Tornado-Babel Documentation

Release 0.1

Openlabs Technologies & Consulting (P) Limited

May 18, 2016

Contents

1 Installation **3**

2 Date Formatting **5**

3 Using Translations **7**
- 3.1 Making translatable applications . . . 7
- 3.2 Extracting Translations . . . 7
- 3.3 Loading Translations . . . 8

4 API Reference **9**
- 4.1 Datetime Formatting Methods . . . 9
- 4.2 Mixin Utilities for Tornado . . . 10
- 4.3 Low level API for message extraction . . . 10

Python Module Index **13**

Tornado-Babel adds i18n and l10n support to tornado with the help of babel and speaklater. It has builtin support for date formatting as well as a very simple and friendly interface to `gettext` translations.

Torando comes bundled with pretty basic locale support basic and does not offer support for merging translation catalogs and several other features most multi language applications require. The module also adds more date and number formatting helpers.

Tornado-Babel also includes a plugin for Babel to extract translatable strings from tornado templates.

CHAPTER 1

Installation

Install the extension with one of the following commands:

```
$ easy_install Tornado-Babel
```

or alternatively if you have pip installed:

```
$ pip install Tornado-Babel
```

CHAPTER 2

Date Formatting

To format dates you can use the `format_datetime()`. Here are some examples:

```
>>> from datetime import datetime
>>> from tornadobabel.locale import Locale
>>> locale = Locale.parse('pt_BR')
>>> locale
<Locale "pt_BR">
>>> dt = datetime(2007, 04, 01, 15, 30)
>>> locale.format_datetime(dt)
u'01/04/2007 15:30:00'
```

With a different locale:

```
>>> locale = Locale.parse('en_US')
>>> locale
<Locale "en_US">
>>> dt = datetime(2007, 04, 01, 15, 30)
>>> locale.format_datetime(dt)
u'Apr 1, 2007 3:30:00 PM'
```

For more format examples head over to the babel documentation.

CHAPTER 3

Using Translations

3.1 Making translatable applications

First you need to mark all the strings you want to translate in your application. To use translation in your request handlers, you can use the mixin `TornadoBabelMixin`:

```
from tornadobabel import locale

class ProfileHandler(TornadoBabelMixin, ReuqestHandler):
    def get_user_locale(self):
        if self.current_user:
            return locale.get(self.current_user.locale)

        # Fallback to browser based locale detection
        return self.get_browser_locale()
```

The mixin adds a shortcut to `translate()` as a property `_()`, which could be used like:

```
class HomePageHandler(TornadoBabelMixin, ReuqestHandler):

    def get(self):
        _ = self._
        return self.write(_("Hello"))
```

3.2 Extracting Translations

After that, it's time to create a `.pot` file. A `.pot` file contains all the strings and is the template for a `.po` file which contains the translated strings. Babel can do all that for you.

First of all you have to get into the folder where you have your application and create a mapping file. For typical applications, this is what you want in there:

```
[python: **.py]
[tornado: **/templates/**.html]
```

Save it as `babel.cfg` or something similar next to your application. Then it's time to run the *pybabel* command that comes with Babel to extract your strings:

```
$ pybabel extract -F babel.cfg -o messages.pot .
```

If you are using the `lazy_gettext()` function you should tell pybabel that it should also look for such function calls:

```
$ pybabel extract -F babel.cfg -k lazy_gettext -o messages.pot .
```

This will use the mapping from the `babel.cfg` file and store the generated template in `messages.pot`. Now we can create the first translation. For example to translate to German use this command:

```
$ pybabel init -i messages.pot -d translations -l de
```

`-d translations` tells pybabel to store the translations in this folder. This is where Flask-Babel will look for translations. Put it next to your template folder.

Now edit the `translations/de/LC_MESSAGES/messages.po` file as needed. Check out some gettext tutorials if you feel lost.

To compile the translations for use, `pybabel` helps again:

```
$ pybabel compile -d translations
```

What if the strings change? Create a new `messages.pot` like above and then let `pybabel` merge the changes:

```
$ pybabel update -i messages.pot -d translations
```

Afterwards some strings might be marked as fuzzy (where it tried to figure out if a translation matched a changed key). If you have fuzzy entries, make sure to check them by hand and remove the fuzzy flag before compiling.

3.3 Loading Translations

Now that you have a translatable application and translations, you can load the translations of your application using `load_gettext_translations()`. The function takes two arguments: *directory* which is the root path of the folder where translations are stored and *domain* (which is used to match against translation file names). In the above example, the directory would be the absolute path of *translations* and the domain would be *messages*.

