invenio-i18n Documentation

Release 1.1.0

CERN

Nov 06, 2018

Contents

1 User's Guide **3**
- 1.1 Installation . . . 3
- 1.2 Configuration . . . 3
- 1.3 Usage . . . 4
- 1.4 Example application . . . 8

2 API Reference **11**
- 2.1 API Docs . . . 11

3 Additional Notes **15**
- 3.1 Contributing . . . 15
- 3.2 Changes . . . 17
- 3.3 License . . . 17
- 3.4 Contributors . . . 17

Python Module Index **19**

Invenio internationalization module based on Flask-BabelEx.

Features:

- Loading and merging message catalogs.
- Algorithm for detecting a user's locale.
- Views for changing the locale.
- Jinja2 macros and filters for I18N.

Further documentation available at https://invenio-i18n.readthedocs.io/

CHAPTER 1

User's Guide

This part of the documentation will show you how to get started in using Invenio-I18N.

1.1 Installation

Invenio-I18N is on PyPI so all you need is:

```
$ pip install invenio-i18n
```

1.2 Configuration

Configuration for the Invenio internationalization module.

In addition to the configuration variables listed below, Flask-BabelEx also sets system-wide defaults. For further details see:

- https://pythonhosted.org/Flask-BabelEx/#configuration

invenio_i18n.config.**I18N_DEFAULT_REDIRECT_ENDPOINT = None**
Endpoint to redirect if no next parameter is provided.

invenio_i18n.config.**I18N_LANGUAGES = []**
List of tuples of available languages.

Example configuration with english and danish with english as default language:

```
from flask_babelex import lazy_gettext as _
BABEL_DEFAULT_LOCALE = 'en'
I18N_LANGUAGES = (('da', _('Danish')),)
```

Note: You should not include `BABEL_DEFAULT_LOCALE` in this list.

invenio_i18n.config.**I18N_SESSION_KEY = 'language'**
Key to retrieve language identifier from the current session object.

invenio_i18n.config.**I18N_SET_LANGUAGE_URL = '/lang'**
URL prefix for set language view.

Set to `None` to prevent view from being installed.

invenio_i18n.config.**I18N_TRANSLATIONS_PATHS = []**
List of paths to load message catalogs from.

invenio_i18n.config.**I18N_USER_LANG_ATTR = 'prefered_language'**
Attribute name which contains language identifier on the User object.

It is used only when the login manager is installed and a user is authenticated. Set to `None` to prevent selector from being used.

1.3 Usage

Invenio internationalization module.

This module provide features for loading and merging message catalogs. It is built on top of Flask-BabelEx and most external modules should just depend on Flask-BabelEx instead of Invenio-I18N. Only applications in need of loading and merging many message catalogs should integrate this module.

1.3.1 Quick start

First initialize the extension (Flask-BabelEx is also automatically initialized by the extension):

```
>>> from flask import Flask
>>> from flask_babelex import lazy_gettext as _
>>> app = Flask('myapp')
>>> app.config['I18N_LANGUAGES'] = [('cs', _('Czech')), ('da', _('Danish'))]
>>> from invenio_i18n import InvenioI18N
>>> i18n = InvenioI18N(app)
```

You can now use the Flask-BabelEx localization features:

```
>>> from flask_babelex import format_number
>>> with app.test_request_context(headers=[('Accept-Language', 'en')]):
...     format_number(10.1) == '10.1'
True
>>> with app.test_request_context(headers=[('Accept-Language', 'cs')]):
...     format_number(10.1) == '10,1'
True
```

as well as internationalization features:

```
>>> from flask_babelex import gettext
>>> with app.test_request_context(headers=[('Accept-Language', 'en')]):
...     gettext('Language:') == 'Language:'
True
>>> with app.test_request_context(headers=[('Accept-Language', 'cs')]):
...     gettext('Language:') == 'Jazyk:'
True
```

1.3.2 Marking strings for translation

Following is a short overview on how to mark strings in Python code and templates for translation so that they can be automatically extracted:

