

# MimerDesk Localization Guidelines

---

Teemu Arina <teemu at ionstream.fi>, Mikael Söderholm <mikke at ionstream.fi>

This document describes the MimerDesk localization guidelines: How to localize MimerDesk and how to code localization ready applications.

## Contents

<b>1</b>	<b>Overview of Internationalization &amp; Localization</b>	<b>2</b>
<b>2</b>	<b>What might be the shortest way to localize MimerDesk?</b>	<b>2</b>
<b>3</b>	<b>So what did the developers choose to fix this Localization problem?</b>	<b>3</b>
<b>4</b>	<b>Gettext system</b>	<b>3</b>
4.1	How it works . . . . .	3
4.2	Translating dynamic content (applications) . . . . .	3
4.2.1	Tools for editing .po files . . . . .	5
4.2.2	Other useful resources . . . . .	6
4.3	Translating static content (templates) . . . . .	6
4.3.1	How it works . . . . .	6
4.3.2	How to translate templates . . . . .	6
4.4	How to use the gettext system in your applications . . . . .	6
<b>5</b>	<b>Translating button images</b>	<b>7</b>
5.1	buttons.txt . . . . .	8
5.2	menucategory.txt . . . . .	8
5.3	menuitem.txt . . . . .	8
<b>6</b>	<b>How to add new languages</b>	<b>8</b>
<b>7</b>	<b>Tools for translators included in MimerDesk</b>	<b>9</b>
7.1	Introduction . . . . .	9
7.1.1	bin/check_po_syntax [language] . . . . .	9
7.1.2	bin/compare_po_entries [language] . . . . .	9
7.1.3	bin/addlang2tmpls -lang=xx TEMPLATES_DIRECTORY . . . . .	9
7.1.4	bin/addlang2locale -lang=xx LOCALE_DIRECTORY . . . . .	9

7.1.5	bin/rnton FILES...	10
<b>8</b>	<b>Documentation system</b>	<b>10</b>
8.1	How it works	10
8.2	How to translate documents	10

## 1 Overview of Internationalization & Localization

Today it is hard to find software that supports many languages. Due to the cost of translating software and difficulties in production of a fully localized application for wide variety of languages from western to Asian most developers skip this absolutely necessary feature completely.

*Internationalization* (I18N, "i" + 18 letters + "n") *a.k.a. Globalization* means designing and developing a software product to function in multiple locales <sup>1</sup>. This process involves identifying the locales that must be supported, designing features which support those locales, and writing code that functions equally well in any of the supported locales. This means a lot of work with character sets and proper transmission. Since computers were created to work with a small character set in the beginning, there are historical reasons why bringing chinese, english and russian text to the end user is so difficult. There are also problems between languages. For example, you don't read from left to right in every language. Some languages have symbols that present a complete word or a part of a word. In addition, there are difficulties when we talk about searching documents: Chinese and some other Asian languages lack spaces and that's why it's difficult to create a good search index that supports all of the languages. You might also know that date, currencies and how you display them correctly in the current locale is different in many countries. How frustrating for a programmer.

*Localization* (L10N, "l" + 10 letters + "n") means modifying or adapting a software product to fit the requirements of a particular locale. This process includes (but may not be limited to) translating the user interface, documentation and packaging, changing dialog box geometries, customizing features (if necessary), and testing the translated product to ensure that it still works (at least as well as the original). There are many issues in localization as well that render the job very difficult to complete successfully. For example, "Desktop for user J.Random" is "J.Randomin työpöytä" and "Desktop for user Kalle Kivinen" is "Kalle Kivisen työpöytä" in Finnish. These examples show that everything cannot be directly translated in every situation. There is need for different text formatting and parsing in many languages, not just Finnish. This is especially true for dynamic generation of text.

## 2 What might be the shortest way to localize MimerDesk?

Ok. Now for the fun part. How MimerDesk developers are going to complete this difficult task? One approach might be maintaining several code trees for every language and code all the special cases differently in every code tree. This might work for a couple of languages if the program is small. No need for tricky stuff. But the maintaining of several code trees of the same product should soon ring the bells that there is something done wrong here. When the program size grows and clients ask support for more languages everything crashes on the developers and the development slows down. You guessed it, we are definitely not going to take this path.

<sup>1</sup>Locale: A set of conventions affected or determined by human language and customs, as defined within a particular geo-political region. These conventions include (but are not necessarily limited to) the written language, formats for dates, numbers and currency, sorting orders, etc.

