GNU gettext tools, version 0.11.2

Native Language Support Library and Tools
Edition 0.11.2, 22 April 2002

Ulrich Drepper
Jim Meyering
François Pinard
Bruno Haible

Copyright © 1995, 1996, 1997, 1998, 2001, 2002 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Foundation.

1 Introduction

> This manual is still in *DRAFT* state. Some sections are still empty, or almost. We keep merging material from other sources (essentially e-mail folders) while the proper integration of this material is delayed.

In this manual, we use *he* when speaking of the programmer or maintainer, *she* when speaking of the translator, and *they* when speaking of the installers or end users of the translated program. This is only a convenience for clarifying the documentation. It is *absolutely* not meant to imply that some roles are more appropriate to males or females. Besides, as you might guess, GNU `gettext` is meant to be useful for people using computers, whatever their sex, race, religion or nationality!

This chapter explains the goals sought in the creation of GNU `gettext` and the free Translation Project. Then, it explains a few broad concepts around Native Language Support, and positions message translation with regard to other aspects of national and cultural variance, as they apply to to programs. It also surveys those files used to convey the translations. It explains how the various tools interact in the initial generation of these files, and later, how the maintenance cycle should usually operate.

Please send suggestions and corrections to:

Internet address:

`bug-gnu-gettext@gnu.org`

Please include the manual's edition number and update date in your messages.

1.1 The Purpose of GNU `gettext`

Usually, programs are written and documented in English, and use English at execution time to interact with users. This is true not only of GNU software, but also of a great deal of commercial and free software. Using a common language is quite handy for communication between developers, maintainers and users from all countries. On the other hand, most people are less comfortable with English than with their own native language, and would prefer to use their mother tongue for day to day's work, as far as possible. Many would simply *love* to see their computer screen showing a lot less of English, and far more of their own language.

However, to many people, this dream might appear so far fetched that they may believe it is not even worth spending time thinking about it. They have no confidence at all that the dream might ever become true. Yet some have not lost hope, and have organized themselves. The Translation Project is a formalization of this hope into a workable structure, which has a good chance to get all of us nearer the achievement of a truly multi-lingual set of programs.

GNU `gettext` is an important step for the Translation Project, as it is an asset on which we may build many other steps. This package offers to programmers, translators and even users, a well integrated set of tools and documentation. Specifically, the GNU `gettext` utilities are a set of tools that provides a framework within which other free packages may produce multi-lingual messages. These tools include

- A set of conventions about how programs should be written to support message catalogs.

- A directory and file naming organization for the message catalogs themselves.
- A runtime library supporting the retrieval of translated messages.
- A few stand-alone programs to massage in various ways the sets of translatable strings, or already translated strings.
- A special mode for Emacs[1] which helps preparing these sets and bringing them up to date.

GNU `gettext` is designed to minimize the impact of internationalization on program sources, keeping this impact as small and hardly noticeable as possible. Internationalization has better chances of succeeding if it is very light weighted, or at least, appear to be so, when looking at program sources.

The Translation Project also uses the GNU `gettext` distribution as a vehicle for documenting its structure and methods. This goes beyond the strict technicalities of documenting the GNU `gettext` proper. By so doing, translators will find in a single place, as far as possible, all they need to know for properly doing their translating work. Also, this supplemental documentation might also help programmers, and even curious users, in understanding how GNU `gettext` is related to the remainder of the Translation Project, and consequently, have a glimpse at the *big picture*.

1.2 I18n, L10n, and Such

Two long words appear all the time when we discuss support of native language in programs, and these words have a precise meaning, worth being explained here, once and for all in this document. The words are *internationalization* and *localization*. Many people, tired of writing these long words over and over again, took the habit of writing *i18n* and *l10n* instead, quoting the first and last letter of each word, and replacing the run of intermediate letters by a number merely telling how many such letters there are. But in this manual, in the sake of clarity, we will patiently write the names in full, each time. . .

By *internationalization*, one refers to the operation by which a program, or a set of programs turned into a package, is made aware of and able to support multiple languages. This is a generalization process, by which the programs are untied from calling only English strings or other English specific habits, and connected to generic ways of doing the same, instead. Program developers may use various techniques to internationalize their programs. Some of these have been standardized. GNU `gettext` offers one of these standards. See Chapter 10 [Programmers], page 80.

By *localization*, one means the operation by which, in a set of programs already internationalized, one gives the program all needed information so that it can adapt itself to handle its input and output in a fashion which is correct for some native language and cultural habits. This is a particularisation process, by which generic methods already implemented in an internationalized program are used in specific ways. The programming environment puts several functions to the programmers disposal which allow this runtime configuration. The formal description of specific set of cultural habits for some country, together with all

[1] In this manual, all mentions of Emacs refers to either GNU Emacs or to XEmacs, which people sometimes call FSF Emacs and Lucid Emacs, respectively.

associated translations targeted to the same native language, is called the *locale* for this language or country. Users achieve localization of programs by setting proper values to special environment variables, prior to executing those programs, identifying which locale should be used.

In fact, locale message support is only one component of the cultural data that makes up a particular locale. There are a whole host of routines and functions provided to aid programmers in developing internationalized software and which allow them to access the data stored in a particular locale. When someone presently refers to a particular locale, they are obviously referring to the data stored within that particular locale. Similarly, if a programmer is referring to "accessing the locale routines", they are referring to the complete suite of routines that access all of the locale's information.

One uses the expression *Native Language Support*, or merely NLS, for speaking of the overall activity or feature encompassing both internationalization and localization, allowing for multi-lingual interactions in a program. In a nutshell, one could say that internationalization is the operation by which further localizations are made possible.

Also, very roughly said, when it comes to multi-lingual messages, internationalization is usually taken care of by programmers, and localization is usually taken care of by translators.

1.3 Aspects in Native Language Support

For a totally multi-lingual distribution, there are many things to translate beyond output messages.

- As of today, GNU `gettext` offers a complete toolset for translating messages output by C programs. Perl scripts and shell scripts will also need to be translated. Even if there are today some hooks by which this can be done, these hooks are not integrated as well as they should be.
- Some programs, like `autoconf` or `bison`, are able to produce other programs (or scripts). Even if the generating programs themselves are internationalized, the generated programs they produce may need internationalization on their own, and this indirect internationalization could be automated right from the generating program. In fact, quite usually, generating and generated programs could be internationalized independently, as the effort needed is fairly orthogonal.
- A few programs include textual tables which might need translation themselves, independently of the strings contained in the program itself. For example, RFC 1345 gives an English description for each character which the `recode` program is able to reconstruct at execution. Since these descriptions are extracted from the RFC by mechanical means, translating them properly would require a prior translation of the RFC itself.
- Almost all programs accept options, which are often worded out so to be descriptive for the English readers; one might want to consider offering translated versions for program options as well.
- Many programs read, interpret, compile, or are somewhat driven by input files which are texts containing keywords, identifiers, or replies which are inherently translatable. For example, one may want `gcc` to allow diacriticized characters in identifiers or use

translated keywords; '`rm -i`' might accept something else than '`y`' or '`n`' for replies, etc. Even if the program will eventually make most of its output in the foreign languages, one has to decide whether the input syntax, option values, etc., are to be localized or not.

- The manual accompanying a package, as well as all documentation files in the distribution, could surely be translated, too. Translating a manual, with the intent of later keeping up with updates, is a major undertaking in itself, generally.

As we already stressed, translation is only one aspect of locales. Other internationalization aspects are system services and are handled in GNU `libc`. There are many attributes that are needed to define a country's cultural conventions. These attributes include beside the country's native language, the formatting of the date and time, the representation of numbers, the symbols for currency, etc. These local *rules* are termed the country's locale. The locale represents the knowledge needed to support the country's native attributes.

There are a few major areas which may vary between countries and hence, define what a locale must describe. The following list helps putting multi-lingual messages into the proper context of other tasks related to locales. See the GNU `libc` manual for details.

Characters and Codesets

The codeset most commonly used through out the USA and most English speaking parts of the world is the ASCII codeset. However, there are many characters needed by various locales that are not found within this codeset. The 8-bit ISO 8859-1 code set has most of the special characters needed to handle the major European languages. However, in many cases, the ISO 8859-1 font is not adequate: it doesn't even handle the major European currency. Hence each locale will need to specify which codeset they need to use and will need to have the appropriate character handling routines to cope with the codeset.

Currency

The symbols used vary from country to country as does the position used by the symbol. Software needs to be able to transparently display currency figures in the native mode for each locale.

Dates

The format of date varies between locales. For example, Christmas day in 1994 is written as 12/25/94 in the USA and as 25/12/94 in Australia. Other countries might use ISO 8061 dates, etc.

Time of the day may be noted as *hh*:*mm*, *hh*.*mm*, or otherwise. Some locales require time to be specified in 24-hour mode rather than as AM or PM. Further, the nature and yearly extent of the Daylight Saving correction vary widely between countries.

Numbers

Numbers can be represented differently in different locales. For example, the following numbers are all written correctly for their respective locales:

```
12,345.67       English
12.345,67       German
 12345,67       French
```

```
1,2345.67       Asia
```

Some programs could go further and use different unit systems, like English units or Metric units, or even take into account variants about how numbers are spelled in full.

Messages

The most obvious area is the language support within a locale. This is where GNU `gettext` provides the means for developers and users to easily change the language that the software uses to communicate to the user.

Components of locale outside of message handling are standardized in the ISO C standard and the SUSV2 specification. GNU `libc` fully implements this, and most other modern systems provide a more or less reasonable support for at least some of the missing components.

1.4 Files Conveying Translations

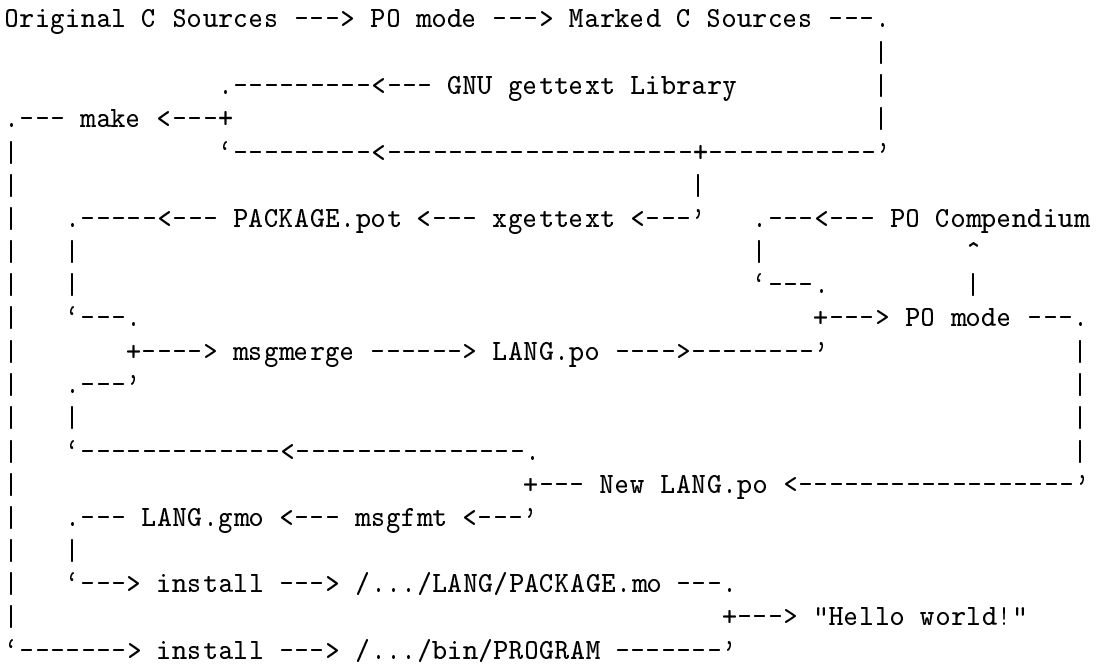
The letters PO in '`.po`' files means Portable Object, to distinguish it from '`.mo`' files, where MO stands for Machine Object. This paradigm, as well as the PO file format, is inspired by the NLS standard developed by Uniforum, and first implemented by Sun in their Solaris system.

PO files are meant to be read and edited by humans, and associate each original, translatable string of a given package with its translation in a particular target language. A single PO file is dedicated to a single target language. If a package supports many languages, there is one such PO file per language supported, and each package has its own set of PO files. These PO files are best created by the `xgettext` program, and later updated or refreshed through the `msgmerge` program. Program `xgettext` extracts all marked messages from a set of C files and initializes a PO file with empty translations. Program `msgmerge` takes care of adjusting PO files between releases of the corresponding sources, commenting obsolete entries, initializing new ones, and updating all source line references. Files ending with '`.pot`' are kind of base translation files found in distributions, in PO file format.

MO files are meant to be read by programs, and are binary in nature. A few systems already offer tools for creating and handling MO files as part of the Native Language Support coming with the system, but the format of these MO files is often different from system to system, and non-portable. The tools already provided with these systems don't support all the features of GNU `gettext`. Therefore GNU `gettext` uses its own format for MO files. Files ending with '`.gmo`' are really MO files, when it is known that these files use the GNU format.

1.5 Overview of GNU `gettext`

The following diagram summarizes the relation between the files handled by GNU `gettext` and the tools acting on these files. It is followed by somewhat detailed explanations, which you should read while keeping an eye on the diagram. Having a clear understanding of these interrelations will surely help programmers, translators and maintainers.



The indication '`PO mode`' appears in two places in this picture, and you may safely read it as merely meaning "hand editing", using any editor of your choice, really. However, for those of you being the lucky users of Emacs, PO mode has been specifically created for providing a cozy environment for editing or modifying PO files. While editing a PO file, PO mode allows for the easy browsing of auxiliary and compendium PO files, as well as for following references into the set of C program sources from which PO files have been derived. It has a few special features, among which are the interactive marking of program strings as translatable, and the validatation of PO files with easy repositioning to PO file lines showing errors.

As a programmer, the first step to bringing GNU `gettext` into your package is identifying, right in the C sources, those strings which are meant to be translatable, and those which are untranslatable. This tedious job can be done a little more comfortably using emacs PO mode, but you can use any means familiar to you for modifying your C sources. Beside this some other simple, standard changes are needed to properly initialize the translation library. See Chapter 3 [Sources], page 18, for more information about all this.

For newly written software the strings of course can and should be marked while writing it. The `gettext` approach makes this very easy. Simply put the following lines at the beginning of each file or in a central header file:

```
#define _(String) (String)
#define N_(String) String
#define textdomain(Domain)
#define bindtextdomain(Package, Directory)
```

Doing this allows you to prepare the sources for internationalization. Later when you feel ready for the step to use the `gettext` library simply replace these definitions by the following:

```
#include <libintl.h>
#define _(String) gettext (String)
#define gettext_noop(String) String
#define N_(String) gettext_noop (String)
```

and link against '`libintl.a`' or '`libintl.so`'. Note that on GNU systems, you don't need to link with `libintl` because the `gettext` library functions are already contained in GNU libc. That is all you have to change.

Once the C sources have been modified, the `xgettext` program is used to find and extract all translatable strings, and create a PO template file out of all these. This '*package*`.pot`' file contains all original program strings. It has sets of pointers to exactly where in C sources each string is used. All translations are set to empty. The letter `t` in '`.pot`' marks this as a Template PO file, not yet oriented towards any particular language. See Section 4.1 [xgettext Invocation], page 28, for more details about how one calls the `xgettext` program. If you are *really* lazy, you might be interested at working a lot more right away, and preparing the whole distribution setup (see Chapter 12 [Maintainers], page 102). By doing so, you spare yourself typing the `xgettext` command, as `make` should now generate the proper things automatically for you!

The first time through, there is no '*lang*`.po`' yet, so the `msgmerge` step may be skipped and replaced by a mere copy of '*package*`.pot`' to '*lang*`.po`', where *lang* represents the target language. See Chapter 5 [Creating], page 32 for details.

Then comes the initial translation of messages. Translation in itself is a whole matter, still exclusively meant for humans, and whose complexity far overwhelms the level of this manual. Nevertheless, a few hints are given in some other chapter of this manual (see Chapter 11 [Translators], page 96). You will also find there indications about how to contact translating teams, or becoming part of them, for sharing your translating concerns with others who target the same native language.

