django-require-i18n Documentation

Release 1.2.2

Collab

November 13, 2015

Contents

Django management command for extracting and compiling internationalization/localization string resources used in the Require.js i18n plugin.

CHAPTER 1

Installation

Use pip to download and install the package from PyPi:

```
pip install django-require-i18n
```

Or checkout the source code from Github:

```
git clone https://github.com/collab-project/django-require-i18n.git
```

CHAPTER 2

Usage

After installation a new Django management command is available called `compile_js`. Use the `--help` option to learn more:

```
./manage.py compile_js --help
```

CHAPTER 3

Tutorial

Read the Tutorial for more information on how to configure your Django and Require.js project.

3.1 Tutorial

This tutorial shows how to extract and update translation strings with django-require-i18n.

The goal is to add and update the translation for the Dutch language (`nl`).

3.1.1 Create bundle

Start by defining a require.js i18n bundle.

For example, in a Django project called `myproject` there is a require.js application called `site` with a directory structure similar to this:

```
+ myproject
  - manage.py
  + locale
  + myproject
  + static
    + js
      + site
        - main.js
        + views
        + nls
          - colors.js
          + root
            - colors.js
```

The Dutch language code `nl` in `site/nls/colors.js` is enabled:

```
define({
    "root": true,

    "nl": true
});
```

The root file `site/nls/root/colors.js` contains the key/value pairs to translate:

```
// Copyright (c) My Project

define(
{
    "redLabel": "Red",
    "greenLabel": "Green"
}
);
```

3.1.2 Configure Django application

Add the `require_i18n` and `tower` apps to the `INSTALLED_APPS` setting in the Django project settings file:

```
INSTALLED_APPS = [
  # ...
  'tower',
  'require_i18n'
]
```

Note: This example uses the default Django settings for static files and localization. In practice this means that:

- the `static` directory matches the Django STATIC_ROOT setting but it can also be included in the Django STATICFILES_DIRS list instead.
- the `locale` directory is included in the Django LOCALE_PATHS setting.

Configure the Tower application (refer to the documentation for details):

```
import os

# Tower root
ROOT = os.path.dirname(__file__)

TEXT_DOMAIN = 'messages'

# Jinja configuration for tower.
def JINJA_CONFIG():
    config = {'extensions': ['tower.template.i18n',
                             'jinja2.ext.with_',
                             'jinja2.ext.loopcontrols'],
              'finalize': lambda x: x if x is not None else ''}
    return config

# function that takes arbitrary set of args and combines them with ROOT to
# form a new path.
path = lambda *args: os.path.abspath(os.path.join(ROOT, *args))
```

Add the `DOMAIN_METHODS` setting so it matches the require.js application directory structure:

```
# dict of domain to file spec and extraction method tuples.
DOMAIN_METHODS = {
    'site': [
        ('static/js/site/nls/root/*.js', 'require_i18n.util.extract_tower_json'),
    ]
}
```

The keys in this dict refer to the domain name (`site`) and it's values are mappings between paths to the root translation files and the Python method that will be used to extract the translation strings

(require_i18n.util.extract_tower_json).

3.1.3 Customize settings

By default the license header in the translated catalog contains some dummy data and you probably want to change that to match your project. This can be done by adding the REQUIRE_I18N_HEADER setting:

```
REQUIRE_I18N_HEADER = """Copyright (C) 2015 Me
This file is distributed under the same license as the Foo project.
"""
```

The default template used when creating Javascript files for translated strings can also be customized with the REQUIRE_I18N_JS_TEMPLATE setting:

```
REQUIRE_I18N_JS_TEMPLATE = """// Copyright (C) 2015 Me

define(
{0}
);
"""
```

The metadata written in the translated catalog can also be customized with the REQUIRE_I18N_PO_METADATA setting:

```
REQUIRE_I18N_PO_METADATA = {
    'Project-Id-Version': '1.0',
    'Report-Msgid-Bugs-To': 'i18n-bugs@root',
    'Last-Translator': 'Foo <you@root>',
    'Language-Team': '{label} <{code}@root>'
}
```

Note that you have access to {label} and {code} variabels in the Last-Translator section. During compilation {label} is replaced by the language label (Dutch) and {code} is replaced by the language code (nl).

3.1.4 Extract strings

Run the compile_js command to extract the translation strings and generate a catalog for the nl locale in the site domain:

```
manage.py compile_js --no-empty --domain=site --locale=nl
```

This will create two new files:

- locale/templates/LC_MESSAGES/site.pot contains the string resources that were extracted from the Javascript root translation files.
- locale/nl/LC_MESSAGES/site.po is the translated catalog that contains the actual Dutch translations.

3.1.5 Translate strings

Open locale/nl/LC_MESSAGES/site.po with poedit or a text-editor and add translations for the msgid strings. For example:

```
#: static/js/site/nls/root/colors.js:5
msgid "Red"
msgstr "Rood"
```

```
#: static/js/site/nls/root/colors.js:6
msgid "Green"
msgstr "Groen"
```

3.1.6 Compile translations

Run the `compile_js` command again to write the translated strings to the Javascript translation file(s):

```
manage.py compile_js --no-empty --domain=site --locale=nl
```

After running this command you can find the translated Javascript file(s) in the `static/js/site/nls/nl` directory. The contents of `colors.js` would look like this:

```
// Copyright (C) 2015 Me

define(
{
 "redLabel": "Rood",
 "greenLabel": "Groen"
}
);
```

By default it writes the translations to `.js` files but you can also specify `json` with the `--output-type` option to create `.json` files instead:

```
manage.py compile_js --no-empty --output-type=json --domain=site --locale=nl
```

The contents of `colors.json` would look like this:

```
{
 "redLabel": "Rood",
 "greenLabel": "Groen"
}
```

3.1.7 Conclusion

You now have all the files available to localize your require.js application. Simply run the `compile_js` whenever you update your translations or want to support a new language.

3.2 Development

After checkout, install package in active virtualenv:

```
pip install -e .
```

3.2.1 Testing

Running tests with Tox:

```
tox -v
```

Or directly with `django-admin`:

```
django-admin test --settings=require_i18n.tests.settings require_i18n
```

3.2.2 Coverage

To generate a test coverage report using coverage.py:

```
coverage run --source='.' /path/to/virtualenv/bin/django-admin test --settings=require_i18n.tests.set
coverage html
```

The resulting HTML report can be found in the `htmlcov` directory.

3.2.3 Release

Creating a new release on test PyPi:

```
python setup.py sdist upload -r pypitest
```

And on live PyPi:

```
python setup.py sdist upload -r pypi
```