### 3 So what did the developers choose to fix this Localization problem?

We are no way language experts. We are no way maniacs who try to implement a perfect localization approach because that would take a decade to complete. We are lazy. That's what we are and that's why we are going to implement a system that is near perfect but easy to use at the same time. Our system consists of the following parts:

- Gettext system similar to Unix standard localization approach for dynamically generated text
- Gettext system similar to Unix standard localization approach for static text inside the templates
- Button generator for generating the translations of the hundreds of button images in MimerDesk
- Template parser that creates translated templates for every language into database for speed (No need to parse templates every time a page is loaded)
- XML based documentation system that enables the transform of the documentation to several formats and languages

## 4 Gettext system

### 4.1 How it works

The Gettext system is quite simple. First you use the `new_gettext()` function to create an object. This is done by giving parameters to the function so it knows what language translation to read and what the program is (from what directory to load the translation). Then you use `gettext()` in place of the normal strings you have in your application. You give the original text as the parameter (like '*Welcome to MimerDesk*'. MimerDesk uses English as the original language) and the function returns the translated text if it was found. If no translation is found `gettext()` will return the parameter you gave.

The Gettext system loads .po files for the application. Po files contain the original text and the translation for every language. Po files have their own syntax defined by a standard described in the next section.

### 4.2 Translating dynamic content (applications)

When an application is gettext ready it is fairly simple to translate the content. There is a folder for localization of programs which is called *locale*. The folder includes a sub folder for every program. A program folder has a .po file for every language, where the first part of the filename is a language code. When making a new translation of the program, you should copy the original file, en.po, to a .po file with the correct language code. When you open an en.po file you should see a few lines with info about the file which is not relevant for the translator and should be left untouched. After the info lines comes the text to be translated. Each translatable entry consists of three parts, the first line shows on which line the text appears in the program code : `#. calendar.html:49`, in this case line 49, this line should not be edited. The second line contains the original text, and it is also the the id for the text in the code: `msgid "Personal - Calendar"` , this line should not be edited, if you edit this line the

program will fail to find the correct translation. The third line is the text that will be translated: *msgstr "Personal - Calendar"*, as you see it is the same as the id line because the file was copied from the en.po. In the following example is the first translated entry from the program calendar and the file fi.po:

```
#. calendar.html:49
msgid "Personal - Calendar"
msgstr "Henkilökohtaiset - Kalenteri"
```

Every entry in the file should be translated like this one.

Some entries can have some special handles to retrieve dynamic data from the program, for example in the same calendar program in file fi.po:

```
#. calendar.html:233
#, c-format
msgid "%s's calendar"
msgstr "Kalenteri: %s"
```

Here the three lines should be left untouched and the fourth line should be translated. The example output of the program might be: *Kalenteri: Pekka*. Things like the %s should never be edited but used so that the end result will look good in the final output, figure out what the meaning of these are and translate so that the result is right in the language in question. The Swedish translation of the use of these special symbols would look like this:

```
#. calendar.html:233
#, c-format
msgid "%s's calendar"
msgstr "kalender för %s"
```

You should also note that there can be other *%[letter]* codes used as well like %d. Just treat them similar to the %s example. View **printf(3)** manual page if you want to know more about these magic formatting codes.

Also, because the character " is special (it identifies when a line starts and when it ends) you have to treat it specially. Escape it with the symbol \:

```
#. files.html:245
#, c-format
msgid "File \"%s\" not found!"
msgstr "Tiedostoa \"%s\" ei löytynyt!"
```

This will translate to *Tiedostoa "%s" ei löytynyt!*

Logically, this makes the symbol \ a special character, so if you want to use it you have to treat it specially:

```
#. files.html:230
#, c-format
msgid "Folder \\users\\%s does not exist!"
msgstr "Hakemisto \\users\\&s ei ole olemassa!"
```

This will translate to *Hakemisto \users\%s ei ole olemassa!*.

Another special flag you might notice is the fuzzy flag. Here is an example of a fuzzy entry:

```
#. users.html:230
#, fuzzy
msgid "Username not specified!"
msgstr "Username not specified!"
```

This special flag indicates, that the translation is untranslated, incomplete or the translator is not so sure about the correct translation. Once the translator has translated the entry and is happy with it, she should remove the `#, fuzzy` line:

```
#. users.html:230
msgid "Username not specified!"
msgstr "Käyttäjänimeä ei ole määritetty!"
```

You can also add your own comments, if you want:

```
# I'm not so sure about this... well
#. users.html:230
msgid "Username not specified!"
msgstr "Käyttäjänimeä ei ole määritetty!"
```

You should be aware that if you edit the `.po` files in a windows environment the files don't work as such because a windows specific newline character is created in the end of every line (`\r\n`). This end-of-line is different in unix systems (`\n`). This is not a problem in the official MimerDesk development because the files are usually checked and all such line endings are converted to unix line endings. If you work with the developers of MimerDesk, try to find a way to save the file in unix text format or use the *rinton* utility before sending translated files to developers.

