# Python I18n L10n

Christian Külker

2023-04-13

## Contents

Internationalization (i18n) is the process of making a program aware of multiple languages. Localization (l10n) refers to the process of adapting your program, once internationalized, to the local language and cultural customs. A locale is a set of internationalization parameters that defines a computer user's language, region, and any special variant preferences that the user wishes to see displayed in the user interface of programs, scripts, or the desktop. Sometimes the locale consists of a language code, like `en`, `de`, but more often it is a combination of a language code and a country/region code, like `en_EN` or `de_DE`. This is because some countries speak more than one language. Often a `codeset` information (which charset to use) and a modifier is added. On `POSIX` systems this results in a string like `language[_territory][.codeset][@modifier]`.

Example:

- German language in Germany written in UTF-8: `de_DE.UTF-8`
- German language in Belgium written in ISO-8859: `de_BE.ISO-8859-1`

- German language in Belgium written in ISO-8859 with the EURO symbol: `de_BE.ISO-8859-15@euro` often shorted to `de_BE5@euro` as the EURO symbol is an option in ISO-8859 but not in UTF-8.
- Application: python3 + gettext

## 1 Internationalization

Often `GNU gettext` or a similar system is used. The procedure consists of several steps:

1. A program is written using special tags or functions known to `gettext`, such as `_`. Each call to print is processed by `gettext`. So print works as a wrapper for `gettext`. The usual `print("text")` becomes `print(_("text"))`.
2. The `gettext` framework provides scripts that can extract strings that are parameters to the `_` function. So in the above case it would extract "text". These strings, usually in English but any other language will do, are called `msgid` and become the ID of a text. So here we have the message ID: `msg_id "text"`. These message IDs are collected in a file with the extension `*.pot` (PO template) and other useful information is also collected.
3. Steps 1 and 2 are basically the internationalization of a program. The next step is the localization. Here `gettext` is used to create a local version. This is basically a copy from `*.pot` to `*.po`. In case of British English it would be `en_GB.UTF-8.po` and in case of German spoken in Germany it would be `de_DE.UTF-8.po`. Next to the `msg_id` entry you will find a `msgstr ""` entry. For a localization to be complete, you need to translate the `msgid` to the `msgstr`. For English this would be `msid "Text"` and `msgstr "Text"`.
4. Theoretically, the localization is done. However, some programs would like to have the translated string database in a binary format. There are also tools in the `gettext` framework to convert the `*.po` file into a binary format. Usually they have the extension `*.mo`.

## 2 GNU gettext

Consider the following Python program `main`:

```
import gettext
_ = gettext.gettext
```

```python
def print_some_strings():
    print(_("Hello world"))
    print(_("Internationalisation"))

if __name__=='__main__':
    print_some_strings()
```

The corresponding `*.pot` file would look like:

```
  # SOME DESCRIPTIVE TITLE.
  # Copyright (C) YEAR ORGANIZATION
  # FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
  #
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 2022-01-28 16:47+0000\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Generated-By: pygettext.py 1.5\n"

#: src/main.py:5
msgid "Hello world"
msgstr ""

#: src/main.py:6
msgid "Internationalisation"
msgstr ""
```

This is translated from reference English (whatever the developer wants) to the target language (for example, English as spoken in the United States of America). In this case, it is just a copy of strings.

```
# English translations for PACKAGE package.
# Copyright (C) 2022 THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# Automatically generated, 2022.
#
```

```
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2022-05-18 06:38+0200\n"
"PO-Revision-Date: 2022-05-20 17:16+0200\n"
"Last-Translator: Automatically generated\n"
"Language-Team: none\n"
"Language: en_US\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=2; plural=(n != 1);\n"

#: src/main.py:5
msgid "Hello world"
msgstr "Hello world"

#: src/main.py:6
msgid "Internationalisation"
msgstr "Internationalization"
```

In short:

2. Create `*.pot`: `xgettext --no-wrap --from-code=UTF-8 --keyword=_ -L Python    --copy`
3. Create `*.po`: `msginit --no-wrap --no-translator --input=main.pot    --locale=en_US`
4. Update `*.po` from `*.pot`: `msgmerge --no-wrap --backup=none --update    main.po main.`
5. Create `*.mo` from `*.po`: `msgfmt -o en_US.mo main.pot`

While the above is possible, it is more common to store the `*.mo` files in other ways, such as

```
locale├──
 de_DE│
   └── LC_MESSAGES│
       ├── main.mo│
       └── main.po├──
 en_US│
   └── LC_MESSAGES│
       ├── main.mo│
       └── main.po├──
 ja_JP│
```

```
    └── LC_MESSAGES│
        ├── main.mo│
        └── main.po└──
main.pot
```

## 3 Jinja2

To use `*.mo` files in Python with `jinja2`, you can use the extension: `jinja2.ext.i18n`.

```
env = Environment(loader=file_loader,extensions=["jinja2.ext.i18n"])
```

This allows the use of `gettext` and `ngettext` function calls in the template.

```
{{ getext('Hello World') }}
```

It is also possible to configure `jinja2` to use a default call or to rename the function call.

## 4 Limitations

This document describes the basics of `GNU gettext`. Advanced usage like **numbers**, **singular** and **plural** or `ngettext` has not been touched.

## 5 History

| Version | Date | Notes |
|---|---|---|
| 0.1.2 | 2023-04-13 | Improve writing |
| 0.1.1 | 2023-01-18 | Add Jinja2 section |
| 0.1.0 | 2022-05-30 | Initial release |

## 6 Disclaimer of Warranty

THERE IS NO WARRANTY FOR THIS INFORMATION, DOCUMENTS AND PROGRAMS, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE INFORMATION, DOCUMENT OR THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE INFORMATION, DOCUMENTS AND PROGRAMS IS WITH YOU. SHOULD THE INFORMATION, DOCUMENTS OR PROGRAMS PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

## 7 Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE INFORMATION, DOCUMENTS OR PROGRAMS AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE INFORMATION, DOCUMENTS OR PROGRAMS (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE INFORMATION, DOCUMENTS OR PROGRAMS TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.