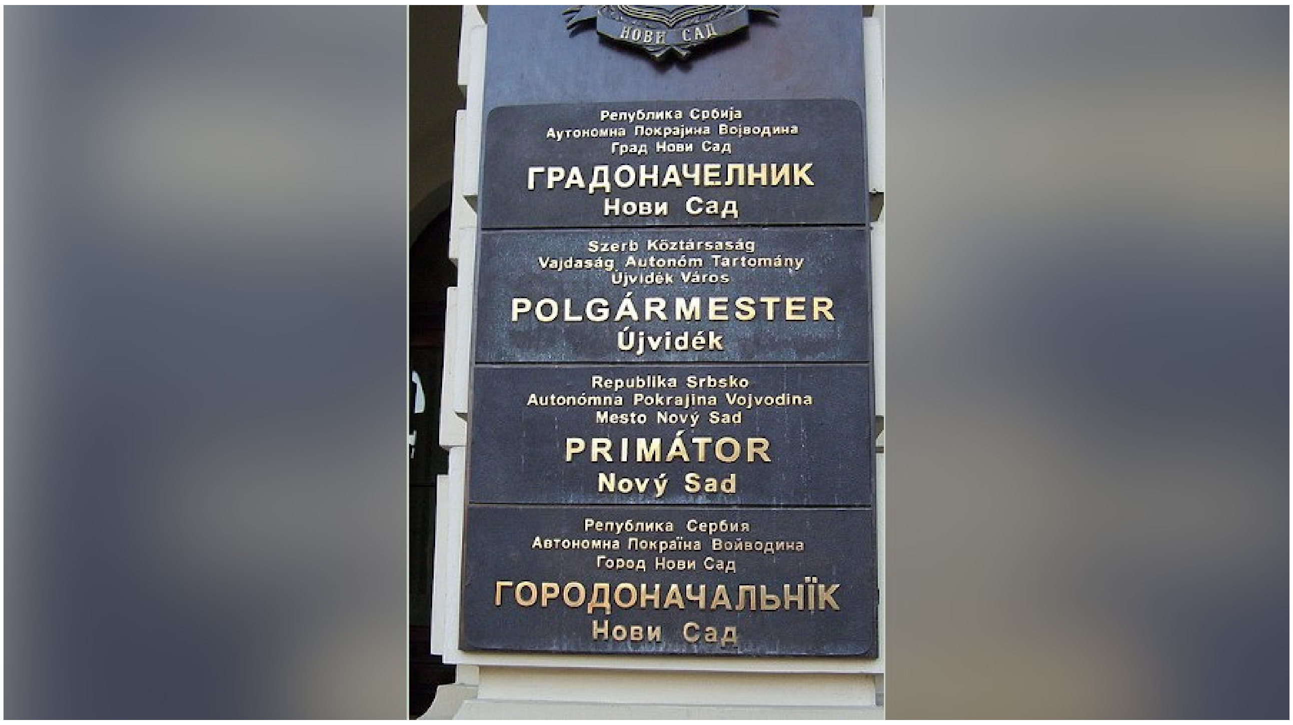
Simple text markup internationalization (i18n)

Gettext approach

Nikolaj Potashnikov

Version 1.0, 2022-01-11

Table of contents


НОВИ САД
Република Србија
Аутономна Покрајина Војводина
Град Нови Сад
ГРАДОНАЧЕЛНИК
Нови Сад
Szerb Köztársaság
Vajdaság Autonóm Tartomány
Újvidék Város
POLGÁRMESTER
Újvidék
Republika Srbsko
Autonómna Pokrajina Vojvodina
Mesto Nový Sad
PRIMÁTOR
Nový Sad
Републикa Сербия
Автономна Покраïна Войводина
Город Нови Сад
ГОРОДОНАЧАЛЬНЇК
Нови Сад

Introduction

Перейти к русской версии

Several years ago my colleague posted an article about making presentations in Asciidoctor. Since then, we don’t use any other approach.

Some time ago there have appeared a problem of translating presentations into several languages and have them synchronized. The solution appeared to be so simple and mature that I decided to describe it in this post.

This solution is syntax-unaware. This means, it doesn’t matter whether we use Asciidoc, other light text markup or even mixed format (with my favourite Plantuml or any other diagrams, for example). The serious limitations of this approach are (1) the translator shouldn’t break markup structure and (2) we can’t directly apply machine translation to the original text.