*Note:* If you run a test environment of your own, you should check for errors like this if you have made translations and the `po` file isn't working as you thought it should work.

#### 4.2.1 Tools for editing `.po` files

**KBabel** (<http://i18n.kde.org/tools/kbabel>) is a very good tool for editing `.po` files. Main part is a powerful and comfortable PO file editor which features full navigation capabilities, full editing functionality, possibility to search for translations in different dictionaries, spell (uses `ispell`) and syntax checking, showing diffs and many more. Also included is a "Catalog Manager", a file manager view which helps keeping an overview of PO files. KBabel will help you to translate fast and also keep consistent translations. You can even go through the complete translation by only looking for fuzzy entries.

**MuLi** (<http://muli.sourceforge.net>) is a nice alternative to KBabel.

**gtranslator** (<http://www.gtranslator.org>) is a gettext `po` file editor for the GNOME desktop environment. It handles all kind/metamorphic forms of `po` files like compiled gettext `po` files (`gmo/mo` files), compressed `po` files (`po.gz/po.bz2` etc.) and features many comfortable features like find, replace, auto translation, learning, messages table, easy navigation and easy editing of the messages.

**KTranslator** (<http://www.geocities.com/bilibao>) is a basic PO file editor.

**poEdit** (<http://poedit.sourceforge.net> ) is cross-platform gettext catalogs (.po files) editor. It is built with wxWindows toolkit and can run on any platform supported by it (although it was only tested on Unix with GTK+ and Windows). It aims to provide more convenient approach to editing catalogs than launching vi and editing the file by hand.

**emacs** contains a gettext editing mode for PO files if you are a regular emacs user.

#### 4.2.2 Other useful resources

**GNU Gettext home page** (<http://www.gnu.org/software/gettext> ) This is the original GNU Gettext home page. Contains the very complete manual about GNU Gettext and about the tools included in it.

**Mimers brunn translator tools** (<http://mimersbrunn.sourceforge.net> ) contains a couple nice tools for translators.

**The Translation Project** (<http://www.iro.umontreal.ca/contrib/po/HTML/index.html> ) is an effort to get all software translators together.

### 4.3 Translating static content (templates)

#### 4.3.1 How it works

First the template system reads in all the template files and parses the templates searching for language IDs (Numbers surrounded with "%". Example: %33%). Then it creates a template entry into the database for each of the languages based on the PO files containing the language ID as msgid. When MimerDesk generates dynamic content it reads the template for the specified language directly from the database instead of parsing every template each time separately and adding the translation. Parsing the templates during the installation is faster than on the fly each time the program is running.

#### 4.3.2 How to translate templates

The template system in MimerDesk is very similar to the application system. You translate po files but this time the po files are located in *templates* directory. The templates directory contains various folders which each contain a subfolder called *locale*. The locale directory contains the po files for translation. The translation process for translating po files of templates is a little bit different but you can use the same tools and methods for editing the files:

- Template PO files do not contain c-formatted strings. There is no dynamically generated text inside the template strings so c-formatting is not needed
- You can't use multi-line entries as you can use in application po files (dynamic content)
- A translation must exist for each language ID or the templates will broke up. Installation will warn if a translation is missing

### 4.4 How to use the gettext system in your applications

- Open your program

- Add a new global called *\$TRANS* into *use vars*
- After *authenticate* function, add the following lines:

```
$TRANS = lib::MimerDesk->new_gettext(
program => 'yourprogramname',
language => $config{'language'});
$APPLICATION = $TRANS->gettext("Cat-Prog");
```

- It is obvious that you change 'yourprogramname' (which is by the way the subdirectory you have to create under directory *locale/*) and 'Category - Program Name' to what ever your program is
- Where ever you have dynamically generated text in your application, use the *gettext()* function like this:

```
# print 'Dynamic text'; change to -->
print $TRANS->gettext('Dynamic text');

# If you have variables inside strings:
# $text = "Text $name"; change to -->
$text = sprintf($TRANS->gettext("Text %s"), $name);

# Naturally you know how sprintf() works =>
# print "Text $name"; Use printf()-->
printf($TRANS->gettext("Text %s"), $name);
```

- Caution: Avoid passing complete sentences / words as input to *gettext()*. Example "Don't do this":

```
@array = qw(User Name Rights);
$TRANS->gettext($_) foreach @array;
```

- If you use variables as input to the *gettext()* function, you have to add the needed entries by hand into the .po files. You can avoid this by coding you application so that you don't need to do this
- Create .po files into *locale/yourprogramname* using *extra/pl2po.pl* script. Use it like this:

```
bin/pl2po.pl -l --target=locale yourprogram.html
```

## 5 Translating button images

It's probably just pain for your mental health and for your hands to manually translate the hundreds of images included in MimerDesk with a paintbrush application. That's why we created Buttongen (<http://freshmeat.net/projects/buttongen>), a simple tool for generating buttons. Buttongen, MimerDesk tiles and scripts for MimerDesk purpose are not included in MimerDesk distribution or separately for public because the tool is not completely finished yet. Anyway, it's possible to translate the buttons by adding new files to directory *locale\_buttons* based on the english version (e.g. `cp -rf locale_buttons/en locale_buttons/it` for *Italy*) and sending the files to the MimerDesk team.