While adding the translated messages into the '*lang*`.po`' PO file, if you do not have Emacs handy, you are on your own for ensuring that your efforts fully respect the PO file format, and quoting conventions (see Section 2.2 [PO Files], page 10). This is surely not an impossible task, as this is the way many people have handled PO files already for Uniforum or Solaris. On the other hand, by using PO mode in Emacs, most details of PO file format are taken care of for you, but you have to acquire some familiarity with PO mode itself. Besides main PO mode commands (see Section 2.3 [Main PO Commands], page 13), you should know how to move between entries (see Section 2.4 [Entry Positioning], page 14), and how to handle untranslated entries (see Section 6.4 [Untranslated Entries], page 40).

If some common translations have already been saved into a compendium PO file, translators may use PO mode for initializing untranslated entries from the compendium, and also save selected translations into the compendium, updating it (see Section 6.11 [Compendium], page 49). Compendium files are meant to be exchanged between members of a given translation team.

Programs, or packages of programs, are dynamic in nature: users write bug reports and suggestion for improvements, maintainers react by modifying programs in various ways. The fact that a package has already been internationalized should not make maintainers shy of adding new strings, or modifying strings already translated. They just do their job the best they can. For the Translation Project to work smoothly, it is important that

maintainers do not carry translation concerns on their already loaded shoulders, and that translators be kept as free as possible of programming concerns.

The only concern maintainers should have is carefully marking new strings as translatable, when they should be, and do not otherwise worry about them being translated, as this will come in proper time. Consequently, when programs and their strings are adjusted in various ways by maintainers, and for matters usually unrelated to translation, `xgettext` would construct '*package*`.pot`' files which are evolving over time, so the translations carried by '*lang*`.po`' are slowly fading out of date.

It is important for translators (and even maintainers) to understand that package translation is a continuous process in the lifetime of a package, and not something which is done once and for all at the start. After an initial burst of translation activity for a given package, interventions are needed once in a while, because here and there, translated entries become obsolete, and new untranslated entries appear, needing translation.

The `msgmerge` program has the purpose of refreshing an already existing '*lang*`.po`' file, by comparing it with a newer '*package*`.pot`' template file, extracted by `xgettext` out of recent C sources. The refreshing operation adjusts all references to C source locations for strings, since these strings move as programs are modified. Also, `msgmerge` comments out as obsolete, in '*lang*`.po`', those already translated entries which are no longer used in the program sources (see Section 6.5 [Obsolete Entries], page 40). It finally discovers new strings and inserts them in the resulting PO file as untranslated entries (see Section 6.4 [Untranslated Entries], page 40). See Section 6.1 [msgmerge Invocation], page 36, for more information about what `msgmerge` really does.

Whatever route or means taken, the goal is to obtain an updated '*lang*`.po`' file offering translations for all strings.

The temporal mobility, or fluidity of PO files, is an integral part of the translation game, and should be well understood, and accepted. People resisting it will have a hard time participating in the Translation Project, or will give a hard time to other participants! In particular, maintainers should relax and include all available official PO files in their distributions, even if these have not recently been updated, without exerting pressure on the translator teams to get the job done. The pressure should rather come from the community of users speaking a particular language, and maintainers should consider themselves fairly relieved of any concern about the adequacy of translation files. On the other hand, translators should reasonably try updating the PO files they are responsible for, while the package is undergoing pretest, prior to an official distribution.

Once the PO file is complete and dependable, the `msgfmt` program is used for turning the PO file into a machine-oriented format, which may yield efficient retrieval of translations by the programs of the package, whenever needed at runtime (see Section 8.3 [MO Files], page 75). See Section 8.1 [msgfmt Invocation], page 70, for more information about all modes of execution for the `msgfmt` program.

Finally, the modified and marked C sources are compiled and linked with the GNU `gettext` library, usually through the operation of `make`, given a suitable '`Makefile`' exists for the project, and the resulting executable is installed somewhere users will find it. The MO files themselves should also be properly installed. Given the appropriate environment variables are set (see Section 9.3 [End Users], page 79), the program should localize itself automatically, whenever it executes.

The remainder of this manual has the purpose of explaining in depth the various steps outlined above.

2 PO Files and PO Mode Basics

The GNU `gettext` toolset helps programmers and translators at producing, updating and using translation files, mainly those PO files which are textual, editable files. This chapter stresses the format of PO files, and contains a PO mode starter. PO mode description is spread throughout this manual instead of being concentrated in one place. Here we present only the basics of PO mode.

2.1 Completing GNU `gettext` Installation

Once you have received, unpacked, configured and compiled the GNU `gettext` distribution, the '`make install`' command puts in place the programs `xgettext`, `msgfmt`, `gettext`, and `msgmerge`, as well as their available message catalogs. To top off a comfortable installation, you might also want to make the PO mode available to your Emacs users.

During the installation of the PO mode, you might want to modify your file '`.emacs`', once and for all, so it contains a few lines looking like:

```
(setq auto-mode-alist
      (cons '("\\.po\\'\\|\\.po\\." . po-mode) auto-mode-alist))
(autoload 'po-mode "po-mode" "Major mode for translators to edit PO files" t)
```

Later, whenever you edit some '`.po`' file, or any file having the string '`.po.`' within its name, Emacs loads '`po-mode.elc`' (or '`po-mode.el`') as needed, and automatically activates PO mode commands for the associated buffer. The string *PO* appears in the mode line for any buffer for which PO mode is active. Many PO files may be active at once in a single Emacs session.

If you are using Emacs version 20 or newer, and have already installed the appropriate international fonts on your system, you may also tell Emacs how to determine automatically the coding system of every PO file. This will often (but not always) cause the necessary fonts to be loaded and used for displaying the translations on your Emacs screen. For this to happen, add the lines:

```
(modify-coding-system-alist 'file "\\.po\\'\\|\\.po\\."
                            'po-find-file-coding-system)
(autoload 'po-find-file-coding-system "po-mode")
```

to your '`.emacs`' file. If, with this, you still see boxes instead of international characters, try a different font set (via Shift Mouse button 1).

2.2 The Format of PO Files

A PO file is made up of many entries, each entry holding the relation between an original untranslated string and its corresponding translation. All entries in a given PO file usually pertain to a single project, and all translations are expressed in a single target language. One PO file *entry* has the following schematic structure:

white-space
`#` *translator-comments*
`#.` *automatic-comments*
`#:` *reference...*

```
#, flag...
msgid untranslated-string
msgstr translated-string
```

The general structure of a PO file should be well understood by the translator. When using PO mode, very little has to be known about the format details, as PO mode takes care of them for her.

Entries begin with some optional white space. Usually, when generated through GNU `gettext` tools, there is exactly one blank line between entries. Then comments follow, on lines all starting with the character `#`. There are two kinds of comments: those which have some white space immediately following the `#`, which comments are created and maintained exclusively by the translator, and those which have some non-white character just after the `#`, which comments are created and maintained automatically by GNU `gettext` tools. All comments, of either kind, are optional.

After white space and comments, entries show two strings, namely first the untranslated string as it appears in the original program sources, and then, the translation of this string. The original string is introduced by the keyword `msgid`, and the translation, by `msgstr`. The two strings, untranslated and translated, are quoted in various ways in the PO file, using `"` delimiters and `\` escapes, but the translator does not really have to pay attention to the precise quoting format, as PO mode fully takes care of quoting for her.

The `msgid` strings, as well as automatic comments, are produced and managed by other GNU `gettext` tools, and PO mode does not provide means for the translator to alter these. The most she can do is merely deleting them, and only by deleting the whole entry. On the other hand, the `msgstr` string, as well as translator comments, are really meant for the translator, and PO mode gives her the full control she needs.

The comment lines beginning with `#,` are special because they are not completely ignored by the programs as comments generally are. The comma separated list of *flag*s is used by the `msgfmt` program to give the user some better diagnostic messages. Currently there are two forms of flags defined:

`fuzzy`
: This flag can be generated by the `msgmerge` program or it can be inserted by the translator herself. It shows that the `msgstr` string might not be a correct translation (anymore). Only the translator can judge if the translation requires further modification, or is acceptable as is. Once satisfied with the translation, she then removes this *`fuzzy`* attribute. The `msgmerge` program inserts this when it combined the `msgid` and `msgstr` entries after fuzzy search only. See Section 6.3 [Fuzzy Entries], page 39.

`c-format`
`no-c-format`
: These flags should not be added by a human. Instead only the `xgettext` program adds them. In an automated PO file processing system as proposed here the user changes would be thrown away again as soon as the `xgettext` program generates a new template file.

  In case the *`c-format`* flag is given for a string the `msgfmt` does some more tests to check to validity of the translation. See Section 8.1 [msgfmt Invocation], page 70.

A different kind of entries is used for translations which involve plural forms.

white-space
`#` *translator-comments*
`#.` *automatic-comments*
`#:` *reference...*
`#,` *flag...*
`msgid` *untranslated-string-singular*
`msgid_plural` *untranslated-string-plural*
`msgstr[0]` *translated-string-case-0*
`...`
`msgstr[N]` *translated-string-case-n*

It happens that some lines, usually whitespace or comments, follow the very last entry of a PO file. Such lines are not part of any entry, and PO mode is unable to take action on those lines. By using the PO mode function *`M-x po-normalize`*, the translator may get rid of those spurious lines. See Section 2.5 [Normalizing], page 16.

The remainder of this section may be safely skipped by those using PO mode, yet it may be interesting for everybody to have a better idea of the precise format of a PO file. On the other hand, those not having Emacs handy should carefully continue reading on.

Each of *untranslated-string* and *translated-string* respects the C syntax for a character string, including the surrounding quotes and embedded backslashed escape sequences. When the time comes to write multi-line strings, one should not use escaped newlines. Instead, a closing quote should follow the last character on the line to be continued, and an opening quote should resume the string at the beginning of the following PO file line. For example:

```
msgid ""
"Here is an example of how one might continue a very long string\n"
"for the common case the string represents multi-line output.\n"
```

In this example, the empty string is used on the first line, to allow better alignment of the *`H`* from the word '`Here`' over the *`f`* from the word '`for`'. In this example, the `msgid` keyword is followed by three strings, which are meant to be concatenated. Concatenating the empty string does not change the resulting overall string, but it is a way for us to comply with the necessity of `msgid` to be followed by a string on the same line, while keeping the multi-line presentation left-justified, as we find this to be a cleaner disposition. The empty string could have been omitted, but only if the string starting with '`Here`' was promoted on the first line, right after `msgid`.[1] It was not really necessary either to switch between the two last quoted strings immediately after the newline '`\n`', the switch could have occurred after *any* other character, we just did it this way because it is neater.

One should carefully distinguish between end of lines marked as '`\n`' *inside* quotes, which are part of the represented string, and end of lines in the PO file itself, outside string quotes, which have no incidence on the represented string.

Outside strings, white lines and comments may be used freely. Comments start at the beginning of a line with '`#`' and extend until the end of the PO file line. Comments written

[1] This limitation is not imposed by GNU `gettext`, but is for compatibility with the `msgfmt` implementation on Solaris.

by translators should have the initial '#' immediately followed by some white space. If the '#' is not immediately followed by white space, this comment is most likely generated and managed by specialized GNU tools, and might disappear or be replaced unexpectedly when the PO file is given to `msgmerge`.

2.3 Main PO mode Commands

After setting up Emacs with something similar to the lines in Section 2.1 [Installation], page 10, PO mode is activated for a window when Emacs finds a PO file in that window. This puts the window read-only and establishes a po-mode-map, which is a genuine Emacs mode, in a way that is not derived from text mode in any way. Functions found on `po-mode-hook`, if any, will be executed.

When PO mode is active in a window, the letters '`PO`' appear in the mode line for that window. The mode line also displays how many entries of each kind are held in the PO file. For example, the string '`132t+3f+10u+2o`' would tell the translator that the PO mode contains 132 translated entries (see Section 6.2 [Translated Entries], page 38, 3 fuzzy entries (see Section 6.3 [Fuzzy Entries], page 39), 10 untranslated entries (see Section 6.4 [Untranslated Entries], page 40) and 2 obsolete entries (see Section 6.5 [Obsolete Entries], page 40). Zero-coefficients items are not shown. So, in this example, if the fuzzy entries were unfuzzied, the untranslated entries were translated and the obsolete entries were deleted, the mode line would merely display '`145t`' for the counters.

The main PO commands are those which do not fit into the other categories of subsequent sections. These allow for quitting PO mode or for managing windows in special ways.

`_` Undo last modification to the PO file (`po-undo`).

`Q` Quit processing and save the PO file (`po-quit`).

`q` Quit processing, possibly after confirmation (`po-confirm-and-quit`).

`0` Temporary leave the PO file window (`po-other-window`).

`?`
`h` Show help about PO mode (`po-help`).

`=` Give some PO file statistics (`po-statistics`).

`V` Batch validate the format of the whole PO file (`po-validate`).

The command `_` (`po-undo`) interfaces to the Emacs *undo* facility. See section "Undoing Changes" in *The Emacs Editor*. Each time *`U`* is typed, modifications which the translator did to the PO file are undone a little more. For the purpose of undoing, each PO mode command is atomic. This is especially true for the ⟨RET⟩ command: the whole edition made by using a single use of this command is undone at once, even if the edition itself implied several actions. However, while in the editing window, one can undo the edition work quite parsimoniously.

The commands *`Q`* (`po-quit`) and `q` (`po-confirm-and-quit`) are used when the translator is done with the PO file. The former is a bit less verbose than the latter. If the file has been modified, it is saved to disk first. In both cases, and prior to all this, the commands check if any untranslated messages remain in the PO file and, if so, the translator is asked

if she really wants to leave off working with this PO file. This is the preferred way of getting rid of an Emacs PO file buffer. Merely killing it through the usual command *C-x k* (`kill-buffer`) is not the tidiest way to proceed.

The command *0* (`po-other-window`) is another, softer way, to leave PO mode, temporarily. It just moves the cursor to some other Emacs window, and pops one if necessary. For example, if the translator just got PO mode to show some source context in some other, she might discover some apparent bug in the program source that needs correction. This command allows the translator to change sex, become a programmer, and have the cursor right into the window containing the program she (or rather *he*) wants to modify. By later getting the cursor back in the PO file window, or by asking Emacs to edit this file once again, PO mode is then recovered.

The command *h* (`po-help`) displays a summary of all available PO mode commands. The translator should then type any character to resume normal PO mode operations. The command *?* has the same effect as *h*.

The command *=* (`po-statistics`) computes the total number of entries in the PO file, the ordinal of the current entry (counted from 1), the number of untranslated entries, the number of obsolete entries, and displays all these numbers.

The command *V* (`po-validate`) launches `msgfmt` in checking and verbose mode over the current PO file. This command first offers to save the current PO file on disk. The `msgfmt` tool, from GNU `gettext`, has the purpose of creating a MO file out of a PO file, and PO mode uses the features of this program for checking the overall format of a PO file, as well as all individual entries.

The program `msgfmt` runs asynchronously with Emacs, so the translator regains control immediately while her PO file is being studied. Error output is collected in the Emacs '`*compilation*`' buffer, displayed in another window. The regular Emacs command *C-x`* (`next-error`), as well as other usual compile commands, allow the translator to reposition quickly to the offending parts of the PO file. Once the cursor is on the line in error, the translator may decide on any PO mode action which would help correcting the error.

2.4 Entry Positioning

The cursor in a PO file window is almost always part of an entry. The only exceptions are the special case when the cursor is after the last entry in the file, or when the PO file is empty. The entry where the cursor is found to be is said to be the current entry. Many PO mode commands operate on the current entry, so moving the cursor does more than allowing the translator to browse the PO file, this also selects on which entry commands operate.

Some PO mode commands alter the position of the cursor in a specialized way. A few of those special purpose positioning are described here, the others are described in following sections (for a complete list try *C-h m*):

. Redisplay the current entry (`po-current-entry`).

n Select the entry after the current one (`po-next-entry`).

p Select the entry before the current one (`po-previous-entry`).

`<` Select the first entry in the PO file (`po-first-entry`).

`>` Select the last entry in the PO file (`po-last-entry`).

m Record the location of the current entry for later use (`po-push-location`).

r Return to a previously saved entry location (`po-pop-location`).

x Exchange the current entry location with the previously saved one (`po-exchange-location`).

Any Emacs command able to reposition the cursor may be used to select the current entry in PO mode, including commands which move by characters, lines, paragraphs, screens or pages, and search commands. However, there is a kind of standard way to display the current entry in PO mode, which usual Emacs commands moving the cursor do not especially try to enforce. The command `.` (`po-current-entry`) has the sole purpose of redisplaying the current entry properly, after the current entry has been changed by means external to PO mode, or the Emacs screen otherwise altered.