Python

You specify translations in Python by importing `gettext` or `lazy_gettext` from Flask-BabelEx:

```
>>> from flask_babelex import gettext as _
>>> _('Test') == 'Test'
True
```

For further details and examples see:

- https://pythonhosted.org/Flask-BabelEx/#using-translations
- http://babel.pocoo.org/en/latest/messages.html#working-with-message-catalogs

Jinja2

In templates you can use either the underscore function:

```
>>> from flask import render_template_string
>>> with app.app_context():
...     render_template_string("{{_('Test')}}") == 'Test'
True
```

or the `{% trans %}` tag:

```
>>> with app.app_context():
...     r = render_template_string('{% trans %}Long translation{% endtrans %}')
...     r == 'Long translation'
True
```

For further details and examples see:

- http://jinja.pocoo.org/docs/dev/templates/#i18n

Angular

There is also simple integration for Angular application using Angular-Gettext library. First, you need to mark HTML tags which contain translatable string or expression.

```
<a href="/" translate>Hello {{name}}</a>
```

For further details see:

- https://angular-gettext.rocketeer.be/
- https://github.com/neillc/angular-gettext-babel

1.3.3 Templates

This section only gives a very quick introduction into custom context variables and filters.

Context

The `current_i18n` global variable is available within Jinja2 templates to give access to an instance of *InvenioI18N* attached to current application context.

```
>>> with app.test_request_context(headers=[('Accept-Language', 'en')]):
...     r = render_template_string('{{ current_i18n.language }}')
...     r == 'en'
True
>>> with app.test_request_context(headers=[('Accept-Language', 'en')]):
...     r = render_template_string('{{ current_i18n.timezone }}')
...     r == 'UTC'
True
>>> with app.test_request_context(headers=[('Accept-Language', 'da')]):
...     r = render_template_string('{{ current_i18n.locale }}')
...     r == 'da'
True
```

Filters

There are several useful filters automatically added to the Jinja2 template context:

- `tousertimezone` converts a datetime object to the user's timezone (see *filter_to_user_timezone()*).
- `toutc` converts a datetime object to UTC and drop tzinfo (see *filter_to_utc()*).
- `language_name` converts language code into display name in current locale (see *filter_language_name()*).
- `language_name_local` converts language code into display name in local locale (see *filter_language_name_local()*).

Macros

Invenio-I18N also provides three templates macros that you can use to render a language selector in templates with:

- `language_selector` - Renders a list of links and uses GET requests to change the locale.
- `language_selector_form` - Same as above, but uses POST requests to change the locale.
- `language_selector_dropdown` - Renders a dropdown with languages and uses a POST request to change the locale.

You use the macros by importing one of them from `invenio_i18n/macros/language_selector.html`, for instance:

```
>>> with app.test_request_context():
...     r = render_template_string(
...         '{% from "invenio_i18n/macros/language_selector.html"'
...         '   import language_selector %}'
...         '{{ language_selector() }}'
...     )
```

1.3.4 Working with Message Catalogs

Babel package contains really good documentation which you should read first:

- http://babel.pocoo.org/en/latest/messages.html

Angular-Gettext

This part focuses on how to configure Babel extraction for Angular application in custom `babel-js.ini` file:

```
[angular_gettext: **/static/templates/**/**.html]
```

To make message extraction and catalog extraction easier you can add following aliases to `setup.cfg` (replace `${PACKAGE_PATH}` with package path):

```
[aliases]
extract_messages_js = extract_messages -F babel-js.ini -o \
    ${PACKAGE_PATH}/translations/messages-js.pot
init_catalog_js = init_catalog -D messages-js --input-file \
    ${PACKAGE_PATH}/translations/messages-js.pot
update_catalog_js = update_catalog -D messages-js --input-file \
    ${PACKAGE_PATH}/translations/messages-js.pot
```