*locale\_buttons* contains directories which each contain the following files:

### 5.1 buttons.txt

This file has the format of "*English translation = Translation of the string*". Example:

Show groups = Show groups

In finnish:

Show groups = Näytä ryhmät

### 5.2 menucategory.txt

This file has the format of "*button image = button name = Translation*". Example:

con-end = midconfig = Config

In finnish:

con-end = midconfig = Asetukset

### 5.3 menuitem.txt

This file has the format of "*button name = Translation*". Example:

botsessions = Sessions

In finnish:

botsessions = Istunnot

## 6 How to add new languages

Adding new languages to MimerDesk is easy. Basically you duplicate the english .po files and pictures as your translation, translate the files and add a line to mimerdesk.languages.

1. Find out the short two letter specification code for the language you are translating (e.g *it for Italy and fi for Finnish*). You will need it many times
2. In the directory *static/pictures*, create a new directory with the same name as the new language code. Copy the contents from directory *static/pictures/en* to the new directory (e.g. `cp -r static/pictures/en static/pictures/it` for Italy)
3. Do the same thing in *static/pictures2* (e.g. `cp -r static/pictures2/en static/pictures2/it` for Italy)
4. Go to the *locale/* directory. Create a new .po file (*copy the contents of en.po*, e.g. `cp en.po it.po` for Italy) for the new language for every MimerDesk module in the subdirectories or use *addlang2locale* utility



5. Go to the *templates/* directory. Create a new .po file (*copy the contents of en.po, e.g. cp en.po it.po for Italy*) for the new language for every MimerDesk module in the subdirectory *locale/* of the subdirectories or use *addlang2tpls* utility
6. Add the language and browser character-set to *config/mimerdesk.languages*
7. Start translating. Refer to the other sections in this document to know how to do it

## 7 Tools for translators included in MimerDesk

### 7.1 Introduction

MimerDesk included a couple useful tools for translators to maintain their translations. These tools are especially handy when updating an out-dated translation or verifying that the po files follow the correct syntax rules. Some of these tools require the GNU Gettext to be installed.

#### 7.1.1 **bin/check\_po\_syntax [language]**

This command checks the syntax of the po files and if all files passed, displays the number of translated messages and number of fuzzy messages.

If no parameter is given it checks po files of all of the languages. If language id is given (example: fi for finnish) it checks only po files of a particular language. This utility is useful for finding any syntactic typos in the po files and for finding out how many untranslated (fuzzy) entries there are still left.

#### 7.1.2 **bin/compare\_po\_entries [language]**

This command compares language po files against en.po in each directory. This tool is good for finding entries that are duplicate (same msgid for two or more entries) or entries that are in en.po but not in the translated po file. This helps you to update out-dated translation which might be missing some new entries added to the po files.

If no parameter is given it compares po files of all of the languages. If language id is given (example: fi for finnish) it compares only po files of a particular language. Your translation is probably complete if all the entries are translated and this tool is not producing any errors.

#### 7.1.3 **bin/addlang2tpls -lang=xx TEMPLATES\_DIRECTORY**

This tool is good for adding the new po files to every *locale/* directory in *templates/* based on the original *en.po* files. Lang parameter takes a two letter language id (example: fi for finnish) and TEMPLATES\_DIRECTORY is the directory where the templates are located, e.g. *templates/*.

#### 7.1.4 **bin/addlang2locale -lang=xx LOCALE\_DIRECTORY**

This tool is good for adding the new po files to every *directory in locale/* based on the original *en.po* files. Lang parameter takes a two letter language id (example: fi for finnish) and LOCALE\_DIRECTORY is the directory where the locales are located, e.g. *locale/*.

### 7.1.5 bin/rnton FILES...

If you have edited the po files in Windows and you were not able to save them in the unix text format, then this utility is just for you. It does the conversion of the line endings for you. FILES is a list of files to convert separated with spaces.

## 8 Documentation system

Not yet ready. We are planning to do a XML based documentation system.

### 8.1 How it works

### 8.2 How to translate documents