Babel also provides a powerful feature of merging translations from multiple such directories and domains. This could be used when you use third party modules which may have bundled translations along with it and you want to use them with your application. An example of such a module is **'pycountry'_**:

```
>>> import pycountry
>>> from tornadobabel.locale import load_gettext_translations
>>> load_gettext_translations(pycountry.LOCALES_DIR, 'iso3166')
```

Now load your application translations:

```
>>> load_gettext_translations('locales', 'messages')
```

Tornado-Babel will automatically merge both your translations:

```
>>> pt_BR = locale.get("pt_BR")
>>> pt_BR.translate("United States")
u'Estados Unidos'
```

CHAPTER 4

API Reference

4.1 Datetime Formatting Methods

`Locale.format_datetime`(*datetime=None*, *format='medium'*, *tzinfo=None*)

Return a date formatted according to the given pattern.

Parameters

- **datetime** – the datetime object; if None, the current date and time is used
- **format** – one of "full", "long", "medium", or "short", or a custom date/time pattern
- **tzinfo** – the timezone to apply to the time for display

```
>>> from datetime import datetime
>>> locale = Locale.parse('pt_BR')
>>> locale
<Locale "pt_BR">
>>> dt = datetime(2007, 04, 01, 15, 30)
>>> locale.format_datetime(dt)
u'01/04/2007 15:30:00'
```

`Locale.format_date`(*date=None*, *format='medium'*)

Return a date formatted according to the locale.

Parameters

- **date** – the date or datetime object; if None, the current date is used
- **format** – one of "full", "long", "medium", or "short", or a custom date/time pattern

`Locale.format_time`(*time=None*, *format='medium'*, *tzinfo=None*)

Return a time formatted according to the locale.

Parameters

- **time** – the time or datetime object; if None, the current time in UTC is used
- **format** – one of "full", "long", "medium", or "short", or a custom date/time pattern
- **tzinfo** – the time-zone to apply to the time for display

`Locale.format_timedelta`(*delta*, *granularity='second'*, *threshold=0.85*)

Return a time delta according to the rules of the given locale.

Parameters

- **delta** – a timedelta object representing the time difference to format, or the delta in seconds as an int value
- **granularity** – determines the smallest unit that should be displayed, the value can be one of "year", "month", "week", "day", "hour", "minute" or "second"
- **threshold** – factor that determines at which point the presentation switches to the next higher unit

4.2 Mixin Utilities for Tornado

4.3 Low level API for message extraction

extract

Extract messages from tornado templates

copyright

3. 2012 by Openlabs Technologies & Consulting (P) Limited

license BSD, see LICENSE for more details.

class `tornadobabel.extract.`**`DummyTemplate`**(*template_string*, *name='<string>'*, *loader=None*, *compress_whitespace=None*, *autoescape=<object object>*)

A template object just used to parse the template string. The class resembles tornado.Template but stops at parsing the file.

`tornadobabel.extract.`**`walk`**(*node*)

Given a template node, walk over all its descendants

```
>>> t = DummyTemplate("{{ _('Hello') }}")
>>> len(list(walk(t.file)))
3
>>> t = DummyTemplate("{% if a %}{{ _('Hello') }}{% end %}")
>>> len(list(walk(t.file)))
5
```

`tornadobabel.extract.`**`extract_from_node`**(*expression*, *gettext_functions=None*)

Extract localizable strings from the given Template Expression

Parameters

- **expression** – A node of type tornado.template._Expression
- **gettext_functions** – A list of gettext function names that should be parsed from template

Returns iterator returning tuple in the format babel wants

`tornadobabel.extract.`**`extract_tornado`**(*fileobj*, *keywords*, *comment_tags*, *options*)

Extract messages from Python source code.

Parameters

- **fileobj** – the seekable, file-like object the messages should be extracted from
- **keywords** – a list of keywords (i.e. function names) that should be recognized as translation functions

- **comment_tags** – a list of translator tags to search for and include in the results. (Not implemented yet)
- **options** – a dictionary of additional options (optional)

Returns an iterator over `(lineno, funcname, message, comments)` tuples

Return type `iterator`

t

tornadobabel.extract, 10

D

DummyTemplate (class in tornadobabel.extract), 10

E

extract_from_node() (in module tornadobabel.extract), 10
extract_tornado() (in module tornadobabel.extract), 10

F

format_date() (tornadobabel.locale.Locale method), 9
format_datetime() (tornadobabel.locale.Locale method), 9
format_time() (tornadobabel.locale.Locale method), 9
format_timedelta() (tornadobabel.locale.Locale method), 9

T

tornadobabel.extract (module), 10

W

walk() (in module tornadobabel.extract), 10