1.3.5 Integration with Transifex service

There is a Python package that provides CLI. You can start by installing *Transifex* package and check if the `tx` command is available:

```
$ pip install transifex-client
$ tx --version
0.12.2
```

The integration is configured in `.tx/config` file (replace `${PACKAGE_PATH}` and `${PACKAGE_NAME}`).

```
[main]
host = https://www.transifex.com

[invenio.${PACKAGE_NAME}-messages]
file_filter = ${PACKAGE_PATH}/translations/<lang>/LC_MESSAGES/messages.po
source_file = ${PACKAGE_PATH}/translations/messages.pot
source_lang = en
type = PO

[invenio.${PACKAGE_NAME}-messages-js]
file_filter = ${PACKAGE_PATH}/translations/<lang>/LC_MESSAGES/messages-js.po
source_file = ${PACKAGE_PATH}/translations/messages-js.pot
source_lang = en
type = PO
```

1. Create message catalog

Start by extracting localizable messages from a collection of source files.

```
$ python setup.py extract_messages
$ python setup.py init_catalog -l <lang>
```

If you have localizable Angular messages run commands with `_js` suffix too.

```
$ python setup.py extract_messages_js
$ python setup.py init_catalog_js -l <lang>
```

2. Transifex project

Ensure project has been created on Transifex under the `inveniosoftware` organisation.

3. First push

Push source (`.pot`) and translations (`.po`) to Transifex.

```
$ tx push -s -t
```

Note: From now on do not edit `.po` files localy, but only on Transifex.

4. Fetch changes

Pull translations for a single/all language(s) from Transifex.

```
$ tx pull -l <lang>
$ tx pull -a
```

Check fetched tranlations, commit changes `git commit -a -s -m 'i18n: updates from Transifex'` and create pull-request.

5. Update message catalog

When new localizable messages are introduced or changes, the message catalog needs to be extracted again. Then you need to push the source files to Transifex service which will take care about updating catalogs. At the end pull translations for all languages from Transifex and commit the changes.

```
$ python setup.py extract_messages
$ python setup.py extract_messages_js
$ tx push -s
$ tx pull -a
```

1.4 Example application

A simple example application demonstrating Invenio-I18N language rendering.

First, install and set up the example application:

```
$ pip install -e .[all]
$ cd examples
$ ./app-setup.sh
```

Now start the example application server:

```
$ FLASK_APP=app.py flask run --debugger -p 5000
```

The example application will render "Hello World" and display the language selectors for English, Danish and Spanish. It will allow you to change the text to the given language.

You can uninstall the example application as follows:

```
$ ./app-teardown.sh
```

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API Docs

2.1.1 Extension

Invenio internationalization module.

class invenio_i18n.ext.**InvenioI18N**(*app=None*, *date_formats=None*, *localeselector=None*, *timezoneselector=None*, *entry_point_group='invenio_i18n.translations'*)

Invenio I18N extension.

Initialize extension.

Parameters

- **app** – Flask application.
- **data_formats** – Override default date/time formatting.
- **localeselector** – Callback function used for locale selection. (Default: *invenio_i18n.selectors.get_locale()*)
- **timezoneselector** – Callback function used for timezone selection. (Default: BABEL_DEFAULT_TIMEZONE)
- **entry_point_group** – Entrypoint used to load translations from. Set to None to not load translations from entry points.

get_languages()
Get list of languages.

get_locales()
Get a list of supported locales.

Computes the list using `I18N_LANGUAGES` configuration variable.

init_app(*app*)
Flask application initialization.

The initialization will:

- Set default values for the configuration variables.
- Load translations from paths specified in `I18N_TRANSLATIONS_PATHS`.
- Load translations from `app.root_path>/translations` if it exists.
- Load translations from a specified entry point.
- Add `toutc` and `tousertimezone` template filters.
- Install a custom JSON encoder on app.

init_config(*app*)
Initialize configuration.

iter_languages()
Iterate over list of languages.

language
Get current language code.

locale
Get current locale.

timezone
Get current timezone.

invenio_i18n.ext.**get_lazystring_encoder**(*app*)
Return a JSONEncoder for handling lazy strings from Babel.