The solution uses Translation Toolkit and standard GNU Gettext tools.

To make this solution clear this article has English and Russian versions. Its repository contains some simple automation that synchronizes translation, creates printing version (pdf, docx, odt) and creates Markdown file for publishing to Habr.

In my recent article on testing documentation I didn’t pay much attention to text linters, because the focus was on approaches, not exact tools. Still these tools are great. To fill the gap, I’ll use vale for this article.

The idea

Gettext assumes that key strings for translation are original messages.

Gettext uses files with `.po` extension (PO — Portable Object) to keep both original and translated messages. A great number of editors allow to edit such files either in a single user or collaborative environments.

The idea of Translation Toolkit is to use blocks of adjacent lines as such strings.

Take this example:

```
.Winter is

* snow
* frost

* Christmas
* New Year
```

It has three blocks of adjacent strings. Translation Toolkit will extract three key strings for translation and will put it to a file with the `.pot` extension (`.pot` stands for `.po` Template).

Putting or removing line breaks in this example you may make any number of strings from 1 to 5. This depends on convenience to the translator.

Using a `.pot` file as a template Gettext creates (updates) `.po` files for all required languages. Translators handle exactly these files. After that Translation Toolkit takes (1) a `.po` file with translation, (2) the original file and creates a final translated file.

The Process

The process consists of the following steps.

- Initial steps to get initial translations in one or several languages.
- Updating translation steps to synchronize translation with the modified original text.

The following diagrams assume that the original file is `i18n-adoc.adoc` and the translation goes to `i18n-adoc-ru.adoc`.

