Internationalizing WordPress Plugin

Version 0.1 (2007-11-05)

Table of Contents

Introduction

Internationalization of software is important stage whether or not a software is mature enough. Not long ago, I decided to internationalize my WordPress plugin, On This Day (OTD). I will show you how I planned and made it happening. Since I had given up Windows, therefore this time I only use tools on Linux. The last time, I did for a Python program. That program runs on Windows, so I used the popular tool, poEdit. poEdit is quite great tool. It can update your message catalog directly from source or from an updated PO template. It is available on many operating systems, however it crashes on my Fedora 7. I don't know why and I don't know how to fix it, nevertheless the crash is a good thing for me. I have to learn using CLI tools to do localization for OTD.

All the works in this article can be done by using poEdit, but I won't mention how to do with it in this article. This article only focuses on how to have translations for WordPress plugin.

Terms Used

I will use the following terms throughout this article and here are my explanations for them. I may have unclear thoughts about them, please feel free to correct me.

developer
: In this article, this represents the plugin's developer.

translator
: In this article, this represents the plugin's translator.

i18n (Internationalization)
: The process of enabling softwares having capabilities to have translations, to show currency in different units, to show local date and time, etc. This is developers' task.

l10n (Localization)
: Using the efforts above to do translating, to switch to another unit, etc. This usually translators' task.

gettext
: A GNU tool for translation. Also the function for getting a translated message.

locale
: A name represents adoptions of a language, a unit, a date format, etc. This name can be a language name, a language name plus a country name or a location name.

textdomain, package
: This is nothing to do with locale. If your systems has many sub-systems, says libraries or like plugins in WordPress. You don't want to put all informations in one place, that would be messy. So you need to separate them into a small pieces, each one has own place to store its information. textdomain is like namespace. In WordPress, WordPress core has its own textdomain, so should plugins. The name of the textdomain usually is the same as the name of package.

original message

The plain message in default locale of package.

msgid

Usually it is same as original message. However, we should keep in mind: since it is called `msgid`, that means it is an index for messages; therefore that could be just a **id** not a message.

Here is a quick explanation:
For printing out a date in different places of UI. Sometimes we may have lager space for such “August 23rd, 2007”, sometimes we don't have. We can have only “2007-08-23”. Therefore they are different date format.

Original output	Original `msgid`	Adjusted `msgid`	Translated `msgstr` (en_US)	Translated `msgstr` (zh_TW)
August 23rd, 2007	`F jS, Y`	`long-date-fmt`	`F jS, Y`	`Y年n月j日`
2007-08-23	`Y-m-d`	`short-date-fmt`	`Y-m-d`	`Y-m-d`

PHP `date()` format

This example also tells us, always should create an locale for original language. There is another hidden point here: what if gettext fail to be initialized? Normally, this would not be a problem if `msgid` is also the message; but it is a problem now, since `msgid` is an `ID` only. The solution is a gettext function with a failback option.

```
function ___($msgid, $domain, $failback) {
    $translated = __($msgid, $domain);
    return ($translated == $msgid) ? $failBack : $translated;
    }
```

How it works? It is simple. When the gettext function returns a translated message which is the same as `$msgid`, this version of gettext function will return `$failBack`. Some people would be prefer to call it `$default`.

msgstr

the translated message after translating into another language

`_`, `__`, `_e`, `___` and `ngettext`

Usually, we have `_` and `__`. First one uses `msgid` to find a translated message, second one returns same as `msgid`. Why do we need the second one? Under some circumstances, like an array with things like states

```
$s = array('state001', 'state002', 'state003');
```

You may need to use them in switch case statements, so you don't want to translate them. When you need to show current state to users, your users wouldn't want to read “state001”. Then we need to translate like

```
echo __($currentState);
```

However we don't have such `__` by default in WordPress. In WordPress `__` is like `_`. The third one is just a shorthand for echoing translated message. The fourth one `___` is called failbacked gettext by me, it has been explained earlier.

`ngettext` handles pluralizations of messages. In WordPress, it is `__ngettext`.

source
The source code. In this article, it means PHP here.

`.pot`
PO Template or message catalog template. Since it is called “template”, you should no need to modify it. It should only be duplicated by translators once. After duplication, that would be merging, or called updating.

`.po`
PO, Portable Object or message catalog. That's the file translators take care of. It has the same structure as `.pot` since it is duplicated from `.pot`.