It is yet to be decided if PO mode helps the translator, or otherwise irritates her, by forcing a rigid window disposition while she is doing her work. We originally had quite precise ideas about how windows should behave, but on the other hand, anyone used to Emacs is often happy to keep full control. Maybe a fixed window disposition might be offered as a PO mode option that the translator might activate or deactivate at will, so it could be offered on an experimental basis. If nobody feels a real need for using it, or a compulsion for writing it, we should drop this whole idea. The incentive for doing it should come from translators rather than programmers, as opinions from an experienced translator are surely more worth to me than opinions from programmers *thinking* about how *others* should do translation.

The commands *n* (`po-next-entry`) and *p* (`po-previous-entry`) move the cursor the entry following, or preceding, the current one. If *n* is given while the cursor is on the last entry of the PO file, or if *p* is given while the cursor is on the first entry, no move is done.

The commands `<` (`po-first-entry`) and `>` (`po-last-entry`) move the cursor to the first entry, or last entry, of the PO file. When the cursor is located past the last entry in a PO file, most PO mode commands will return an error saying '`After last entry`'. Moreover, the commands `<` and `>` have the special property of being able to work even when the cursor is not into some PO file entry, and one may use them for nicely correcting this situation. But even these commands will fail on a truly empty PO file. There are development plans for the PO mode for it to interactively fill an empty PO file from sources. See Section 3.4 [Marking], page 22.

The translator may decide, before working at the translation of a particular entry, that she needs to browse the remainder of the PO file, maybe for finding the terminology or phraseology used in related entries. She can of course use the standard Emacs idioms for saving the current cursor location in some register, and use that register for getting back, or else, use the location ring.

PO mode offers another approach, by which cursor locations may be saved onto a special stack. The command *m* (`po-push-location`) merely adds the location of current entry to the stack, pushing the already saved locations under the new one. The command *r* (`po-pop-location`) consumes the top stack element and repositions the cursor to the entry

associated with that top element. This position is then lost, for the next *r* will move the cursor to the previously saved location, and so on until no locations remain on the stack.

If the translator wants the position to be kept on the location stack, maybe for taking a look at the entry associated with the top element, then go elsewhere with the intent of getting back later, she ought to use *m* immediately after *r*.

The command *x* (`po-exchange-location`) simultaneously repositions the cursor to the entry associated with the top element of the stack of saved locations, and replaces that top element with the location of the current entry before the move. Consequently, repeating the *x* command toggles alternatively between two entries. For achieving this, the translator will position the cursor on the first entry, use *m*, then position to the second entry, and merely use *x* for making the switch.

2.5 Normalizing Strings in Entries

There are many different ways for encoding a particular string into a PO file entry, because there are so many different ways to split and quote multi-line strings, and even, to represent special characters by backslashed escaped sequences. Some features of PO mode rely on the ability for PO mode to scan an already existing PO file for a particular string encoded into the `msgid` field of some entry. Even if PO mode has internally all the built-in machinery for implementing this recognition easily, doing it fast is technically difficult. To facilitate a solution to this efficiency problem, we decided on a canonical representation for strings.

A conventional representation of strings in a PO file is currently under discussion, and PO mode experiments with a canonical representation. Having both `xgettext` and PO mode converging towards a uniform way of representing equivalent strings would be useful, as the internal normalization needed by PO mode could be automatically satisfied when using `xgettext` from GNU `gettext`. An explicit PO mode normalization should then be only necessary for PO files imported from elsewhere, or for when the convention itself evolves.

So, for achieving normalization of at least the strings of a given PO file needing a canonical representation, the following PO mode command is available:

M-x po-normalize
: Tidy the whole PO file by making entries more uniform.

The special command *M-x po-normalize*, which has no associated keys, revises all entries, ensuring that strings of both original and translated entries use uniform internal quoting in the PO file. It also removes any crumb after the last entry. This command may be useful for PO files freshly imported from elsewhere, or if we ever improve on the canonical quoting format we use. This canonical format is not only meant for getting cleaner PO files, but also for greatly speeding up `msgid` string lookup for some other PO mode commands.

M-x po-normalize presently makes three passes over the entries. The first implements heuristics for converting PO files for GNU `gettext` 0.6 and earlier, in which `msgid` and `msgstr` fields were using K&R style C string syntax for multi-line strings. These heuristics may fail for comments not related to obsolete entries and ending with a backslash; they also depend on subsequent passes for finalizing the proper commenting of continued lines for

obsolete entries. This first pass might disappear once all oldish PO files would have been adjusted. The second and third pass normalize all `msgid` and `msgstr` strings respectively. They also clean out those trailing backslashes used by XView's `msgfmt` for continued lines.

Having such an explicit normalizing command allows for importing PO files from other sources, but also eases the evolution of the current convention, evolution driven mostly by aesthetic concerns, as of now. It is easy to make suggested adjustments at a later time, as the normalizing command and eventually, other GNU `gettext` tools should greatly automate conformance. A description of the canonical string format is given below, for the particular benefit of those not having Emacs handy, and who would nevertheless want to handcraft their PO files in nice ways.

Right now, in PO mode, strings are single line or multi-line. A string goes multi-line if and only if it has *embedded* newlines, that is, if it matches '`[^\n]\n+[^\n]`'. So, we would have:

```
msgstr "\n\nHello, world!\n\n\n"
```

but, replacing the space by a newline, this becomes:

```
msgstr ""
"\n"
"\n"
"Hello,\n"
"world!\n"
"\n"
"\n"
```

We are deliberately using a caricatural example, here, to make the point clearer. Usually, multi-lines are not that bad looking. It is probable that we will implement the following suggestion. We might lump together all initial newlines into the empty string, and also all newlines introducing empty lines (that is, for $n > 1$, the n-1'th last newlines would go together on a separate string), so making the previous example appear:

```
msgstr "\n\n"
"Hello,\n"
"world!\n"
"\n\n"
```

There are a few yet undecided little points about string normalization, to be documented in this manual, once these questions settle.

3 Preparing Program Sources

For the programmer, changes to the C source code fall into three categories. First, you have to make the localization functions known to all modules needing message translation. Second, you should properly trigger the operation of GNU `gettext` when the program initializes, usually from the `main` function. Last, you should identify and especially mark all constant strings in your program needing translation.

Presuming that your set of programs, or package, has been adjusted so all needed GNU `gettext` files are available, and your '`Makefile`' files are adjusted (see Chapter 12 [Maintainers], page 102), each C module having translated C strings should contain the line:

```
#include <libintl.h>
```

The remaining changes to your C sources are discussed in the further sections of this chapter.

3.1 Triggering `gettext` Operations

The initialization of locale data should be done with more or less the same code in every program, as demonstrated below:

```
int
main (argc, argv)
     int argc;
     char argv;
{
  ...
  setlocale (LC_ALL, "");
  bindtextdomain (PACKAGE, LOCALEDIR);
  textdomain (PACKAGE);
  ...
}
```

PACKAGE and *LOCALEDIR* should be provided either by '`config.h`' or by the Makefile. For now consult the `gettext` or `hello` sources for more information.

The use of `LC_ALL` might not be appropriate for you. `LC_ALL` includes all locale categories and especially `LC_CTYPE`. This later category is responsible for determining character classes with the `isalnum` etc. functions from '`ctype.h`' which could especially for programs, which process some kind of input language, be wrong. For example this would mean that a source code using the ç (c-cedilla character) is runnable in France but not in the U.S.

Some systems also have problems with parsing numbers using the `scanf` functions if an other but the `LC_ALL` locale is used. The standards say that additional formats but the one known in the `"C"` locale might be recognized. But some systems seem to reject numbers in the `"C"` locale format. In some situation, it might also be a problem with the notation itself which makes it impossible to recognize whether the number is in the `"C"` locale or the local format. This can happen if thousands separator characters are used. Some locales define this character accordfing to the national conventions to `'.'` which is the same character used in the `"C"` locale to denote the decimal point.

So it is sometimes necessary to replace the `LC_ALL` line in the code above by a sequence of `setlocale` lines

```
{
  ...
  setlocale (LC_CTYPE, "");
  setlocale (LC_MESSAGES, "");
  ...
}
```

On all POSIX conformant systems the locale categories `LC_CTYPE`, `LC_COLLATE`, `LC_MONETARY`, `LC_NUMERIC`, and `LC_TIME` are available. On some modern systems there is also a locale `LC_MESSAGES` which is called on some old, XPG2 compliant systems `LC_RESPONSES`.

Note that changing the `LC_CTYPE` also affects the functions declared in the `<ctype.h>` standard header. If this is not desirable in your application (for example in a compiler's parser), you can use a set of substitute functions which hardwire the C locale, such as found in the `<c-ctype.h>` and `<c-ctype.c>` files in the gettext source distribution.

It is also possible to switch the locale forth and back between the environment dependent locale and the C locale, but this approach is normally avoided because a `setlocale` call is expensive, because it is tedious to determine the places where a locale switch is needed in a large program's source, and because switching a locale is not multithread-safe.

3.2 Preparing Translatable Strings

Before strings can be marked for translations, they sometimes need to be adjusted. Usually preparing a string for translation is done right before marking it, during the marking phase which is described in the next sections. What you have to keep in mind while doing that is the following.

- Decent English style.
- Entire sentences.
- Split at paragraphs.
- Use format strings instead of string concatenation.

Let's look at some examples of these guidelines.

Translatable strings should be in good English style. If slang language with abbreviations and shortcuts is used, often translators will not understand the message and will produce very inappropriate translations.

```
"%s: is parameter\n"
```

This is nearly untranslatable: Is the displayed item *a* parameter or *the* parameter?

```
"No match"
```

The ambiguity in this message makes it ununderstandable: Is the program attempting to set something on fire? Does it mean "The given object does not match the template"? Does it mean "The template does not fit for any of the objects"?

In both cases, adding more words to the message will help both the translator and the English speaking user.

Translatable strings should be entire sentences. It is often not possible to translate single verbs or adjectives in a substitutable way.

```
printf ("File %s is %s protected", filename, rw ? "write" : "read");
```

Most translators will not look at the source and will thus only see the string "File %s is %s protected", which is unintelligible. Change this to

```
printf (rw ? "File %s is write protected" : "File %s is read protected",
        filename);
```

This way the translator will not only understand the message, she will also be able to find the appropriate grammatical construction. The French translator for example translates "write protected" like "protected against writing".

Often sentences don't fit into a single line. If a sentence is output using two subsequent `printf` statements, like this

```
printf ("Locale charset \"%s\" is different from\n", lcharset);
printf ("input file charset \"%s\".\n", fcharset);
```

the translator would have to translate two half sentences, but nothing in the POT file would tell her that the two half sentences belong together. It is necessary to merge the two `printf` statements so that the translator can handle the entire sentence at once and decide at which place to insert a line break in the translation (if at all):

```
printf ("Locale charset \"%s\" is different from\n\
input file charset \"%s\".\n", lcharset, fcharset);
```

You may now ask: how about two or more adjacent sentences? Like in this case:

```
puts ("Apollo 13 scenario: Stack overflow handling failed.");
puts ("On the next stack overflow we will crash!!!");
```

Should these two statements merged into a single one? I would recommend to merge them if the two sentences are related to each other, because then it makes it easier for the translator to understand and translate both. On the other hand, if one of the two messages is a stereotypic one, occurring in other places as well, you will do a favour to the translator by not merging the two. (Identical messages occurring in several places are combined by xgettext, so the translator has to handle them once only.)

Translatable strings should be limited to one paragraph; don't let a single message be longer than ten lines. The reason is that when the translatable string changes, the translator is faced with the task of updating the entire translated string. Maybe only a single word will have changed in the English string, but the translator doesn't see that (with the current translation tools), therefore she has to proofread the entire message.

Many GNU programs have a '`--help`' output that extends over several screen pages. It is a courtesy towards the translators to split such a message into several ones of five to ten lines each. While doing that, you can also attempt to split the documented options into groups, such as the input options, the output options, and the informative output options. This will help every user to find the option he is looking for.

Hardcoded string concatenation is sometimes used to construct English strings:

```
strcpy (s, "Replace ");
strcat (s, object1);
strcat (s, " with ");
strcat (s, object2);
strcat (s, "?");
```

In order to present to the translator only entire sentences, and also because in some languages the translator might want to swap the order of `object1` and `object2`, it is necessary to change this to use a format string:

```
sprintf (s, "Replace %s with %s?", object1, object2);
```

A similar case is compile time concatenation of strings. The ISO C 99 include file `<inttypes.h>` contains a macro `PRId64` that can be used as a formatting directive for outputting an '`int64_t`' integer through `printf`. It expands to a constant string, usually "d" or "ld" or "lld" or something like this, depending on the platform. Assume you have code like

```
printf ("The amount is %0" PRId64 "\n"), number);
```

After marking, this cannot become

```
printf (gettext ("The amount is %0") PRId64 "\n"), number);
```

because it would simply be invalid C syntax. It cannot become

```
printf (gettext ("The amount is %0" PRId64 "\n")), number);
```

because the value of `PRId64` is not known to `xgettext`, and even if were, there would be three or more possibilities, and the translator would have to translate three or more strings that differ in a single letter.

The solution for this problem is to change the code like this:

```
char buf1[100];
sprintf (buf1, "%0" PRId64, number);
printf (gettext ("The amount is %s\n"), buf1);
```

This means, you put the platform dependent code in one statement, and the internationalization code in a different statement. Note that a buffer length of 100 is safe, because all available hardware integer types are limited to 128 bits, and to print a 128 bit integer one needs at most 54 characters, regardless whether in decimal, octal or hexadecimal.

All this applies to other programming languages as well. For example, in Java, string contenation is very frequently used, because it is a compiler built-in operator. Like in C, in Java, you would change

```
System.out.println("Replace "+object1+" with "+object2+"?");
```

into a statement involving a format string:

```
System.out.println(
    MessageFormat.format("Replace {0} with {1}?",
                         new Object[] { object1, object2 }));
```

3.3 How Marks Appear in Sources

All strings requiring translation should be marked in the C sources. Marking is done in such a way that each translatable string appears to be the sole argument of some function or preprocessor macro. There are only a few such possible functions or macros meant for translation, and their names are said to be marking keywords. The marking is attached to strings themselves, rather than to what we do with them. This approach has more uses. A blatant example is an error message produced by formatting. The format string needs translation, as well as some strings inserted through some '`%s`' specification in the format,

while the result from `sprintf` may have so many different instances that it is impractical to list them all in some '`error_string_out()`' routine, say.

This marking operation has two goals. The first goal of marking is for triggering the retrieval of the translation, at run time. The keyword are possibly resolved into a routine able to dynamically return the proper translation, as far as possible or wanted, for the argument string. Most localizable strings are found in executable positions, that is, attached to variables or given as parameters to functions. But this is not universal usage, and some translatable strings appear in structured initializations. See Section 3.6 [Special cases], page 26.

The second goal of the marking operation is to help `xgettext` at properly extracting all translatable strings when it scans a set of program sources and produces PO file templates.

The canonical keyword for marking translatable strings is '`gettext`', it gave its name to the whole GNU `gettext` package. For packages making only light use of the '`gettext`' keyword, macro or function, it is easily used *as is*. However, for packages using the `gettext` interface more heavily, it is usually more convenient to give the main keyword a shorter, less obtrusive name. Indeed, the keyword might appear on a lot of strings all over the package, and programmers usually do not want nor need their program sources to remind them forcefully, all the time, that they are internationalized. Further, a long keyword has the disadvantage of using more horizontal space, forcing more indentation work on sources for those trying to keep them within 79 or 80 columns.

Many packages use '`_`' (a simple underline) as a keyword, and write '`_("Translatable string")`' instead of '`gettext ("Translatable string")`'. Further, the coding rule, from GNU standards, wanting that there is a space between the keyword and the opening parenthesis is relaxed, in practice, for this particular usage. So, the textual overhead per translatable string is reduced to only three characters: the underline and the two parentheses. However, even if GNU `gettext` uses this convention internally, it does not offer it officially. The real, genuine keyword is truly '`gettext`' indeed. It is fairly easy for those wanting to use '`_`' instead of '`gettext`' to declare:

```
#include <libintl.h>
#define _(String) gettext (String)
```

instead of merely using '`#include <libintl.h>`'.

Later on, the maintenance is relatively easy. If, as a programmer, you add or modify a string, you will have to ask yourself if the new or altered string requires translation, and include it within '`_()`' if you think it should be translated. '`"%s: %d"`' is an example of string *not* requiring translation!