Initial steps

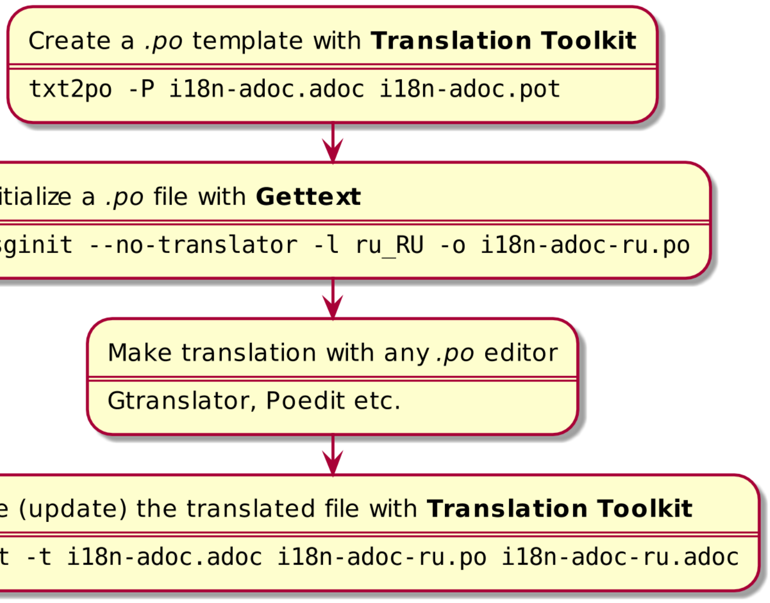
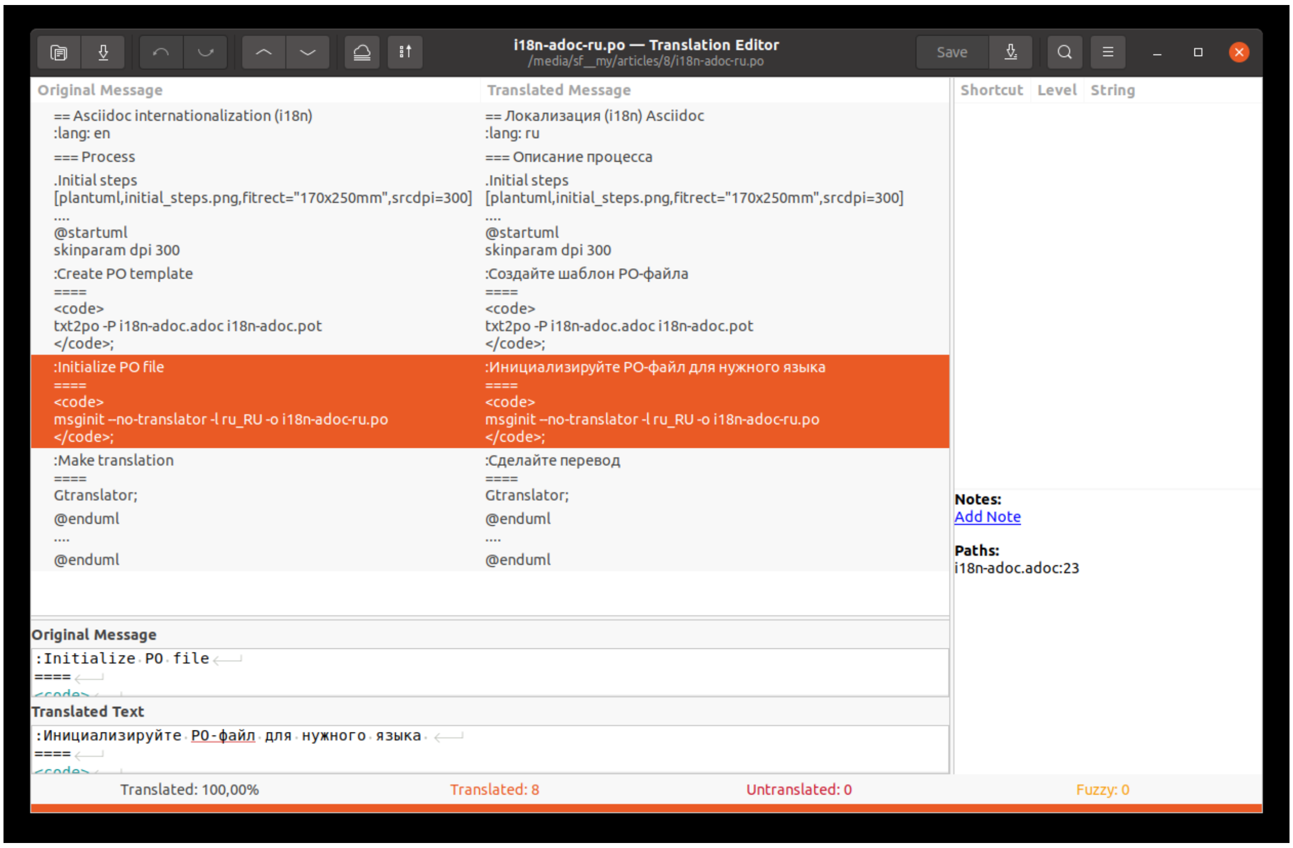

Create a *.po* template with **Translation Toolkit**
txt2po -P i18n-adoc.adoc i18n-adoc.pot
Initialize a *.po* file with **Gettext**
msginit --no-translator -l ru_RU -o i18n-adoc-ru.po
Make translation with any *.po* editor
Gtranslator, Poedit etc.
Create (update) the translated file with **Translation Toolkit**
po2txt -t i18n-adoc.adoc i18n-adoc-ru.po i18n-adoc-ru.adoc

Figure 1. Initial steps

There is a vast number of editors for translating `.po` files. The following screenshot shows Gtranslator interface. I prefer Poedit, although the way it replaces backtick with tick while applying machine translation is annoying.


i18n-adoc-ru.po — Translation Editor
/media/sf__my/articles/8/i18n-adoc-ru.po

Original Message	Translated Message
== Asciidoc internationalization (i18n) :lang: en	== Локализация (i18n) Asciidoc :lang: ru
=== Process	=== Описание процесса
.Initial steps [plantuml,initial_steps.png,fitrect="170x250mm",srcdpi=300] .... @startuml skinparam dpi 300	.Initial steps [plantuml,initial_steps.png,fitrect="170x250mm",srcdpi=300] .... @startuml skinparam dpi 300
:Create PO template ==== <code> txt2po -P i18n-adoc.adoc i18n-adoc.pot </code>;	:Создайте шаблон PO-файла ==== <code> txt2po -P i18n-adoc.adoc i18n-adoc.pot </code>;
:Initialize PO file ==== <code> msginit --no-translator -l ru_RU -o i18n-adoc-ru.po </code>;	:Инициализируйте PO-файл для нужного языка ==== <code> msginit --no-translator -l ru_RU -o i18n-adoc-ru.po </code>;
:Make translation ==== Gtranslator;	:Сделайте перевод ==== Gtranslator;
@enduml	@enduml ....
@enduml	@enduml