`.mo`
MO, Machine Object or binary message catalog. It's for machine reading.

message
Consists of a `msgid` and a `msgstr`.

translated message
It means a message has non-empty `msgstr`.

fuzzy message
It means the `msgid` might be changed in source and the translators need to make changes to `msgstr` as well if any.

untranslated message
It means `msgstr` is empty.

obsolete message
It means that message is no longer available.

i18n support from WordPress for plugin developers

As a WordPress plugin developer, you only need to call a function for initializing, which is

```
function load_textdomain($domain, $mofile);
```

`$domain` is the text domain of your plugin, and `$mofile` is where you can file the binary message catalog files. After initializing, you need gettext functions to get translated messages. In WordPress world, that would be

```
function __($text, $domain = 'default')
function _e($text, $domain = 'default')
```

`$text` actually is the `msgid`.

Internationalization of OTD

In this section, I will describe the environment of OTD development for internalization.

Environment

I list all the things I used, it is just for mentioning. They are not required for you, you can always use alternative tools to achieve same goals.

System and softwares

- Fedora 7
- WordPress 2.3.1
- Vim
- xgettext, msgmerge and msgfmt

Files and directory structure

This structure is exactly the same as in subversion repository.

- `trunk/`
  - `po/` - Stores `.pot`, `.po`.
  - `OnThisDay/` - The directory to be copied to `wp-content/plugins/`.
    - `locale/` - Stores `.mo`.
    - `OnThisDay.php`
    - `OptionsPage.php`
  - `bmgc.conf` - To be sourced by tool script.

Tool script – bmgc

I wrote this tool script to simplify localization. You can download this script in its trunk. Note that this tool script is quite depending on the structure described in previous sub-section. It has two files a configuration file and the script:

`bmgc.sample.conf` has some variables you need to assign. You should copy it to trunk with name `bmgc.conf`.

`bmgc` can take care of some tasks of localization. Put this in a searchable directory and run it in `trunk/`. It is used in 4 steps: build, merge, generate and check.

> `bmgc b [version]` - **build** `.pot` from sources
> This mode not only using xgettext to extract msgids but also replacing entries of header like `Project-Id-Version`, `Report-Msgid-Bugs-To`, `POT-Creation-Date` and `Content-Type`. First two will use values in `bmgc.conf`, third uses current time, and last it is always `UTF-8`.
> If the values in `bmgc.conf` are set properly, their will be a `$DOMAIN.pot` in `$PO_DIR`.

`bmgc m <all|locale>` - **merge** `.pot` into .po
You can use this mode to create new locale.po, actually `$DOMAIN-locale.po`; or update your message catalog after you have new PO template.
If there is no such `$DOMAIN-locale.po` in `$PO_DIR`, then `bmgc` will duplicate `$DOMAIN.pot` for you. No matter it is used for creating new or merging, it replaces `Project-Id-Version` with same entry from `$DOMAIN-locale.po`.

`bmgc g <all|locale>` - **generate** `.mo`
Once you finish translating, you need to compile message catalog into binary message catalog for testing. It will replace `PO-Revision-Date` with current time and `Last-Translator` with the value you set in `bmgc.conf`.
After generating, it moves the generated file to `LOCALE_DIR`.

`bmgc c <all|locale>` - **check** `.po`
It simply do the same thing as generating `.mo` but not stores generated file. It should be used by developers for checking completion of translation.

Internationalizing

From this section, I will describe how did I do for internationalization and localization. This task definitely is a developer's task.

Deciding a `DOMAIN` for you plugin.

Limitations

- It has to be unique.
- It can't be the same as `DOMAIN`s of other plugins.
- It can't be "`default`." That's the `DOMAIN` of WordPress.
- It has be confined by file system naming constraints. Because `.mo` filename consists of `DOMAIN-locale.mo`.

How to choose?

- You plugin's name.
- You plugin's main installing directory.

For example, the `DOMAIN` for OTD plugin is "`OnThisDay`" ans it is also the directory name to be copied to `wp-content/plugins`.

Deciding the path to .mo files

It means where you store the binary message catalog files (`.mo`). The directory should be separated from you sources. My suggestions is a directory called "`locale`" with you sources.

- `locale/`
  - `en_US.mo`
  - `zh_TW.mo`

    - and so on
- `OnThisDay.php`

Preparing the source

Step 1 Cooperating with WordPress

You need two lines to get i18n support from WordPress. Open you main plugin's source. Here is the two-line code for `OnThisDay.php`,

```
1. define('ON_THIS_DAY_DOMAIN', 'OnThisDay');
2. load_plugin_textdomain(ON_THIS_DAY_DOMAIN, 'wp-
   content/plugins/' . ON_THIS_DAY_DOMAIN . '/locale');
```

In line 1, using `define` to have a constant `ON_THIS_DAY_DOMAIN`. Line 2, asking WordPress to load your `.mo`. The `load_plugin_textdomain` accepts two parameters: 1) `DOMAIN`, 2) path to the `.mo` files. The path is a relative path to your WordPress installed directory, it usually likes `/home/username/public_html/blog/`. In this case, it results `/home/username/public_html/blog/wp-content/plugins/OnThisDay/locale`. That's where you should put the `.mo` files to.