3.4 Marking Translatable Strings

In PO mode, one set of features is meant more for the programmer than for the translator, and allows him to interactively mark which strings, in a set of program sources, are translatable, and which are not. Even if it is a fairly easy job for a programmer to find and mark such strings by other means, using any editor of his choice, PO mode makes this work more comfortable. Further, this gives translators who feel a little like programmers, or programmers who feel a little like translators, a tool letting them work at marking trans-

latable strings in the program sources, while simultaneously producing a set of translation in some language, for the package being internationalized.

The set of program sources, targetted by the PO mode commands describe here, should have an Emacs tags table constructed for your project, prior to using these PO file commands. This is easy to do. In any shell window, change the directory to the root of your project, then execute a command resembling:

```
etags src/*.[hc] lib/*.[hc]
```

presuming here you want to process all '`.h`' and '`.c`' files from the '`src/`' and '`lib/`' directories. This command will explore all said files and create a '`TAGS`' file in your root directory, somewhat summarizing the contents using a special file format Emacs can understand.

For packages following the GNU coding standards, there is a make goal `tags` or `TAGS` which constructs the tag files in all directories and for all files containing source code.

Once your '`TAGS`' file is ready, the following commands assist the programmer at marking translatable strings in his set of sources. But these commands are necessarily driven from within a PO file window, and it is likely that you do not even have such a PO file yet. This is not a problem at all, as you may safely open a new, empty PO file, mainly for using these commands. This empty PO file will slowly fill in while you mark strings as translatable in your program sources.

`,` Search through program sources for a string which looks like a candidate for translation (`po-tags-search`).

M-`,` Mark the last string found with '`_()`' (`po-mark-translatable`).

M-`.` Mark the last string found with a keyword taken from a set of possible keywords. This command with a prefix allows some management of these keywords (`po-select-mark-and-mark`).

The `,` (`po-tags-search`) command searches for the next occurrence of a string which looks like a possible candidate for translation, and displays the program source in another Emacs window, positioned in such a way that the string is near the top of this other window. If the string is too big to fit whole in this window, it is positioned so only its end is shown. In any case, the cursor is left in the PO file window. If the shown string would be better presented differently in different native languages, you may mark it using *M-*`,` or *M-*`.`. Otherwise, you might rather ignore it and skip to the next string by merely repeating the `,` command.

A string is a good candidate for translation if it contains a sequence of three or more letters. A string containing at most two letters in a row will be considered as a candidate if it has more letters than non-letters. The command disregards strings containing no letters, or isolated letters only. It also disregards strings within comments, or strings already marked with some keyword PO mode knows (see below).

If you have never told Emacs about some '`TAGS`' file to use, the command will request that you specify one from the minibuffer, the first time you use the command. You may later change your '`TAGS`' file by using the regular Emacs command *M-x visit-tags-table*, which will ask you to name the precise '`TAGS`' file you want to use. See section "Tag Tables" in *The Emacs Editor*.

Each time you use the `,` command, the search resumes from where it was left by the previous search, and goes through all program sources, obeying the '`TAGS`' file, until all sources have been processed. However, by giving a prefix argument to the command (*C-u* `,`), you may request that the search be restarted all over again from the first program source; but in this case, strings that you recently marked as translatable will be automatically skipped.

Using this `,` command does not prevent using of other regular Emacs tags commands. For example, regular `tags-search` or `tags-query-replace` commands may be used without disrupting the independent `,` search sequence. However, as implemented, the *initial* `,` command (or the `,` command is used with a prefix) might also reinitialize the regular Emacs tags searching to the first tags file, this reinitialization might be considered spurious.

The *M-*`,` (`po-mark-translatable`) command will mark the recently found string with the '`_`' keyword. The *M-*`.` (`po-select-mark-and-mark`) command will request that you type one keyword from the minibuffer and use that keyword for marking the string. Both commands will automatically create a new PO file untranslated entry for the string being marked, and make it the current entry (making it easy for you to immediately proceed to its translation, if you feel like doing it right away). It is possible that the modifications made to the program source by *M-*`,` or *M-*`.` render some source line longer than 80 columns, forcing you to break and re-indent this line differently. You may use the *O* command from PO mode, or any other window changing command from Emacs, to break out into the program source window, and do any needed adjustments. You will have to use some regular Emacs command to return the cursor to the PO file window, if you want command `,` for the next string, say.

The *M-*`.` command has a few built-in speedups, so you do not have to explicitly type all keywords all the time. The first such speedup is that you are presented with a *preferred* keyword, which you may accept by merely typing ⟨RET⟩ at the prompt. The second speedup is that you may type any non-ambiguous prefix of the keyword you really mean, and the command will complete it automatically for you. This also means that PO mode has to *know* all your possible keywords, and that it will not accept mistyped keywords.

If you reply `?` to the keyword request, the command gives a list of all known keywords, from which you may choose. When the command is prefixed by an argument (*C-u M-*`.`), it inhibits updating any program source or PO file buffer, and does some simple keyword management instead. In this case, the command asks for a keyword, written in full, which becomes a new allowed keyword for later *M-*`.` commands. Moreover, this new keyword automatically becomes the *preferred* keyword for later commands. By typing an already known keyword in response to *C-u M-*`.`, one merely changes the *preferred* keyword and does nothing more.

All keywords known for *M-*`.` are recognized by the `,` command when scanning for strings, and strings already marked by any of those known keywords are automatically skipped. If many PO files are opened simultaneously, each one has its own independent set of known keywords. There is no provision in PO mode, currently, for deleting a known keyword, you have to quit the file (maybe using *q*) and reopen it afresh. When a PO file is newly brought up in an Emacs window, only '`gettext`' and '`_`' are known as keywords, and '`gettext`' is preferred for the *M-*`.` command. In fact, this is not useful to prefer '`_`', as this one is already built in the *M-*`,` command.

3.5 Special Comments preceding Keywords

In C programs strings are often used within calls of functions from the `printf` family. The special thing about these format strings is that they can contain format specifiers introduced with `%`. Assume we have the code

```
printf (gettext ("String `%s' has %d characters\n"), s, strlen (s));
```

A possible German translation for the above string might be:

```
"%d Zeichen lang ist die Zeichenkette `%s'"
```

A C programmer, even if he cannot speak German, will recognize that there is something wrong here. The order of the two format specifiers is changed but of course the arguments in the `printf` don't have. This will most probably lead to problems because now the length of the string is regarded as the address.

To prevent errors at runtime caused by translations the `msgfmt` tool can check statically whether the arguments in the original and the translation string match in type and number. If this is not the case and the '`-c`' option has been passed to `msgfmt`, `msgfmt` will give an error and refuse to produce a MO file. Thus consequent use of '`msgfmt -c`' will catch the error, so that it cannot cause cause problems at runtime.

If the word order in the above German translation would be correct one would have to write

```
"%2$d Zeichen lang ist die Zeichenkette `%1$s'"
```

The routines in `msgfmt` know about this special notation.

Because not all strings in a program must be format strings it is not useful for `msgfmt` to test all the strings in the '`.po`' file. This might cause problems because the string might contain what looks like a format specifier, but the string is not used in `printf`.

Therefore the `xgettext` adds a special tag to those messages it thinks might be a format string. There is no absolute rule for this, only a heuristic. In the '`.po`' file the entry is marked using the `c-format` flag in the `#,` comment line (see Section 2.2 [PO Files], page 10).

The careful reader now might say that this again can cause problems. The heuristic might guess it wrong. This is true and therefore `xgettext` knows about special kind of comment which lets the programmer take over the decision. If in the same line or the immediately preceding line of the `gettext` keyword the `xgettext` program find a comment containing the words *`xgettext:c-format`* it will mark the string in any case with the *`c-format`* flag. This kind of comment should be used when `xgettext` does not recognize the string as a format string but is really is one and it should be tested. Please note that when the comment is in the same line of the `gettext` keyword, it must be before the string to be translated.

This situation happens quite often. The `printf` function is often called with strings which do not contain a format specifier. Of course one would normally use `fputs` but it does happen. In this case `xgettext` does not recognize this as a format string but what happens if the translation introduces a valid format specifier? The `printf` function will try to access one of the parameter but none exists because the original code does not refer to any parameter.

`xgettext` of course could make a wrong decision the other way round, i.e. a string marked as a format string actually is not a format string. In this case the `msgfmt` might give too many warnings and would prevent translating the '`.po`' file. The method to prevent

this wrong decision is similar to the one used above, only the comment to use must contain the string *xgettext:no-c-format*.

If a string is marked with *c-format* and this is not correct the user can find out who is responsible for the decision. See Section 4.1 [xgettext Invocation], page 28 to see how the *--debug* option can be used for solving this problem.

3.6 Special Cases of Translatable Strings

The attentive reader might now point out that it is not always possible to mark translatable string with `gettext` or something like this. Consider the following case:

```
{
  static const char *messages[] = {
    "some very meaningful message",
    "and another one"
  };
  const char *string;
  ...
  string
    = index > 1 ? "a default message" : messages[index];

  fputs (string);
  ...
}
```

While it is no problem to mark the string `"a default message"` it is not possible to mark the string initializers for `messages`. What is to be done? We have to fulfill two tasks. First we have to mark the strings so that the `xgettext` program (see Section 4.1 [xgettext Invocation], page 28) can find them, and second we have to translate the string at runtime before printing them.

The first task can be fulfilled by creating a new keyword, which names a no-op. For the second we have to mark all access points to a string from the array. So one solution can look like this:

```
#define gettext_noop(String) String

{
  static const char *messages[] = {
    gettext_noop ("some very meaningful message"),
    gettext_noop ("and another one")
  };
  const char *string;
  ...
  string
    = index > 1 ? gettext ("a default message") : gettext (messages[index]);

  fputs (string);
  ...
}
```

Please convince yourself that the string which is written by `fputs` is translated in any case. How to get `xgettext` know the additional keyword `gettext_noop` is explained in Section 4.1 [xgettext Invocation], page 28.

The above is of course not the only solution. You could also come along with the following one:

```
#define gettext_noop(String) String

{
  static const char *messages[] = {
    gettext_noop ("some very meaningful message",
    gettext_noop ("and another one")
  };
  const char *string;
  ...
  string
    = index > 1 ? gettext_noop ("a default message") : messages[index];

  fputs (gettext (string));
  ...
}
```

But this has a drawback. The programmer has to take care that he uses `gettext_noop` for the string `"a default message"`. A use of `gettext` could have in rare cases unpredictable results.

One advantage is that you need not make control flow analysis to make sure the output is really translated in any case. But this analysis is generally not very difficult. If it should be in any situation you can use this second method in this situation.

4 Making the PO Template File

After preparing the sources, the programmer creates a PO template file. This section explains how to use `xgettext` for this purpose.

4.1 Invoking the `xgettext` Program

`xgettext [`*option*`] [`*inputfile*`] ...`

The `xgettext` program extracts translatable strings from given input files.

4.1.1 Input file location

'*inputfile* ...'
: Input files.

'`-f` *file*'
'`--files-from=`*file*'
: Read the names of the input files from *file* instead of getting them from the command line.

'`-D` *directory*'
'`--directory=`*directory*'
: Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If *inputfile* is '`-`', standard input is read.

4.1.2 Output file location

'`-d` *name*'
'`--default-domain=`*name*'
: Use '*name*`.po`' for output (instead of '`messages.po`').

'`-o` *file*'
'`--output=`*file*'
: Write output to specified file (instead of '*name*`.po`' or '`messages.po`').

'`-p` *dir*'
'`--output-dir=`*dir*'
: Output files will be placed in directory *dir*.

If the output *file* is '`-`' or '`/dev/stdout`', the output is written to standard output.

4.1.3 Choice of input file language

'`-L` *name*'
'`--language=`*name*'
: Specifies the language of the input files. The supported languages are `C`, `C++`, `ObjectiveC`, `PO`, `Python`, `Lisp`, `EmacsLisp`, `librep`, `Java`, `awk`, `YCP`, `Tcl`, `RST`, `Glade`.

'`-C`'
'`--c++`' This is a shorthand for `--language=C++`.

By default the language is guessed depending on the input file name extension.

4.1.4 Operation mode

'`-j`'
'`--join-existing`'
Join messages with existing file.

'`-x` *file*'
'`--exclude-file=`*file*'
Entries from *file* are not extracted. *file* should be a PO or POT file.

'`-c [`*tag*`]`'
'`--add-comments[=`*tag*`]`'
Place comment block with *tag* (or those preceding keyword lines) in output file.

4.1.5 Language=C/C++ specific options

'`-a`'
'`--extract-all`'
Extract all strings.

'`-k` *keywordspec*'
'`--keyword[=`*keywordspec*`]`'
Additional keyword to be looked for (without *keywordspec* means not to use default keywords).

If *keywordspec* is a C identifer *id*, `xgettext` looks for strings in the first argument of each call to the function or macro *id*. If *keywordspec* is of the form '*id*`:`*argnum*', `xgettext` looks for strings in the *argnum*th argument of the call. If *keywordspec* is of the form '*id*`:`*argnum1*`,`*argnum2*', `xgettext` looks for strings in the *argnum1*st argument and in the *argnum2*nd argument of the call, and treats them as singular/plural variants for a message with plural handling.

The default keyword specifications, which are always looked for if not explicitly disabled, are `gettext`, `dgettext:2`, `dcgettext:2`, `ngettext:1,2`, `dngettext:2,3`, `dcngettext:2,3`, and `gettext_noop`.

'`-T`'
'`--trigraphs`'
Understand ANSI C trigraphs for input.

'`--debug`' Use the flags *`c-format`* and *`possible-c-format`* to show who was responsible for marking a message as a format string. The latter form is used if the `xgettext` program decided, the format form is used if the programmer prescribed it.

By default only the *`c-format`* form is used. The translator should not have to care about these details.

This implementation of `xgettext` is able to process a few awkward cases, like strings in preprocessor macros, ANSI concatenation of adjacent strings, and escaped end of lines for continued strings.

4.1.6 Output details

'`--force-po`'
: Always write an output file even if no message is defined.

'`-i`'
'`--indent`'
: Write the .po file using indented style.

'`--no-location`'
: Do not write '`#:` *filename*`:`*line*' lines.

'`-n`'
'`--add-location`'
: Generate '`#:` *filename*`:`*line*' lines (default).

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`-s`'
'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

'`-F`'
'`--sort-by-file`'
: Sort output by file location.

'`--omit-header`'
: Don't write header with '`msgid ""`' entry.

  This is useful for testing purposes because it eliminates a source of variance for generated `.gmo` files. With `--omit-header`, two invocations of `xgettext` on the same files with the same options at different times are guaranteed to produce the same results.

'`--copyright-holder=`*string*'
: Set the copyright holder in the output. *string* should be the copyright holder of the surrounding package. (Note that the msgstr strings, extracted from the package's sources, belong to the copyright holder of the package.) Translators are expected to transfer or disclaim the copyright for their translations, so that package maintainers can distribute them without legal risk. If *string* is empty, the output files are marked as being in the public domain; in this case, the translators are expected to disclaim their copyright, again so that package maintainers can distribute them without legal risk.

  The default value for *string* is the Free Software Foundation, Inc., simply because `xgettext` was first used in the GNU project.

'`--foreign-user`'
: Omit FSF copyright in output. This option is equivalent to '`--copyright-holder=''`'. It can be useful for packages outside the GNU project that want their translations to be in the public domain.

'`-m` [*string*]'
'`--msgstr-prefix[=`*string*`]`'
: Use *string* (or "" if not specified) as prefix for msgstr entries.

'`-M` [*string*]'
'`--msgstr-suffix[=`*string*`]`'
: Use *string* (or "" if not specified) as suffix for msgstr entries.

4.1.7 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

5 Creating a New PO File

When starting a new translation, the translator creates a file called '*LANG*`.po`', as a copy of the '*package*`.pot`' template file with modifications in the initial comments (at the beginning of the file) and in the header entry (the first entry, near the beginning of the file).

The easiest way to do so is by use of the '`msginit`' program. For example:

```
$ cd PACKAGE-VERSION
$ cd po
$ msginit --verbose
```

The alternative way is to do the copy and modifications by hand. To do so, the translator copies '*package*`.pot`' to '*LANG*`.po`'. Then she modifies the initial comments and the header entry of this file.

5.1 Invoking the `msginit` Program

```
msginit [option]
```

The `msginit` program creates a new PO file, initializing the meta information with values from the user's environment.

5.1.1 Input file location

'`-i` *inputfile*'
'`--input=`*inputfile*'
: Input POT file.