Installed on Flask application by default by *InvenioI18N*.

2.1.2 Translation Domain

Flask-BabelEx domain for merging translations from many directories.

class invenio_i18n.babel.**MultidirDomain**(*paths=None*, *entry_point_group=None*, *domain='messages'*)
Domain supporting merging translations from many catalogs.

The domain contains an internal list of paths that it loads translations from. The translations are merged in order of the list of paths, hence the last path in the list will overwrite strings set by previous paths.

Entry points are added to the list of paths before the `paths`.

Initialize domain.

Parameters

- **paths** – List of paths with translations.
- **entry_point_group** – Name of entry point group.
- **domain** – Name of message catalog domain. (Default: `'messages'`)

add_entrypoint(*entry_point_group*)
Load translations from an entry point.

add_path(*path*)
Load translations from an existing path.

get_translations()
Return the correct gettext translations for a request.

This will never fail and return a dummy translation object if used outside of the request or if a translation cannot be found.

has_paths()
Determine if any paths have been specified.

invenio_i18n.babel.**set_locale**(**args*, ***kwds*)
Set Babel localization in request context.

Parameters **ln** – Language identifier.

2.1.3 Jinja2 filters

Babel datetime localization template filters for Jinja.

See full documentation of corresponding methods in Flask-BabelEx: https://pythonhosted.org/Flask-BabelEx/

invenio_i18n.jinja2.**filter_language_name**(*lang_code*)
Convert language code into display name in current locale.

Installed on application as `language_name`.

invenio_i18n.jinja2.**filter_language_name_local**(*lang_code*)
Convert language code into display name in local locale.

Installed on application as `language_name_local`.

invenio_i18n.jinja2.**filter_to_user_timezone**(*dt*)
Convert a datetime object to the user's timezone.

Installed on application as `tousertimezone`.

invenio_i18n.jinja2.**filter_to_utc**(*dt*)
Convert a datetime object to UTC and drop tzinfo.

Installed on application as `toutc`.

2.1.4 Locale/timezone selectors

Default locale and timezone selectors for Flask-BabelEx.

See Flask-BabelEx documentation for corresponding methods.

invenio_i18n.selectors.**get_locale**()
Get locale.

Searches for locale in the following the order:

- User has specified a concrete language in the query string.
- Current session has a language set.
- User has a language set in the profile.
- Headers of the HTTP request.
- Default language from `BABEL_DEFAULT_LOCALE`.

Will only accept languages defined in `I18N_LANGUAGES`.

`invenio_i18n.selectors.`**`get_timezone`**`()`
Get default timezone (i.e. `BABEL_DEFAULT_TIMEZONE`).

2.1.5 Views

Views for Invenio-I18N.

`invenio_i18n.views.`**`create_blueprint`**`(`*register_default_routes=True*`)`
Create Invenio-I18N blueprint.

`invenio_i18n.views.`**`get_redirect_target`**`()`
Get URL to redirect to and ensure that it is local.

`invenio_i18n.views.`**`is_local_url`**`(`*target*`)`
Determine if URL is safe to redirect to.

`invenio_i18n.views.`**`set_lang`**`(`*lang_code=None*`)`
Set language in session and redirect.

2.1.6 Date/time formatting

For formatting date and time using the current locale settings, you may use the methods provided by Flask-BabelEx.

These methods are also available as Jinja filters:

- format_datetime as `datetimeformat`
- format_date as `dateformat`
- format_time as `timeformat`
- format_timedelta as `timedeltaformat`.

CHAPTER 3

Additional Notes

Notes on how to contribute, legal information and changes are here for the interested.

3.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

3.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/inveniosoftware/invenio-i18n/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Invenio-I18N could always use more documentation, whether as part of the official Invenio-I18N docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/inveniosoftware/invenio-i18n/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.1.2 Get Started!