Step 2 Warping messages with `__` or `_e`

Normally, you put put the original message into `__()` with textdomain. This completely is a boring task. If you use Vim, here is my tips for this task. First, adding new keymaps like:

```
vmap <F2> "xd"="__("<CR>P"xp")"<CR>p
vmap <F2><F2> "xd"="_e("<CR>P"xp")"<CR>p
vmap <F3> "xd"="<?php _e('"<CR>P"xp"='); ?>"<CR>p
```

Select `'Original Message'`, then press `<F2>`. It will be replaced with `__('Original Message')`, or `<F2><F2>` for getting `_e('Original Message')`. Note there is no tailing semicolon `;` to be appended. If you want to translate a message in HTML like

```
<h1>Page Title</h1>
```

Just select `Page Title`, then hit `<F3>`. However, these keymaps don't add the text domain parameter. You can use command like

```
%s/_\([_e]\)(\(['"]\)\(.\{-}\)\2)/_\1(\2\3\2,
ON_THIS_DAY_DOMAIN)/g
```

after you finishing wrapping.

Step 3 Generate `.pot`

In this step, we need to use `xgettext` to extract `msgid`s from source. We don't directly use `xgettext`, but `bmgc` script, instead. Before using it, we need to setup a configuration file

`bmgc.conf` within `trunk/`. Duplicates `bmgc.sample.conf` with name `trunk/bmgc.conf`, then modify it.

Next is run `bmgc b [version]`. You should have `trunk/po/` and `trunk/po/DOMAIN.pot`. The `version` is optional.

Step 4 Verifying translators' .po

In terms of quality, developers should load `.mo` and make sure the UI remains in the design. Sometimes, the translation could break the UI design. For example, translated message is much longer than original message, that may cause unexpected result. Therefore, developers should verify in this point, although this should be translators' task.

Another thing is make sure translators do translate all messages. You can run `bmgc c all` to check all locales.

Step 5 Fixing bugs from localization

Translators may report some bugs, problems, difficulties in translating. For example, the i18n task is insufficient to afford the needs for some languages.

Localizing

To be a translator of a software, you need to follow some things. Developers may ask you to update your message catalog directly from source or from a PO template which they generate. As a translators can also help spotting some bugs, since they need to make sure each message is translated properly.

Deciding the locale name

You need deciding the same naming as WordPress does. The format is `ll_CC`. `ll` is the language code, then a underscore _, last part `CC` is the country code. The complete `.po` filename is `DOMAIN-ll_CC.po`.

Steps

Step 1 Setting up your environment for translating

Having the same environment as developer's should be a good idea. However, not every one uses same OS, softwares, the minimal requirement is you can generate `.mo` and test it in anyway.

Step 2 Generating .pot

Translators may need to generate `.pot` on you own if the develops don't want to run one-line command. However, the developers may ask you to update your `.po` with the .pot they provide. They don't want you to update `.po` from source or .pot that you make because they may be developing new features and those messages are not ready for translating.

You need clearly ask the developers which updating method you should take. If you need to generate `.pot`, then run `bmgc b [version]` under `trunk/`, or use poEdit to generate one from source.

Step 3 Preparing DOMAIN-locale.po

Once you get the `.pot`, duplicate it with extension `.po` and append the locale name, then

- Open it.
- Replace `FIRST AUTHOR` with you if you are the first person works on this `.po`. You may need to replace `Report-Msgid-Bugs-To` with your address in `bmgc.conf` if you need to take the responsibility of fixing mistakes.

Step 4 Translating

If there is a message marked as fuzzy, remove the fuzzy mark line after you re-translating it.

Step 5. Generating `.mo` and Testing

Using `bmgc g <locale>` can

- Replace `Last-Translator` with your name and email.
- Replace `PO-Revision-Date` with current time.
- Move the generated file to `LOCALE_DIR`.

Make sure `.mo` and plugin source are in right place, then change locale setting in `wp-config.php`. Keep doing Step 4 and Step 5 until all messages have been correctly translated. Once you finishing translating, send the `.po` to developers.

Step 6. Updating message catalog

When the source has been updated, there may have few changes. So, translators need to make corresponding changes to their `.po`. Translators may need to do:

- generate new PO template `bmgc g [version]`. If developers send you one, then skip this step.
- run `bmgc m <locale>`.
- read the output message.

Then go back to Step 4.