If no *inputfile* is given, the current directory is searched for the POT file. If it is '`-`', standard input is read.

5.1.2 Output file location

'`-o` *file*'
'`--output-file=`*file*'
: Write output to specified PO file.

If no output file is given, it depends on the '`--locale`' option or the user's locale setting. If it is '`-`', the results are written to standard output.

5.1.3 Output details

'`-l` *ll_CC*'
'`--locale=`*ll_CC*'
: Set target locale. *ll* should be a language code, and *CC* should be a country code. The command '`locale -a`' can be used to output a list of all installed locales. The default is the user's locale setting.

'`--no-translator`'
: Declares that the PO file will not have a human translator and is instead automatically generated.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

5.1.4 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

5.2 Filling in the Header Entry

The initial comments "SOME DESCRIPTIVE TITLE", "YEAR" and "FIRST AUTHOR <EMAIL@ADDRESS>, YEAR" ought to be replaced by sensible information. This can be done in any text editor; if Emacs is used and it switched to PO mode automatically (because it has recognized the file's suffix), you can disable it by typing *`M-x fundamental-mode`*.

Modifying the header entry can already be done using PO mode: in Emacs, type *`M-x po-mode RET`* and then *`RET`* again to start editing the entry. You should fill in the following fields.

Project-Id-Version
: This is the name and version of the package.

POT-Creation-Date
: This has already been filled in by `xgettext`.

PO-Revision-Date
: You don't need to fill this in. It will be filled by the Emacs PO mode when you save the file.

Last-Translator
: Fill in your name and email address (without double quotes).

Language-Team
: Fill in the English name of the language, and the email address or homepage URL of the language team you are part of.

  Before starting a translation, it is a good idea to get in touch with your translation team, not only to make sure you don't do duplicated work, but also to coordinate difficult linguistic issues.

In the Free Translation Project, each translation team has its own mailing list. The up-to-date list of teams can be found at the Free Translation Project's homepage, `http://www.iro.umontreal.ca/contrib/po/HTML/`, in the "National teams" area.

Content-Type

Replace '`CHARSET`' with the character encoding used for your language, in your locale, or UTF-8. This field is needed for correct operation of the `msgmerge` and `msgfmt` programs, as well as for users whose locale's character encoding differs from yours (see Section 10.2.4 [Charset conversion], page 83).

You get the character encoding of your locale by running the shell command '`locale charmap`'. If the result is '`C`' or '`ANSI_X3.4-1968`', which is equivalent to '`ASCII`' (= '`US-ASCII`'), it means that your locale is not correctly configured. In this case, ask your translation team which charset to use. '`ASCII`' is not usable for any language except Latin.

Because the PO files must be portable to operating systems with less advanced internationalization facilities, the character encodings that can be used are limited to those supported by both GNU `libc` and GNU `libiconv`. These are: `ASCII`, `ISO-8859-1`, `ISO-8859-2`, `ISO-8859-3`, `ISO-8859-4`, `ISO-8859-5`, `ISO-8859-6`, `ISO-8859-7`, `ISO-8859-8`, `ISO-8859-9`, `ISO-8859-13`, `ISO-8859-15`, `KOI8-R`, `KOI8-U`, `CP850`, `CP866`, `CP874`, `CP932`, `CP949`, `CP950`, `CP1250`, `CP1251`, `CP1252`, `CP1253`, `CP1254`, `CP1255`, `CP1256`, `CP1257`, `GB2312`, `EUC-JP`, `EUC-KR`, `EUC-TW`, `BIG5`, `BIG5-HKSCS`, `GBK`, `GB18030`, `SHIFT_JIS`, `JOHAB`, `TIS-620`, `VISCII`, `UTF-8`.

In the GNU system, the following encodings are frequently used for the corresponding languages.

`ISO-8859-1` for Afrikaans, Albanian, Basque, Catalan, Dutch, English, Estonian, Faroese, Finnish, French, Galician, German, Greenlandic, Icelandic, Indonesian, Irish, Italian, Malay, Norwegian, Portuguese, Spanish, Swedish,

`ISO-8859-2` for Croatian, Czech, Hungarian, Polish, Romanian, Serbian, Slovak, Slovenian,

`ISO-8859-3` for Maltese,

`ISO-8859-5` for Macedonian, Serbian,

`ISO-8859-6` for Arabic,

`ISO-8859-7` for Greek,

`ISO-8859-8` for Hebrew,

`ISO-8859-9` for Turkish,

`ISO-8859-13` for Latvian, Lithuanian,

`ISO-8859-15` for Basque, Catalan, Dutch, English, Finnish, French, Galician, German, Irish, Italian, Portuguese, Spanish, Swedish,

`KOI8-R` for Russian,

`KOI8-U` for Ukrainian,

`CP1251` for Bulgarian, Byelorussian,

`GB2312`, `GBK`, `GB18030` for simplified writing of Chinese,

`BIG5`, `BIG5-HKSCS` for traditional writing of Chinese,

`EUC-JP` for Japanese,

`EUC-KR` for Korean,

`TIS-620` for Thai,

`UTF-8` for any language, including those listed above.

When single quote characters or double quote characters are used in translations for your language, and your locale's encoding is one of the ISO-8859-* charsets, it is best if you create your PO files in UTF-8 encoding, instead of your locale's encoding. This is because in UTF-8 the real quote characters can be represented (single quote characters: U+2018, U+2019, double quote characters: U+201C, U+201D), whereas none of ISO-8859-* charsets has them all. Users in UTF-8 locales will see the real quote characters, whereas users in ISO-8859-* locales will see the vertical apostrophe and the vertical double quote instead (because that's what the character set conversion will transliterate them to).

To enter such quote characters under X11, you can change your keyboard mapping using the `xmodmap` program. The X11 names of the quote characters are "leftsinglequotemark", "rightsinglequotemark", "leftdoublequotemark", "rightdoublequotemark", "singlelowquotemark", "doublelowquotemark".

Note that only recent versions of GNU Emacs support the UTF-8 encoding: Emacs 20 with Mule-UCS, and Emacs 21. As of January 2001, XEmacs doesn't support the UTF-8 encoding.

The character encoding name can be written in either upper or lower case. Usually upper case is preferred.

Content-Transfer-Encoding
: Set this to `8bit`.

Plural-Forms
: This field is optional. It is only needed if the PO file has plural forms. You can find them by searching for the '`msgid_plural`' keyword. The format of the plural forms field is described in Section 10.2.5 [Plural forms], page 84.

6 Updating Existing PO Files

6.1 Invoking the `msgmerge` Program

`msgmerge [`*option*`] `*def*`.po `*ref*`.pot`

The `msgmerge` program merges two Uniforum style .po files together. The *def*.po file is an existing PO file with translations which will be taken over to the newly created file as long as they still match; comments will be preserved, but extracted comments and file positions will be discarded. The *ref*.pot file is the last created PO file with up-to-date source references but old translations, or a PO Template file (generally created by `xgettext`); any translations or comments in the file will be discarded, however dot comments and file positions will be preserved. Where an exact match cannot be found, fuzzy matching is used to produce better results.

6.1.1 Input file location

'*def*`.po`' Translations referring to old sources.

'*ref*`.pot`' References to the new sources.

'`-D` *directory*'
'`--directory=`*directory*'
Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

'`-C` *file*'
'`--compendium=`*file*'
Specify an additional library of message translations. See Section 6.11 [Compendium], page 49. This option may be specified more than once.

6.1.2 Operation mode

'`-U`'
'`--update`'
Update *def*.po. Do nothing if *def*.po is already up to date.

6.1.3 Output file location

'`-o` *file*'
'`--output-file=`*file*'
Write output to specified file.

The results are written to standard output if no output file is specified or if it is '`-`'.

6.1.4 Output file location in update mode

The result is written back to *def*.po.

'`--backup=`*control*'
: Make a backup of *def*.po

'`--suffix=`*suffix*'
: Override the usual backup suffix.

The version control method may be selected via the `--backup` option or through the `VERSION_CONTROL` environment variable. Here are the values:

'`none`'
'`off`'
: Never make backups (even if `--backup` is given).

'`numbered`'
'`t`'
: Make numbered backups.

'`existing`'
'`nil`'
: Make numbered backups if numbered backups for this file already exist, otherwise make simple backups.

'`simple`'
'`never`'
: Always make simple backups.

The backup suffix is '`~`', unless set with `--suffix` or the `SIMPLE_BACKUP_SUFFIX` environment variable.

6.1.5 Operation modifiers

'`-m`'
'`--multi-domain`'
: Apply *ref*.pot to each of the domains in *def*.po.

6.1.6 Output details

'`--force-po`'
: Always write an output file even if it contains no message.

'`-i`'
'`--indent`'
: Write the .po file using indented style.

'`--no-location`'
: Do not write '`#:` *filename*`:`*line*' lines.

'`--add-location`'
: Generate '`#:` *filename*`:`*line*' lines (default).

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`-s`'
'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

'`-F`'
'`--sort-by-file`'
: Sort output by file location.

6.1.7 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

'`-v`'
'`--verbose`'
: Increase verbosity level.

'`-q`'
'`--quiet`'
'`--silent`'
: Suppress progress indicators.

6.2 Translated Entries

Each PO file entry for which the `msgstr` field has been filled with a translation, and which is not marked as fuzzy (see Section 6.3 [Fuzzy Entries], page 39), is said to be a *translated* entry. Only translated entries will later be compiled by GNU `msgfmt` and become usable in programs. Other entry types will be excluded; translation will not occur for them.

Some commands are more specifically related to translated entry processing.

t Find the next translated entry (`po-next-translated-entry`).

T Find the previous translated entry (`po-previous-translated-entry`).

The commands *t* (`po-next-translated-entry`) and *T* (`po-previous-translated-entry`) move forwards or backwards, chasing for an translated entry. If none is found, the search is extended and wraps around in the PO file buffer.

Translated entries usually result from the translator having edited in a translation for them, Section 6.6 [Modifying Translations], page 41. However, if the variable `po-auto-fuzzy-on-edit` is not `nil`, the entry having received a new translation first becomes a fuzzy entry, which ought to be later unfuzzied before becoming an official, genuine translated entry. See Section 6.3 [Fuzzy Entries], page 39.

6.3 Fuzzy Entries

Each PO file entry may have a set of *attributes*, which are qualities given a name and explicitely associated with the translation, using a special system comment. One of these attributes has the name `fuzzy`, and entries having this attribute are said to have a fuzzy translation. They are called fuzzy entries, for short.

Fuzzy entries, even if they account for translated entries for most other purposes, usually call for revision by the translator. Those may be produced by applying the program `msgmerge` to update an older translated PO files according to a new PO template file, when this tool hypothesises that some new `msgid` has been modified only slightly out of an older one, and chooses to pair what it thinks to be the old translation for the new modified entry. The slight alteration in the original string (the `msgid` string) should often be reflected in the translated string, and this requires the intervention of the translator. For this reason, `msgmerge` might mark some entries as being fuzzy.

Also, the translator may decide herself to mark an entry as fuzzy for her own convenience, when she wants to remember that the entry has to be later revisited. So, some commands are more specifically related to fuzzy entry processing.

z Find the next fuzzy entry (`po-next-fuzzy-entry`).

Z Find the previous fuzzy entry (`po-previous-fuzzy-entry`).

⟨TAB⟩ Remove the fuzzy attribute of the current entry (`po-unfuzzy`).

The commands *z* (`po-next-fuzzy-entry`) and *Z* (`po-previous-fuzzy-entry`) move forwards or backwards, chasing for a fuzzy entry. If none is found, the search is extended and wraps around in the PO file buffer.

The command ⟨TAB⟩ (`po-unfuzzy`) removes the fuzzy attribute associated with an entry, usually leaving it translated. Further, if the variable `po-auto-select-on-unfuzzy` has not the `nil` value, the ⟨TAB⟩ command will automatically chase for another interesting entry to work on. The initial value of `po-auto-select-on-unfuzzy` is `nil`.

The initial value of `po-auto-fuzzy-on-edit` is `nil`. However, if the variable `po-auto-fuzzy-on-edit` is set to `t`, any entry edited through the ⟨RET⟩ command is marked fuzzy, as a way to ensure some kind of double check, later. In this case, the usual paradigm is that an entry becomes fuzzy (if not already) whenever the translator modifies it. If she is satisfied with the translation, she then uses ⟨TAB⟩ to pick another entry to work on, clearing the fuzzy attribute on the same blow. If she is not satisfied yet, she merely uses ⟨SPC⟩ to chase another entry, leaving the entry fuzzy.

The translator may also use the ⟨DEL⟩ command (`po-fade-out-entry`) over any translated entry to mark it as being fuzzy, when she wants to easily leave a trace she wants to later return working at this entry.

Also, when time comes to quit working on a PO file buffer with the *q* command, the translator is asked for confirmation, if fuzzy string still exists.

6.4 Untranslated Entries

When `xgettext` originally creates a PO file, unless told otherwise, it initializes the `msgid` field with the untranslated string, and leaves the `msgstr` string to be empty. Such entries, having an empty translation, are said to be *untranslated* entries. Later, when the programmer slightly modifies some string right in the program, this change is later reflected in the PO file by the appearance of a new untranslated entry for the modified string.

The usual commands moving from entry to entry consider untranslated entries on the same level as active entries. Untranslated entries are easily recognizable by the fact they end with '`msgstr ""`'.

The work of the translator might be (quite naively) seen as the process of seeking for an untranslated entry, editing a translation for it, and repeating these actions until no untranslated entries remain. Some commands are more specifically related to untranslated entry processing.

u Find the next untranslated entry (`po-next-untranslated-entry`).

U Find the previous untranslated entry (`po-previous-untransted-entry`).

k Turn the current entry into an untranslated one (`po-kill-msgstr`).

The commands *u* (`po-next-untranslated-entry`) and *U* (`po-previous-untransted-entry`) move forwards or backwards, chasing for an untranslated entry. If none is found, the search is extended and wraps around in the PO file buffer.

An entry can be turned back into an untranslated entry by merely emptying its translation, using the command *k* (`po-kill-msgstr`). See Section 6.6 [Modifying Translations], page 41.

Also, when time comes to quit working on a PO file buffer with the *q* command, the translator is asked for confirmation, if some untranslated string still exists.

6.5 Obsolete Entries

By *obsolete* PO file entries, we mean those entries which are commented out, usually by `msgmerge` when it found that the translation is not needed anymore by the package being localized.

The usual commands moving from entry to entry consider obsolete entries on the same level as active entries. Obsolete entries are easily recognizable by the fact that all their lines start with `#`, even those lines containing `msgid` or `msgstr`.

Commands exist for emptying the translation or reinitializing it to the original untranslated string. Commands interfacing with the kill ring may force some previously saved text

into the translation. The user may interactively edit the translation. All these commands may apply to obsolete entries, carefully leaving the entry obsolete after the fact.

Moreover, some commands are more specifically related to obsolete entry processing.

o Find the next obsolete entry (`po-next-obsolete-entry`).

O Find the previous obsolete entry (`po-previous-obsolete-entry`).

⟨DEL⟩ Make an active entry obsolete, or zap out an obsolete entry (`po-fade-out-entry`).

The commands *o* (`po-next-obsolete-entry`) and *O* (`po-previous-obsolete-entry`) move forwards or backwards, chasing for an obsolete entry. If none is found, the search is extended and wraps around in the PO file buffer.

PO mode does not provide ways for un-commenting an obsolete entry and making it active, because this would reintroduce an original untranslated string which does not correspond to any marked string in the program sources. This goes with the philosophy of never introducing useless `msgid` values.

However, it is possible to comment out an active entry, so making it obsolete. GNU `gettext` utilities will later react to the disappearance of a translation by using the untranslated string. The command ⟨DEL⟩ (`po-fade-out-entry`) pushes the current entry a little further towards annihilation. If the entry is active (it is a translated entry), then it is first made fuzzy. If it is already fuzzy, then the entry is merely commented out, with confirmation. If the entry is already obsolete, then it is completely deleted from the PO file. It is easy to recycle the translation so deleted into some other PO file entry, usually one which is untranslated. See Section 6.6 [Modifying Translations], page 41.

Here is a quite interesting problem to solve for later development of PO mode, for those nights you are not sleepy. The idea would be that PO mode might become bright enough, one of these days, to make good guesses at retrieving the most probable candidate, among all obsolete entries, for initializing the translation of a newly appeared string. I think it might be a quite hard problem to do this algorithmically, as we have to develop good and efficient measures of string similarity. Right now, PO mode completely lets the decision to the translator, when the time comes to find the adequate obsolete translation, it merely tries to provide handy tools for helping her to do so.