Shortcut Level String

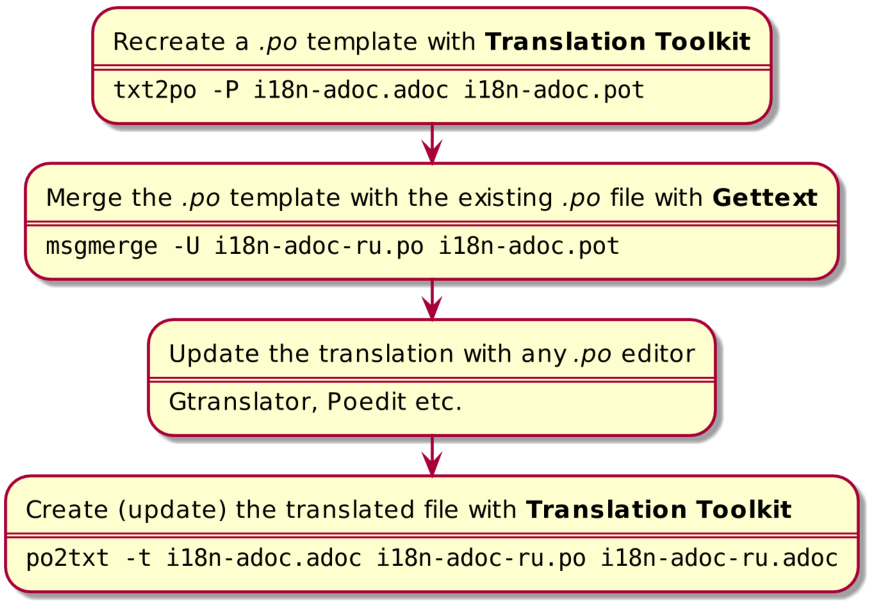
Notes:
Add Note

Paths:
i18n-adoc.adoc:23

Original Message
```
:Initialize PO file
====
<code>
```
Translated Text
```
:Инициализируйте PO-файл для нужного языка
====
<code>
```
Translated: 100,00%   Translated: 8   Untranslated: 0   Fuzzy: 0

Figure 2. Translating in Gtranslator

Updating translation


Recreate a *.po* template with **Translation Toolkit**
```
txt2po -P i18n-adoc.adoc i18n-adoc.pot
```
Merge the *.po* template with the existing *.po* file with **Gettext**
```
msgmerge -U i18n-adoc-ru.po i18n-adoc.pot
```
Update the translation with any *.po* editor

Gtranslator, Poedit etc.

Create (update) the translated file with **Translation Toolkit**
```
po2txt -t i18n-adoc.adoc i18n-adoc-ru.po i18n-adoc-ru.adoc
```

Figure 3. Updating translation

Some notes

1. In our documentation, we often reuse source i18n strings just to be sure that names of interface elements in documentation are equal to the same names in our application. We generate these i18n strings automatically in the following format:

   ```
   :main-menu-documents: Documents
   :main-menu-documents-my: My
   ...
   ```

   We include such a file in the Asciidoc document like `include i17n-{lang}.adoc[]`. Now there is no need to use attributes. Just translate `include i17n-en.adoc[]` to `include i17n-ru.adoc[]`.

2. When `gettext` updates the `.po` files, it uses fuzzy search. If you slightly change the original text, you won’t lose its translation. It will be just marked as flaky.

3. It’s easy to check whether translation file is up-to-date with a Gettext utility `msgfmt`.

   ```
   msgfmt --statistics i18n-adoc-ru.po
   ```

   It shows the number of translated strings, the number of strings that need checking and the number of untranslated strings.

Conclusion

- Translation Toolkit and Gettext make the process of documentation internationalization quite efficient.
- Simple text markups are not so simple. To take full advantage of it, one needs a certain level of skills. Try to give a translator the `.po` file. How many of them will be ready to make a translation? Or they'll ask you for text in a more traditional format like Microsoft Word.
- Quality control: 58 translated messages, 0 errors, 0 warnings and 0 suggestions in 1 file.