest Python versions
- Add Python 3.7 support

4.6.8 v1.8.0 (2018 May 31)

- [PR#39] Serialize packages before giving it to Django (thanks @askoretskiy)
- [PR#41] Add the namespace parameter (thanks @afzaledx and @pomegranited)

4.6.9 v1.7.0 (2018 Feb 11)

- Fix documentation (thanks @philipbelesky and @pre101)

Warning: The following changes are backward-incompatible with the previous release.

- Use the plain locale for filename by default (thanks @genonfire) For legacy behavior, set `STATICI18N_FILENAME_FUNCTION` setting with `'statici18n.utils.legacy_filename'`.

4.6.10 v1.6.1 (2018 Jan 20)

- Use ASCII character instead of UTF-8 one to fix build with Python 3 (thanks @sunpoet)

4.6.11 v1.6.0 (2018 Jan 12)

- Add Django 2.0 support (thanks Martin Pauly)
- Drop Python 3.2 support
- Add Python 3.6 support

4.6.12 v1.5.0 (2017 Dec 08)

- Load statici18n conf via Django's *AppConfig* (thanks @julen)

4.6.13 v1.4.0 (2017 Jun 22)

- Add Django 1.11 support (thanks @bmedx)

4.6.14 v1.3.0 (2017 Jan 19)

- Pass language name instead of locale name on Django 1.10 (thanks @quantum5)
- Fix circle-ci builds

4.6.15 v1.2.1 (2016 Aug 20)

- Fix Django links to use version 1.10

4.6.16 v1.2.0 (2016 Aug 20)

- #17: Mark inlinei18n output as safe (thanks @quantum5)
- #23: Added support for JSON format in command-line based catalog generation (thanks @rarguelloF)
- #18: Added support for Django 1.9 and 1.10; this change also drops support for Django 1.4, 1.5, 1.6 and 1.7 as they are not officially supported by the DSF anymore (thanks @julen)
- #19: Added support for *USE_18N = False* (thanks @julen)

4.6.17 v1.1.5 (2015 Aug 7)

- New release due to missing changelog in previous one

4.6.18 v1.1.4 (2015 Aug 7)

- Fix #14: compilejsi18n command should skip system checks (thanks @bubenkoff and @xolox)
- Update python2.7 to 2.7.10

4.6.19 v1.1.3 (2015 Apr 19)

- Add django 1.8 support
- Fix deprecation warning from django.utils.importlib (thanks @ogai)

4.6.20 v1.1.2 (2015 Mar 18)

- Updated dependencies
- Added Python 3.2 and Django 1.7 test support
- Updated requirements to include the newest version of appconf and changed setup.py to reflect appconf requirement (thanks Nicholas Lockhart)

4.6.21 v1.1.1 (2014 Nov 17)

- Added empty catalog entry to troubleshooting section (thanks @eduardo-matos)

4.6.22 v1.1 (2014 Jan 12)

- Added i18ninline template tag (thanks @jezdez)
- Added RequireJS entry to the FAQ (thanks @Ewjoachim)

4.6.23 v1.0.1 (2013 Nov 20)

- Improved documentation clarity and cross-references
- Updated classifiers

4.6.24 v1.0.0 (2013 Nov 18)

- Added Django 1.6 support (thanks @ryanbutterfield)
- Improved documentation
- Added full test suite

Warning: The following changes are backward-incompatible with the previous release.

- Now use `STATIC_ROOT` as default value for `STATICI18N_ROOT`.

4.6.25 v0.4.5 (2013 Jun 13)

- Fixed ImportError exception.

4.6.26 v0.4.4 (2013 Jun 12)

- Fixed issue in filename function now using language code instead of locale name. Thanks Marc Kirkwood.
- Fixed Django documentation URLs to use 1.5 release.
- Improved the overall documentation.

4.6.27 v0.4.3 (2013 Jun 10)

- Updated documentation reference to Django 1.5.
- Fixed a typo in documentation.

4.6.28 v0.4.2 (2013 Feb 04)

- Fixing compiling the JS formats for non-default languages. Thanks @jezdez.

4.6.29 v0.4.1 (2012 Oct 17)

- Worked around an issue with unescaped string literals in Django JavaScript i18n code. Thanks @jezdez.

4.6.30 v0.4.0 (2012 Apr 04)

- Added statici18n template tag.

4.6.31 v0.3.1 (2012 Apr 03)

- Added license
- Fixed installation error due to missing manifests file.

4.6.32 v0.3.0 (2012 Apr 03)

- Added Sphinx documentation.
- Added many settings managed with django-appconf.

4.6.33 v0.2.0 (2012 Apr 02)

> **Warning:** The following changes are backward-incompatible with the previous release.

- Renamed `collecti18n` command to `compilejsi18n`.

- Now use current static directory instead of `STATIC_ROOT` for sane defaults.

4.6.34 v0.1.0 (2012 Apr 02)

- Initial commit.

Python Module Index

S

Index

S

T