6.6 Modifying Translations

PO mode prevents direct modification of the PO file, by the usual means Emacs gives for altering a buffer's contents. By doing so, it pretends helping the translator to avoid little clerical errors about the overall file format, or the proper quoting of strings, as those errors would be easily made. Other kinds of errors are still possible, but some may be caught and diagnosed by the batch validation process, which the translator may always trigger by the *V* command. For all other errors, the translator has to rely on her own judgment, and also on the linguistic reports submitted to her by the users of the translated package, having the same mother tongue.

When the time comes to create a translation, correct an error diagnosed mechanically or reported by a user, the translators have to resort to using the following commands for modifying the translations.

⟨RET⟩ Interactively edit the translation (`po-edit-msgstr`).

⟨LFD⟩
C-j Reinitialize the translation with the original, untranslated string (`po-msgid-to-msgstr`).

k Save the translation on the kill ring, and delete it (`po-kill-msgstr`).

w Save the translation on the kill ring, without deleting it (`po-kill-ring-save-msgstr`).

y Replace the translation, taking the new from the kill ring (`po-yank-msgstr`).

The command ⟨RET⟩ (`po-edit-msgstr`) opens a new Emacs window meant to edit in a new translation, or to modify an already existing translation. The new window contains a copy of the translation taken from the current PO file entry, all ready for edition, expunged of all quoting marks, fully modifiable and with the complete extent of Emacs modifying commands. When the translator is done with her modifications, she may use *C-c C-c* to close the subedit window with the automatically requoted results, or *C-c C-k* to abort her modifications. See Section 6.8 [Subedit], page 45, for more information.

The command ⟨LFD⟩ (`po-msgid-to-msgstr`) initializes, or reinitializes the translation with the original string. This command is normally used when the translator wants to redo a fresh translation of the original string, disregarding any previous work.

It is possible to arrange so, whenever editing an untranslated entry, the ⟨LFD⟩ command be automatically executed. If you set `po-auto-edit-with-msgid` to `t`, the translation gets initialised with the original string, in case none exists already. The default value for `po-auto-edit-with-msgid` is `nil`.

In fact, whether it is best to start a translation with an empty string, or rather with a copy of the original string, is a matter of taste or habit. Sometimes, the source language and the target language are so different that is simply best to start writing on an empty page. At other times, the source and target languages are so close that it would be a waste to retype a number of words already being written in the original string. A translator may also like having the original string right under her eyes, as she will progressively overwrite the original text with the translation, even if this requires some extra editing work to get rid of the original.

The command *k* (`po-kill-msgstr`) merely empties the translation string, so turning the entry into an untranslated one. But while doing so, its previous contents is put apart in a special place, known as the kill ring. The command *w* (`po-kill-ring-save-msgstr`) has also the effect of taking a copy of the translation onto the kill ring, but it otherwise leaves the entry alone, and does *not* remove the translation from the entry. Both commands use exactly the Emacs kill ring, which is shared between buffers, and which is well known already to Emacs lovers.

The translator may use *k* or *w* many times in the course of her work, as the kill ring may hold several saved translations. From the kill ring, strings may later be reinserted in various Emacs buffers. In particular, the kill ring may be used for moving translation strings between different entries of a single PO file buffer, or if the translator is handling many such buffers at once, even between PO files.

To facilitate exchanges with buffers which are not in PO mode, the translation string put on the kill ring by the `k` command is fully unquoted before being saved: external quotes are removed, multi-line strings are concatenated, and backslash escaped sequences are turned into their corresponding characters. In the special case of obsolete entries, the translation is also uncommented prior to saving.

The command `y` (`po-yank-msgstr`) completely replaces the translation of the current entry by a string taken from the kill ring. Following Emacs terminology, we then say that the replacement string is *yanked* into the PO file buffer. See section "Yanking" in *The Emacs Editor*. The first time `y` is used, the translation receives the value of the most recent addition to the kill ring. If `y` is typed once again, immediately, without intervening keystrokes, the translation just inserted is taken away and replaced by the second most recent addition to the kill ring. By repeating `y` many times in a row, the translator may travel along the kill ring for saved strings, until she finds the string she really wanted.

When a string is yanked into a PO file entry, it is fully and automatically requoted for complying with the format PO files should have. Further, if the entry is obsolete, PO mode then appropriately push the inserted string inside comments. Once again, translators should not burden themselves with quoting considerations besides, of course, the necessity of the translated string itself respective to the program using it.

Note that `k` or `w` are not the only commands pushing strings on the kill ring, as almost any PO mode command replacing translation strings (or the translator comments) automatically saves the old string on the kill ring. The main exceptions to this general rule are the yanking commands themselves.

To better illustrate the operation of killing and yanking, let's use an actual example, taken from a common situation. When the programmer slightly modifies some string right in the program, his change is later reflected in the PO file by the appearance of a new untranslated entry for the modified string, and the fact that the entry translating the original or unmodified string becomes obsolete. In many cases, the translator might spare herself some work by retrieving the unmodified translation from the obsolete entry, then initializing the untranslated entry `msgstr` field with this retrieved translation. Once this done, the obsolete entry is not wanted anymore, and may be safely deleted.

When the translator finds an untranslated entry and suspects that a slight variant of the translation exists, she immediately uses `m` to mark the current entry location, then starts chasing obsolete entries with `o`, hoping to find some translation corresponding to the unmodified string. Once found, she uses the ⟨DEL⟩ command for deleting the obsolete entry, knowing that ⟨DEL⟩ also *kills* the translation, that is, pushes the translation on the kill ring. Then, `r` returns to the initial untranslated entry, and `y` then *yanks* the saved translation right into the `msgstr` field. The translator is then free to use ⟨RET⟩ for fine tuning the translation contents, and maybe to later use `u`, then `m` again, for going on with the next untranslated string.

When some sequence of keys has to be typed over and over again, the translator may find it useful to become better acquainted with the Emacs capability of learning these sequences and playing them back under request. See section "Keyboard Macros" in *The Emacs Editor*.

6.7 Modifying Comments

Any translation work done seriously will raise many linguistic difficulties, for which decisions have to be made, and the choices further documented. These documents may be saved within the PO file in form of translator comments, which the translator is free to create, delete, or modify at will. These comments may be useful to herself when she returns to this PO file after a while.

Comments not having whitespace after the initial '`#`', for example, those beginning with '`#.`' or '`#:`', are *not* translator comments, they are exclusively created by other `gettext` tools. So, the commands below will never alter such system added comments, they are not meant for the translator to modify. See Section 2.2 [PO Files], page 10.

The following commands are somewhat similar to those modifying translations, so the general indications given for those apply here. See Section 6.6 [Modifying Translations], page 41.

`#` Interactively edit the translator comments (`po-edit-comment`).

`K` Save the translator comments on the kill ring, and delete it (`po-kill-comment`).

`W` Save the translator comments on the kill ring, without deleting it (`po-kill-ring-save-comment`).

`Y` Replace the translator comments, taking the new from the kill ring (`po-yank-comment`).

These commands parallel PO mode commands for modifying the translation strings, and behave much the same way as they do, except that they handle this part of PO file comments meant for translator usage, rather than the translation strings. So, if the descriptions given below are slightly succinct, it is because the full details have already been given. See Section 6.6 [Modifying Translations], page 41.

The command `#` (`po-edit-comment`) opens a new Emacs window containing a copy of the translator comments on the current PO file entry. If there are no such comments, PO mode understands that the translator wants to add a comment to the entry, and she is presented with an empty screen. Comment marks (`#`) and the space following them are automatically removed before edition, and reinstated after. For translator comments pertaining to obsolete entries, the uncommenting and recommenting operations are done twice. Once in the editing window, the keys *`C-c C-c`* allow the translator to tell she is finished with editing the comment. See Section 6.8 [Subedit], page 45, for further details.

Functions found on `po-subedit-mode-hook`, if any, are executed after the string has been inserted in the edit buffer.

The command *`K`* (`po-kill-comment`) gets rid of all translator comments, while saving those comments on the kill ring. The command *`W`* (`po-kill-ring-save-comment`) takes a copy of the translator comments on the kill ring, but leaves them undisturbed in the current entry. The command *`Y`* (`po-yank-comment`) completely replaces the translator comments by a string taken at the front of the kill ring. When this command is immediately repeated, the comments just inserted are withdrawn, and replaced by other strings taken along the kill ring.

On the kill ring, all strings have the same nature. There is no distinction between *translation* strings and *translator comments* strings. So, for example, let's presume the translator has just finished editing a translation, and wants to create a new translator comment to document why the previous translation was not good, just to remember what was the problem. Foreseeing that she will do that in her documentation, the translator may want to quote the previous translation in her translator comments. To do so, she may initialize the translator comments with the previous translation, still at the head of the kill ring. Because editing already pushed the previous translation on the kill ring, she merely has to type `M-w` prior to `#`, and the previous translation will be right there, all ready for being introduced by some explanatory text.

On the other hand, presume there are some translator comments already and that the translator wants to add to those comments, instead of wholly replacing them. Then, she should edit the comment right away with `#`. Once inside the editing window, she can use the regular Emacs commands `C-y` (`yank`) and `M-y` (`yank-pop`) to get the previous translation where she likes.

6.8 Details of Sub Edition

The PO subedit minor mode has a few peculiarities worth being described in fuller detail. It installs a few commands over the usual editing set of Emacs, which are described below.

`C-c C-c` Complete edition (`po-subedit-exit`).

`C-c C-k` Abort edition (`po-subedit-abort`).

`C-c C-a` Consult auxiliary PO files (`po-subedit-cycle-auxiliary`).

The window's contents represents a translation for a given message, or a translator comment. The translator may modify this window to her heart's content. Once this is done, the command `C-c C-c` (`po-subedit-exit`) may be used to return the edited translation into the PO file, replacing the original translation, even if it moved out of sight or if buffers were switched.

If the translator becomes unsatisfied with her translation or comment, to the extent she prefers keeping what was existent prior to the ⟨RET⟩ or `#` command, she may use the command `C-c C-k` (`po-subedit-abort`) to merely get rid of edition, while preserving the original translation or comment. Another way would be for her to exit normally with `C-c C-c`, then type `U` once for undoing the whole effect of last edition.

The command `C-c C-a` (`po-subedit-cycle-auxiliary`) allows for glancing through translations already achieved in other languages, directly while editing the current translation. This may be quite convenient when the translator is fluent at many languages, but of course, only makes sense when such completed auxiliary PO files are already available to her (see Section 6.10 [Auxiliary], page 48).

Functions found on `po-subedit-mode-hook`, if any, are executed after the string has been inserted in the edit buffer.

While editing her translation, the translator should pay attention to not inserting unwanted ⟨RET⟩ (newline) characters at the end of the translated string if those are not meant to be there, or to removing such characters when they are required. Since these characters

are not visible in the editing buffer, they are easily introduced by mistake. To help her, ⟨RET⟩ automatically puts the character < at the end of the string being edited, but this < is not really part of the string. On exiting the editing window with `C-c C-c`, PO mode automatically removes such < and all whitespace added after it. If the translator adds characters after the terminating <, it looses its delimiting property and integrally becomes part of the string. If she removes the delimiting <, then the edited string is taken *as is*, with all trailing newlines, even if invisible. Also, if the translated string ought to end itself with a genuine <, then the delimiting < may not be removed; so the string should appear, in the editing window, as ending with two < in a row.

When a translation (or a comment) is being edited, the translator may move the cursor back into the PO file buffer and freely move to other entries, browsing at will. If, with an edition pending, the translator wanders in the PO file buffer, she may decide to start modifying another entry. Each entry being edited has its own subedit buffer. It is possible to simultaneously edit the translation *and* the comment of a single entry, or to edit entries in different PO files, all at once. Typing ⟨RET⟩ on a field already being edited merely resumes that particular edit. Yet, the translator should better be comfortable at handling many Emacs windows!

Pending subedits may be completed or aborted in any order, regardless of how or when they were started. When many subedits are pending and the translator asks for quitting the PO file (with the `q` command), subedits are automatically resumed one at a time, so she may decide for each of them.

6.9 C Sources Context

PO mode is particularily powerful when used with PO files created through GNU `gettext` utilities, as those utilities insert special comments in the PO files they generate. Some of these special comments relate the PO file entry to exactly where the untranslated string appears in the program sources.

When the translator gets to an untranslated entry, she is fairly often faced with an original string which is not as informative as it normally should be, being succinct, cryptic, or otherwise ambiguous. Before chosing how to translate the string, she needs to understand better what the string really means and how tight the translation has to be. Most of times, when problems arise, the only way left to make her judgment is looking at the true program sources from where this string originated, searching for surrounding comments the programmer might have put in there, and looking around for helping clues of *any* kind.

Surely, when looking at program sources, the translator will receive more help if she is a fluent programmer. However, even if she is not versed in programming and feels a little lost in C code, the translator should not be shy at taking a look, once in a while. It is most probable that she will still be able to find some of the hints she needs. She will learn quickly to not feel uncomfortable in program code, paying more attention to programmer's comments, variable and function names (if he dared chosing them well), and overall organization, than to programmation itself.

The following commands are meant to help the translator at getting program source context for a PO file entry.

`s` Resume the display of a program source context, or cycle through them (`po-cycle-source-reference`).

`M-s` Display of a program source context selected by menu (`po-select-source-reference`).

`S` Add a directory to the search path for source files (`po-consider-source-path`).

`M-S` Delete a directory from the search path for source files (`po-ignore-source-path`).

The commands `s` (`po-cycle-source-reference`) and `M-s` (`po-select-source-reference`) both open another window displaying some source program file, and already positioned in such a way that it shows an actual use of the string to be translated. By doing so, the command gives source program context for the string. But if the entry has no source context references, or if all references are unresolved along the search path for program sources, then the command diagnoses this as an error.

Even if `s` (or `M-s`) opens a new window, the cursor stays in the PO file window. If the translator really wants to get into the program source window, she ought to do it explicitly, maybe by using command `0`.

When `s` is typed for the first time, or for a PO file entry which is different of the last one used for getting source context, then the command reacts by giving the first context available for this entry, if any. If some context has already been recently displayed for the current PO file entry, and the translator wandered off to do other things, typing `s` again will merely resume, in another window, the context last displayed. In particular, if the translator moved the cursor away from the context in the source file, the command will bring the cursor back to the context. By using `s` many times in a row, with no other commands intervening, PO mode will cycle to the next available contexts for this particular entry, getting back to the first context once the last has been shown.

The command `M-s` behaves differently. Instead of cycling through references, it lets the translator choose a particular reference among many, and displays that reference. It is best used with completion, if the translator types ⟨TAB⟩ immediately after `M-s`, in response to the question, she will be offered a menu of all possible references, as a reminder of which are the acceptable answers. This command is useful only where there are really many contexts available for a single string to translate.

Program source files are usually found relative to where the PO file stands. As a special provision, when this fails, the file is also looked for, but relative to the directory immediately above it. Those two cases take proper care of most PO files. However, it might happen that a PO file has been moved, or is edited in a different place than its normal location. When this happens, the translator should tell PO mode in which directory normally sits the genuine PO file. Many such directories may be specified, and all together, they constitute what is called the *search path* for program sources. The command `S` (`po-consider-source-path`) is used to interactively enter a new directory at the front of the search path, and the command `M-S` (`po-ignore-source-path`) is used to select, with completion, one of the directories she does not want anymore on the search path.

6.10 Consulting Auxiliary PO Files

PO mode is able to help the knowledgeable translator, being fluent in many languages, at taking advantage of translations already achieved in other languages she just happens to know. It provides these other language translations as additional context for her own work. Moreover, it has features to ease the production of translations for many languages at once, for translators preferring to work in this way.

An *auxiliary* PO file is an existing PO file meant for the same package the translator is working on, but targeted to a different mother tongue language. Commands exist for declaring and handling auxiliary PO files, and also for showing contexts for the entry under work.

Here are the auxiliary file commands available in PO mode.

`a` Seek auxiliary files for another translation for the same entry (`po-cycle-auxiliary`).

`C-c C-a` Switch to a particular auxiliary file (`po-select-auxiliary`).

`A` Declare this PO file as an auxiliary file (`po-consider-as-auxiliary`).

`M-A` Remove this PO file from the list of auxiliary files (`po-ignore-as-auxiliary`).

Command `A` (`po-consider-as-auxiliary`) adds the current PO file to the list of auxiliary files, while command `M-A` (`po-ignore-as-auxiliary` just removes it.