Ready to contribute? Here's how to set up *invenio-i18n* for local development.

1. Fork the *inveniosoftware/invenio-i18n* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/invenio-i18n.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv invenio-i18n
$ cd invenio-i18n/
$ pip install -e .[all]
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

```
$ ./run-tests.sh
```

   The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
      $ git commit -s
    -m "component: title without verbs"
    -m "* NEW Adds your new feature."
    -m "* FIX Fixes an existing issue."
    -m "* BETTER Improves and existing feature."
    -m "* Changes something that should not be visible in release notes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5. Check https://travis-ci.com/inveniosoftware/invenio-i18n/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.1.0 (released 2018-11-06)

- Introduce webpack support.

Version 1.0.0 (released 2018-03-23)

- Initial public release.

3.3 License

3.4 Contributors

- Alizee Pace
- Andrew McCracken
- Diego Rodriguez

- Harris Tzovanakis
- Javier Delgado
- Jiri Kuncar
- Lars Holm Nielsen
- Leonardo Rossi
- Nikos Filippakis
- Paulina Lach
- Sebastian Witowski
- Tibor Simko

Python Module Index

i

invenio_i18n, 4
invenio_i18n.babel, 12
invenio_i18n.config, 3
invenio_i18n.ext, 11
invenio_i18n.jinja2, 13
invenio_i18n.selectors, 13
invenio_i18n.views, 14

Index

A

add_entrypoint() (invenio_i18n.babel.MultidirDomain method), 12
add_path() (invenio_i18n.babel.MultidirDomain method), 12

C

create_blueprint() (in module invenio_i18n.views), 14

F

filter_language_name() (in module invenio_i18n.jinja2), 13
filter_language_name_local() (in module invenio_i18n.jinja2), 13
filter_to_user_timezone() (in module invenio_i18n.jinja2), 13
filter_to_utc() (in module invenio_i18n.jinja2), 13

G

get_languages() (invenio_i18n.ext.InvenioI18N method), 11
get_lazystring_encoder() (in module invenio_i18n.ext), 12
get_locale() (in module invenio_i18n.selectors), 13
get_locales() (invenio_i18n.ext.InvenioI18N method), 11
get_redirect_target() (in module invenio_i18n.views), 14
get_timezone() (in module invenio_i18n.selectors), 14
get_translations() (invenio_i18n.babel.MultidirDomain method), 13

H

has_paths() (invenio_i18n.babel.MultidirDomain method), 13

I

I18N_DEFAULT_REDIRECT_ENDPOINT (in module invenio_i18n.config), 3
I18N_LANGUAGES (in module invenio_i18n.config), 3
I18N_SESSION_KEY (in module invenio_i18n.config), 3
I18N_SET_LANGUAGE_URL (in module invenio_i18n.config), 4
I18N_TRANSLATIONS_PATHS (in module invenio_i18n.config), 4
I18N_USER_LANG_ATTR (in module invenio_i18n.config), 4
init_app() (invenio_i18n.ext.InvenioI18N method), 12
init_config() (invenio_i18n.ext.InvenioI18N method), 12
invenio_i18n (module), 4
invenio_i18n.babel (module), 12
invenio_i18n.config (module), 3
invenio_i18n.ext (module), 11
invenio_i18n.jinja2 (module), 13
invenio_i18n.selectors (module), 13
invenio_i18n.views (module), 14
InvenioI18N (class in invenio_i18n.ext), 11
is_local_url() (in module invenio_i18n.views), 14
iter_languages() (invenio_i18n.ext.InvenioI18N method), 12

L

language (invenio_i18n.ext.InvenioI18N attribute), 12
locale (invenio_i18n.ext.InvenioI18N attribute), 12

M

MultidirDomain (class in invenio_i18n.babel), 12

S

set_lang() (in module invenio_i18n.views), 14
set_locale() (in module invenio_i18n.babel), 13

T

timezone (invenio_i18n.ext.InvenioI18N attribute), 12