The command `a` (`po-cycle-auxiliary`) seeks all auxiliary PO files, round-robin, searching for a translated entry in some other language having an `msgid` field identical as the one for the current entry. The found PO file, if any, takes the place of the current PO file in the display (its window gets on top). Before doing so, the current PO file is also made into an auxiliary file, if not already. So, `a` in this newly displayed PO file will seek another PO file, and so on, so repeating `a` will eventually yield back the original PO file.

The command `C-c C-a` (`po-select-auxiliary`) asks the translator for her choice of a particular auxiliary file, with completion, and then switches to that selected PO file. The command also checks if the selected file has an `msgid` field identical as the one for the current entry, and if yes, this entry becomes current. Otherwise, the cursor of the selected file is left undisturbed.

For all this to work fully, auxiliary PO files will have to be normalized, in that way that `msgid` fields should be written *exactly* the same way. It is possible to write `msgid` fields in various ways for representing the same string, different writing would break the proper behaviour of the auxiliary file commands of PO mode. This is not expected to be much a problem in practice, as most existing PO files have their `msgid` entries written by the same GNU `gettext` tools.

However, PO files initially created by PO mode itself, while marking strings in source files, are normalised differently. So are PO files resulting of the the '`M-x normalize`' command. Until these discrepancies between PO mode and other GNU `gettext` tools get fully resolved, the translator should stay aware of normalisation issues.

6.11 Using Translation Compendia

A *compendium* is a special PO file containing a set of translations recurring in many different packages. The translator can use gettext tools to build a new compendium, to add entries to her compendium, and to initialize untranslated entries, or to update already translated entries, from translations kept in the compendium.

6.11.1 Creating Compendia

Basically every PO file consisting of translated entries only can be declared as a valid compendium. Often the translator wants to have special compendia; let's consider two cases: *concatenating PO files* and *extracting a message subset from a PO file*.

6.11.1.1 Concatenate PO Files

To concatenate several valid PO files into one compendium file you can use '`msgcomm`' or '`msgcat`' (the latter preferred):

```
msgcat -o compendium.po file1.po file2.po
```

By default, `msgcat` will accumulate divergent translations for the same string. Those occurences will be marked as `fuzzy` and highly visible decorated; calling `msgcat` on '`file1.po`':

```
#: src/hello.c:200
#, c-format
msgid "Report bugs to <%s>.\n"
msgstr "Comunicar ‘bugs’ a <%s>.\n"
```

and '`file2.po`':

```
#: src/bye.c:100
#, c-format
msgid "Report bugs to <%s>.\n"
msgstr "Comunicar \"bugs\" a <%s>.\n"
```

will result in:

```
#: src/hello.c:200 src/bye.c:100
#, fuzzy, c-format
msgid "Report bugs to <%s>.\n"
msgstr ""
"#-#-#-#-#  file1.po  #-#-#-#-#\n"
"Comunicar ‘bugs’ a <%s>.\n"
"#-#-#-#-#  file2.po  #-#-#-#-#\n"
"Comunicar \"bugs\" a <%s>.\n"
```

The translator will have to resolve this "conflict" manually; she has to decide whether the first or the second version is appropriate (or provide a new translation), to delete the "marker lines", and finally to remove the `fuzzy` mark.

If the translator knows in advance the first found translation of a message is always the best translation she can make use to the '`--use-first`' switch:

```
msgcat --use-first -o compendium.po file1.po file2.po
```

A good compendium file must not contain `fuzzy` or untranslated entries. If input files are "dirty" you must preprocess the input files or postprocess the result using '`msgattrib --translated --no-fuzzy`'.

6.11.1.2 Extract a Message Subset from a PO File

Nobody wants to translate the same messages again and again; thus you may wish to have a compendium file containing '`getopt.c`' messages.

To extract a message subset (e.g., all '`getopt.c`' messages) from an existing PO file into one compendium file you can use '`msggrep`':

```
msggrep --location src/getopt.c -o compendium.po file.po
```

6.11.2 Using Compendia

You can use a compendium file to initialize a translation from scratch or to update an already existing translation.

6.11.2.1 Initialize a New Translation File

Since a PO file with translations does not exist the translator can merely use '`/dev/null`' to fake the "old" translation file.

```
msgmerge --compendium compendium.po -o file.po /dev/null file.pot
```

6.11.2.2 Update an Existing Translation File

Concatenate the compendium file(s) and the existing PO, merge the result with the POT file and remove the obsolete entries (optional, here done using '`sed`'):

```
msgcat --use-first -o update.po compendium1.po compendium2.po file.po
msgmerge update.po file.pot | sed -e '/^#~/d' > file.po
```

7 Manipulating PO Files

Sometimes it is necessary to manipulate PO files in a way that is better performed automatically than by hand. GNU `gettext` includes a complete set of tools for this purpose.

When merging two packages into a single package, the resulting POT file will be the concatenation of the two packages' POT files. Thus the maintainer must concatenate the two existing package translations into a single translation catalog, for each language. This is best performed using '`msgcat`'. It is then the translators' duty to deal with any possible conflicts that arose during the merge.

When a translator takes over the translation job from another translator, but she uses a different character encoding in her locale, she will convert the catalog to her character encoding. This is best done through the '`msgconv`' program.

When a maintainer takes a source file with tagged messages from another package, he should also take the existing translations for this source file (and not let the translators do the same job twice). One way to do this is through '`msggrep`', another is to create a POT file for that source file and use '`msgmerge`'.

When a translator wants to adjust some translation catalog for a special dialect or orthography — for example, German as written in Switzerland versus German as written in Germany — she needs to apply some text processing to every message in the catalog. The tool for doing this is '`msgfilter`'.

Another use of `msgfilter` is to produce approximately the POT file for which a given PO file was made. This can be done through a filter command like '`msgfilter sed -e d | sed -e '/^# /d'`'. Note that the original POT file may have had different comments and different plural message counts, that's why it's better to use the original POT file if available.

When a translator wants to check her translations, for example according to orthography rules or using a non-interactive spell checker, she can do so using the '`msgexec`' program.

When third party tools create PO or POT files, sometimes duplicates cannot be avoided. But the GNU `gettext` tools give an error when they encounter duplicate msgids in the same file and in the same domain. To merge duplicates, the '`msguniq`' program can be used.

'`msgcomm`' is a more general tool for keeping or throwing away duplicates, occurring in different files.

'`msgcmp`' can be used to check whether a translation catalog is completely translated.

'`msgattrib`' can be used to select and extract only the fuzzy or untranslated messages of a translation catalog.

'`msgen`' is useful as a first step for preparing English translation catalogs. It copies each message's msgid to its msgstr.

7.1 Invoking the `msgcat` Program

`msgcat` [*option*] [*inputfile*]...

The `msgcat` program concatenates and merges the specified PO files. It finds messages which are common to two or more of the specified PO files. By using the `--more-than`

option, greater commonality may be requested before messages are printed. Conversely, the `--less-than` option may be used to specify less commonality before messages are printed (i.e. '`--less-than=2`' will only print the unique messages). Translations, comments and extract comments will be cumulated, except that if `--use-first` is specified, they will be taken from the first PO file to define them. File positions from all PO files will be cumulated.

7.1.1 Input file location

'*inputfile* ...'
: Input files.

'`-f` *file*'
'`--files-from=`*file*'
: Read the names of the input files from *file* instead of getting them from the command line.

'`-D` *directory*'
'`--directory=`*directory*'
: Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If *inputfile* is '`-`', standard input is read.

7.1.2 Output file location

'`-o` *file*'
'`--output-file=`*file*'
: Write output to specified file.

The results are written to standard output if no output file is specified or if it is '`-`'.

7.1.3 Message selection

'`-<` *number*'
'`--less-than=`*number*'
: Print messages with less than *number* definitions, defaults to infinite if not set.

'`->` *number*'
'`--more-than=`*number*'
: Print messages with more than *number* definitions, defaults to 0 if not set.

'`-u`'
'`--unique`'
: Shorthand for '`--less-than=2`'. Requests that only unique messages be printed.

7.1.4 Output details

'`-t`'
'`--to-code=`*name*'
: Specify encoding for output.

'`--use-first`'
: Use first available translation for each message. Don't merge several translations into one.

'`--force-po`'
: Always write an output file even if it contains no message.

'`-i`'
'`--indent`'
: Write the .po file using indented style.

'`--no-location`'
: Do not write '`#:` *filename*`:`*line*' lines.

'`-n`'
'`--add-location`'
: Generate '`#:` *filename*`:`*line*' lines (default).

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`-s`'
'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

'`-F`'
'`--sort-by-file`'
: Sort output by file location.

7.1.5 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

7.2 Invoking the `msgconv` Program

`msgconv` [*option*] [*inputfile*]

The `msgconv` program converts a translation catalog to a different character encoding.

7.2.1 Input file location

'*inputfile*' Input PO file.

'`-D` *directory*'
'`--directory=`*directory*'
: Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If no *inputfile* is given or if it is '`-`', standard input is read.

7.2.2 Output file location

'`-o` *file*'
'`--output-file=`*file*'
: Write output to specified file.

The results are written to standard output if no output file is specified or if it is '`-`'.

7.2.3 Conversion target

'`-t`'
'`--to-code=`*name*'
: Specify encoding for output.

The default encoding is the current locale's encoding.

7.2.4 Output details

'`--force-po`'
: Always write an output file even if it contains no message.

'`-i`'
'`--indent`'
: Write the .po file using indented style.

'`--no-location`'
: Do not write '`#:` *filename*`:`*line*' lines.

'`--add-location`'
: Generate '`#:` *filename*`:`*line*' lines (default).

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`-s`'
'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

'`-F`'
'`--sort-by-file`'
: Sort output by file location.

7.2.5 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

7.3 Invoking the `msggrep` Program

`msggrep [`*option*`] [`*inputfile*`]`

The `msggrep` program extracts all messages of a translation catalog that match a given pattern or belong to some given source files.

7.3.1 Input file location

'*inputfile*' Input PO file.

'`-D` *directory*'
'`--directory=`*directory*'
: Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If no *inputfile* is given or if it is '-', standard input is read.

7.3.2 Output file location

'`-o` *file*'
'`--output-file=`*file*'
: Write output to specified file.

The results are written to standard output if no output file is specified or if it is '-'.

7.3.3 Message selection

```
[-N sourcefile]... [-M domainname]...
[-K msgid-pattern] [-T msgstr-pattern] [-C comment-pattern]
```

A message is selected if

- it comes from one of the specified source files,
- or if it comes from one of the specified domains,
- or if '`-K`' is given and its key (msgid or msgid_plural) matches *msgid-pattern*,
- or if '`-T`' is given and its translation (msgstr) matches *msgstr-pattern*,
- or if '`-C`' is given and the translator's comment matches *comment-pattern*.

When more than one selection criterion is specified, the set of selected messages is the union of the selected messages of each criterion.

msgid-pattern or *msgstr-pattern* syntax:

```
[-E | -F] [-e pattern | -f file]...
```

*pattern*s are basic regular expressions by default, or extended regular expressions if -E is given, or fixed strings if -F is given.

'`-N` *sourcefile*'
'`--location=`*sourcefile*'
: Select messages extracted from *sourcefile*. *sourcefile* can be either a literal file name or a wildcard pattern.

'`-M` *domainname*'
'`--domain=`*domainname*'
: Select messages belonging to domain *domainname*.

'`-K`'
'`--msgid`' Start of patterns for the msgid.

'`-T`'
'`--msgstr`'
: Start of patterns for the msgstr.

'`-E`'
'`--extended-regexp`'
: Specify that *pattern* is an extended regular expression.

'`-F`'
'`--fixed-strings`'
: Specify that *pattern* is a set of newline-separated strings.

'`-e` *pattern*'
'`--regexp=`*pattern*'
: Use *pattern* as a regular expression.

'`-f` *file*'
'`--file=`*file*'
: Obtain *pattern* from *file*.

'`-i`'
'`--ignore-case`'
: Ignore case distinctions.

7.3.4 Output details

'`--force-po`'
: Always write an output file even if it contains no message.

'`--indent`'
: Write the .po file using indented style.

'`--no-location`'
: Do not write '`#:` *filename*`:`*line*' lines.

'`--add-location`'
: Generate '`#:` *filename*`:`*line*' lines (default).

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

'`--sort-by-file`'
: Sort output by file location.

7.3.5 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
Output version information and exit.

7.4 Invoking the `msgfilter` Program

`msgfilter [`*option*`] `*filter*` [`*filter-option*`]`

The `msgfilter` program applies a filter to all translations of a translation catalog.

7.4.1 Input file location

'`-i` *inputfile*'
'`--input=`*inputfile*'
Input PO file.

'`-D` *directory*'
'`--directory=`*directory*'
Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If no *inputfile* is given or if it is '`-`', standard input is read.

7.4.2 Output file location

'`-o` *file*'
'`--output-file=`*file*'
Write output to specified file.

The results are written to standard output if no output file is specified or if it is '`-`'.

7.4.3 The filter

The *filter* can be any program that reads a translation from standard input and writes a modified translation to standard output. A frequently used filter is '`sed`'.

Note: It is your responsibility to ensure that the *filter* can cope with input encoded in the translation catalog's encoding. If the *filter* wants input in a particular encoding, you can in a first step convert the translation catalog to that encoding using the '`msgconv`' program, before invoking '`msgfilter`'. If the *filter* wants input in the locale's encoding, but you want to avoid the locale's encoding, then you can first convert the translation catalog to UTF-8 using the '`msgconv`' program and then make '`msgfilter`' work in an UTF-8 locale, by using the `LC_ALL` environment variable.

Note: Most translations in a translation catalog don't end with a newline character. For this reason, it is important that the *filter* recognizes its last input line even if it ends without a newline, and that it doesn't add an undesired trailing newline at the end. The '`sed`' program on some platforms is known to ignore the last line of input if it is not terminated with a newline. You can use GNU `sed` instead; it does not have this limitation.

7.4.4 Useful *filter-option*s when the *filter* is '`sed`'

'`-e` *script*'
'`--expression=`*script*'
: Add *script* to the commands to be executed.

'`-f` *scriptfile*'
'`--file=`*scriptfile*'
: Add the contents of *scriptfile* to the commands to be executed.

'`-n`'
'`--quiet`'
'`--silent`'
: Suppress automatic printing of pattern space.

7.4.5 Output details

'`--force-po`'
: Always write an output file even if it contains no message.

'`--indent`'
: Write the .po file using indented style.

'`--keep-header`'
: Keep the header entry, i.e. the message with '`msgid ""`', unmodified, instead of filtering it. By default, the header entry is subject to filtering like any other message.

'`--no-location`'
: Do not write '`#:` *filename*`:`*line*' lines.

'`--add-location`'
: Generate '`#:` *filename*`:`*line*' lines (default).

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`-s`'
'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

'`-F`'
'`--sort-by-file`'
: Sort output by file location.

7.4.6 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

7.5 Invoking the `msguniq` Program

`msguniq` [*option*] [*inputfile*]

The `msguniq` program unifies duplicate translations in a translation catalog. It finds duplicate translations of the same message ID. Such duplicates are invalid input for other programs like `msgfmt`, `msgmerge` or `msgcat`. By default, duplicates are merged together. When using the '`--repeated`' option, only duplicates are output, and all other messages are discarded. Comments and extracted comments will be cumulated, except that if '`--use-first`' is specified, they will be taken from the first translation. File positions will be cumulated. When using the '`--unique`' option, duplicates are discarded.

7.5.1 Input file location

'*inputfile*' Input PO file.

'`-D` *directory*'
'`--directory=`*directory*'
: Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If no *inputfile* is given or if it is '`-`', standard input is read.

7.5.2 Output file location

'`-o` *file*'
'`--output-file=`*file*'
: Write output to specified file.

The results are written to standard output if no output file is specified or if it is '`-`'.

7.5.3 Message selection

'`-d`'
'`--repeated`'
: Print only duplicates.

'`-u`'
'`--unique`'
: Print only unique messages, discard duplicates.

7.5.4 Output details

'`-t`'
'`--to-code=`*name*'
: Specify encoding for output.

'`--use-first`'
: Use first available translation for each message. Don't merge several translations into one.

'`--force-po`'
: Always write an output file even if it contains no message.

'`-i`'
'`--indent`'
: Write the .po file using indented style.

'`--no-location`'
: Do not write '`#:` *filename*`:`*line*' lines.

'`-n`'
'`--add-location`'
: Generate '`#:` *filename*`:`*line*' lines (default).

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`-s`'
'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

'`-F`'
'`--sort-by-file`'
: Sort output by file location.

7.5.5 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

7.6 Invoking the `msgcomm` Program

`msgcomm [`*option*`] [`*inputfile*`]...`

The `msgcomm` program finds messages which are common to two or more of the specified PO files. By using the `--more-than` option, greater commonality may be requested before messages are printed. Conversely, the `--less-than` option may be used to specify less commonality before messages are printed (i.e. '`--less-than=2`' will only print the unique messages). Translations, comments and extract comments will be preserved, but only from the first PO file to define them. File positions from all PO files will be cumulated.

7.6.1 Input file location

'*inputfile* ...'
: Input files.

'`-f` *file*'
'`--files-from=`*file*'
: Read the names of the input files from *file* instead of getting them from the command line.

'`-D` *directory*'
'`--directory=`*directory*'
: Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If *inputfile* is '`-`', standard input is read.

7.6.2 Output file location

'`-o` *file*'
'`--output-file=`*file*'
: Write output to specified file.

The results are written to standard output if no output file is specified or if it is '`-`'.

7.6.3 Message selection

'`-<` *number*'
'`--less-than=`*number*'
: Print messages with less than *number* definitions, defaults to infinite if not set.

'`->` *number*'
'`--more-than=`*number*'
: Print messages with more than *number* definitions, defaults to 1 if not set.

'`-u`'
'`--unique`'
: Shorthand for '`--less-than=2`'. Requests that only unique messages be printed.

7.6.4 Output details

'`--force-po`'
: Always write an output file even if it contains no message.

'`-i`'
'`--indent`'
: Write the .po file using indented style.

'`--no-location`'
: Do not write '`#:` *filename*`:`*line*' lines.

'`-n`'
'`--add-location`'
: Generate '`#:` *filename*`:`*line*' lines (default).

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`-s`'
'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

'`-F`'
'`--sort-by-file`'
: Sort output by file location.

'`--omit-header`'
: Don't write header with '`msgid ""`' entry.

7.6.5 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

7.7 Invoking the `msgcmp` Program

`msgcmp` [*option*] *def*.`po` *ref*.`pot`

The `msgcmp` program compares two Uniforum style .po files to check that both contain the same set of msgid strings. The *def*.po file is an existing PO file with the translations. The *ref*.pot file is the last created PO file, or a PO Template file (generally created by `xgettext`). This is useful for checking that you have translated each and every message in your program. Where an exact match cannot be found, fuzzy matching is used to produce better diagnostics.

7.7.1 Input file location

'*def*.`po`' Translations.

'*ref*.`pot`' References to the sources.

'`-D` *directory*'
'`--directory=`*directory*'
: Add *directory* to the list of directories. Source files are searched relative to this list of directories.

7.7.2 Operation modifiers

'`-m`'
'`--multi-domain`'
: Apply *ref*.pot to each of the domains in *def*.po.

7.7.3 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
Output version information and exit.

7.8 Invoking the `msgattrib` Program

`msgattrib [`*option*`] [`*inputfile*`]`

The `msgattrib` program filters the messages of a translation catalog according to their attributes, and manipulates the attributes.

7.8.1 Input file location

'*inputfile*' Input PO file.

'`-D` *directory*'
'`--directory=`*directory*'
Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If no *inputfile* is given or if it is '`-`', standard input is read.

7.8.2 Output file location

'`-o` *file*'
'`--output-file=`*file*'
Write output to specified file.

The results are written to standard output if no output file is specified or if it is '`-`'.

7.8.3 Message selection

'`--translated`'
Keep translated messages, remove untranslated messages.

'`--untranslated`'
Keep untranslated messages, remove translated messages.

'`--no-fuzzy`'
Remove 'fuzzy' marked messages.

'`--only-fuzzy`'
Keep 'fuzzy' marked messages, remove all other messsages.

'`--no-obsolete`'
Remove obsolete `#~` messages.

'`--only-obsolete`'
: Keep obsolete #~ messages, remove all other messages.

7.8.4 Attribute manipulation

Attributes are modified after the message selection/removal has been performed.

'`--set-fuzzy`'
: Set all messages 'fuzzy'.

'`--clear-fuzzy`'
: Set all messages non-'fuzzy'.

'`--set-obsolete`'
: Set all messages obsolete.

'`--clear-obsolete`'
: Set all messages non-obsolete.

'`--fuzzy`'
: Synonym for '`--only-fuzzy --clear-fuzzy`': It keeps only the fuzzy messages and removes their 'fuzzy' mark.

'`--obsolete`'
: Synonym for '`--only-obsolete --clear-obsolete`': It keeps only the obsolete messages and makes them non-obsolete.

7.8.5 Output details

'`--force-po`'
: Always write an output file even if it contains no message.

'`-i`'
'`--indent`'
: Write the .po file using indented style.

'`--no-location`'
: Do not write '`#:` *filename*`:`*line*' lines.

'`-n`'
'`--add-location`'
: Generate '`#:` *filename*`:`*line*' lines (default).

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`-s`'
'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

'`-F`'
'`--sort-by-file`'
: Sort output by file location.

7.8.6 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

7.9 Invoking the `msgen` Program

`msgen [`*option*`]` *inputfile*

The `msgen` program creates an English translation catalog. The input file is the last created English PO file, or a PO Template file (generally created by xgettext). Untranslated entries are assigned a translation that is identical to the msgid, and are marked fuzzy.

Note: '`msginit --no-translator --locale=en`' performs a very similar task. The main difference is that `msginit` cares specially about the header entry, whereas `msgen` doesn't.

7.9.1 Input file location

'*inputfile*' Input PO or POT file.

'`-D` *directory*'
'`--directory=`*directory*'
: Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If *inputfile* is '`-`', standard input is read.

7.9.2 Output file location

'`-o` *file*'
'`--output-file=`*file*'
: Write output to specified file.

The results are written to standard output if no output file is specified or if it is '`-`'.

7.9.3 Output details

'`--force-po`'
: Always write an output file even if it contains no message.

'`-i`'
'`--indent`'
: Write the .po file using indented style.

'`--no-location`'
: Do not write '`#:` *filename*`:`*line*' lines.

'`--add-location`'
: Generate '`#:` *filename*`:`*line*' lines (default).

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`-s`'
'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

'`-F`'
'`--sort-by-file`'
: Sort output by file location.

7.9.4 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

7.10 Invoking the `msgexec` Program

`msgexec` [*option*] *command* [*command-option*]

The `msgexec` program applies a command to all translations of a translation catalog. The *command* can be any program that reads a translation from standard input. It is invoked once for each translation. Its output becomes msgexec's output. `msgexec`'s return code is the maximum return code across all invocations.

A special builtin command called '`0`' outputs the translation, followed by a null byte. The output of '`msgexec 0`' is suitable as input for '`xargs -0`'.

During each *command* invocation, the environment variable `MSGEXEC_MSGID` is bound to the message's msgid, and the environment variable `MSGEXEC_LOCATION` is bound to the location in the PO file of the message.

Note: It is your responsibility to ensure that the *command* can cope with input encoded in the translation catalog's encoding. If the *command* wants input in a particular encoding, you can in a first step convert the translation catalog to that encoding using the '`msgconv`' program, before invoking '`msgexec`'. If the *command* wants input in the locale's encoding, but you want to avoid the locale's encoding, then you can first convert the translation catalog to UTF-8 using the '`msgconv`' program and then make '`msgexec`' work in an UTF-8 locale, by using the `LC_ALL` environment variable.

7.10.1 Input file location

'`-i` *inputfile*'
'`--input=`*inputfile*'
: Input PO file.

'`-D` *directory*'
'`--directory=`*directory*'
: Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If no *inputfile* is given or if it is '`-`', standard input is read.

7.10.2 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

8 Producing Binary MO Files

8.1 Invoking the `msgfmt` Program

`msgfmt` [*option*] *filename*`.po` ...

The `msgfmt` programs generates a binary message catalog from a textual translation description.

8.1.1 Input file location

'*filename*`.po` ...'

'`-D` *directory*'
'`--directory=`*directory*'
: Add *directory* to the list of directories. Source files are searched relative to this list of directories. The resulting '`.po`' file will be written relative to the current directory, though.

If an input file is '`-`', standard input is read.

8.1.2 Operation mode

'`-j`'
'`--java`' Java mode: generate a Java `ResourceBundle` class.

'`--java2`' Like –java, and assume Java2 (JDK 1.2 or higher).

'`--tcl`' Tcl mode: generate a tcl/msgcat '`.msg`' file.

8.1.3 Output file location

'`-o` *file*'
'`--output-file=`*file*'
: Write output to specified file.

'`--strict`'
: Direct the program to work strictly following the Uniforum/Sun implementation. Currently this only affects the naming of the output file. If this option is not given the name of the output file is the same as the domain name. If the strict Uniforum mode is enabled the suffix '`.mo`' is added to the file name if it is not already present.

  We find this behaviour of Sun's implementation rather silly and so by default this mode is *not* selected.

If the output *file* is '`-`', output is written to standard output.

8.1.4 Output file location in Java mode

'`-r` *resource*'
'`--resource=`*resource*'
: Specify the resource name.

'`-l` *locale*'
'`--locale=`*locale*'
: Specify the locale name, either a language specification of the form *ll* or a combined language and country specification of the form *ll_CC*.

'`-d` *directory*'
: Specify the base directory of classes directory hierarchy.

The class name is determined by appending the locale name to the resource name, separated with an underscore. The '`-d`' option is mandatory. The class is written under the specified directory.

8.1.5 Output file location in Tcl mode

'`-l` *locale*'
'`--locale=`*locale*'
: Specify the locale name, either a language specification of the form *ll* or a combined language and country specification of the form *ll_CC*.

'`-d` *directory*'
: Specify the base directory of '`.msg`' message catalogs.

The '`-l`' and '`-d`' options are mandatory. The '`.msg`' file is written in the specified directory.

8.1.6 Input file interpretation

'`-c`'
'`--check`' Perform all the checks implied by `--check-format`, `--check-header`, `--check-domain`.

'`--check-format`'
: Check language dependent format strings.

  If the string represents a format string used in a `printf`-like function both strings should have the same number of '`%`' format specifiers, with matching types. If the flag `c-format` or `possible-c-format` appears in the special comment ⟨#,⟩ for this entry a check is performed. For example, the check will diagnose using '`%.*s`' against '`%s`', or '`%d`' against '`%s`', or '`%d`' against '`%x`'. It can even handle positional parameters.

  Normally the `xgettext` program automatically decides whether a string is a format string or not. This algorithm is not perfect, though. It might regard a string as a format string though it is not used in a `printf`-like function and so `msgfmt` might report errors where there are none.

To solve this problem the programmer can dictate the decision to the `xgettext` program (see Section 3.5 [c-format], page 25). The translator should not consider removing the flag from the ⟨#,⟩ line. This "fix" would be reversed again as soon as `msgmerge` is called the next time.

'`--check-header`'
: Verify presence and contents of the header entry. See Section 5.2 [Header Entry], page 33, for a description of the various fields in the header entry.

'`--check-domain`'
: Check for conflicts between domain directives and the `--output-file` option

'`-C`'
'`--check-compatibility`'
: Check that GNU msgfmt behaves like X/Open msgfmt. This will give an error when attempting to use the GNU extensions.

'`--check-accelerators[=`*char*`]`'
: Check presence of keyboard accelerators for menu items. This is based on the convention used in some GUIs that a keyboard accelerator in a menu item string is designated by an immediately preceding '`&`' character. Sometimes a keyboard accelerator is also called "keyboard mnemonic". This check verifies that if the untranslated string has exactly one '`&`' character, the translated string has exactly one '`&`' as well. If this option is given with a *char* argument, this *char* should be a non-alphanumeric character and is used as keyboard acceleator mark instead of '`&`'.

'`-f`'
'`--use-fuzzy`'
: Use fuzzy entries in output. Note that using this option is usually wrong, because fuzzy messages are exactly those which have not been validated by a human translator.

8.1.7 Output details

'`-a` *number*'
'`--alignment=`*number*'
: Align strings to *number* bytes (default: 1).

'`--no-hash`'
: Don't include a hash table in the binary file. Lookup will be more expensive at run time (binary search instead of hash table lookup).

8.1.8 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

'`--statistics`'
: Print statistics about translations.

'`-v`'
'`--verbose`'
: Increase verbosity level.

8.2 Invoking the `msgunfmt` Program

`msgunfmt [`*option*`] [`*file*`] ...`

The `msgunfmt` program converts a binary message catalog to a Uniforum style .po file.

8.2.1 Operation mode

'`-j`'
'`--java`' Java mode: input is a Java `ResourceBundle` class.

'`--tcl`' Tcl mode: input is a tcl/msgcat '`.msg`' file.

8.2.2 Input file location

'*file* ...' Input .mo files.

If no input *file* is given or if it is '`-`', standard input is read.

8.2.3 Input file location in Java mode

'`-r` *resource*'
'`--resource=`*resource*'
: Specify the resource name.

'`-l` *locale*'
'`--locale=`*locale*'
: Specify the locale name, either a language specification of the form *ll* or a combined language and country specification of the form *ll_CC*.

The class name is determined by appending the locale name to the resource name, separated with an underscore. The class is located using the `CLASSPATH`.

8.2.4 Input file location in Tcl mode

'`-l` *locale*'
'`--locale=`*locale*'
: Specify the locale name, either a language specification of the form *ll* or a combined language and country specification of the form *ll_CC*.

'`-d` *directory*'
: Specify the base directory of '`.msg`' message catalogs.

The '`-l`' and '`-d`' options are mandatory. The '`.msg`' file is located in the specified directory.

8.2.5 Output file location

'`-o` *file*'
'`--output-file=`*file*'
: Write output to specified file.

The results are written to standard output if no output file is specified or if it is '`-`'.

8.2.6 Output details

'`--force-po`'
: Always write an output file even if it contains no message.

'`-i`'
'`--indent`'
: Write the .po file using indented style.

'`--strict`'
: Write out a strict Uniforum conforming PO file. Note that this Uniforum format should be avoided because it doesn't support the GNU extensions.

'`-w` *number*'
'`--width=`*number*'
: Set the output page width. Long strings in the output files will be split across multiple lines in order to ensure that each line's width (= number of screen columns) is less or equal to the given *number*.

'`--no-wrap`'
: Do not break long message lines. Message lines whose width exceeds the output page width will not be split into several lines. Only file reference lines which are wider than the output page width will be split.

'`-s`'
'`--sort-output`'
: Generate sorted output. Note that using this option makes it much harder for the translator to understand each message's context.

8.2.7 Informative output

'`-h`'
'`--help`' Display this help and exit.

'`-V`'
'`--version`'
: Output version information and exit.

'`-v`'
'`--verbose`'
: Increase verbosity level.

8.3 The Format of GNU MO Files

The format of the generated MO files is best described by a picture, which appears below.

The first two words serve the identification of the file. The magic number will always signal GNU MO files. The number is stored in the byte order of the generating machine, so the magic number really is two numbers: `0x950412de` and `0xde120495`. The second word describes the current revision of the file format. For now the revision is 0. This might change in future versions, and ensures that the readers of MO files can distinguish new formats from old ones, so that both can be handled correctly. The version is kept separate from the magic number, instead of using different magic numbers for different formats, mainly because '`/etc/magic`' is not updated often. It might be better to have magic separated from internal format version identification.

Follow a number of pointers to later tables in the file, allowing for the extension of the prefix part of MO files without having to recompile programs reading them. This might become useful for later inserting a few flag bits, indication about the charset used, new tables, or other things.

Then, at offset O and offset T in the picture, two tables of string descriptors can be found. In both tables, each string descriptor uses two 32 bits integers, one for the string length, another for the offset of the string in the MO file, counting in bytes from the start of the file. The first table contains descriptors for the original strings, and is sorted so the original strings are in increasing lexicographical order. The second table contains descriptors for the translated strings, and is parallel to the first table: to find the corresponding translation one has to access the array slot in the second array with the same index.

Having the original strings sorted enables the use of simple binary search, for when the MO file does not contain an hashing table, or for when it is not practical to use the hashing table provided in the MO file. This also has another advantage, as the empty string in a PO file GNU `gettext` is usually *translated* into some system information attached to that particular MO file, and the empty string necessarily becomes the first in both the original and translated tables, making the system information very easy to find.

The size S of the hash table can be zero. In this case, the hash table itself is not contained in the MO file. Some people might prefer this because a precomputed hashing table takes disk space, and does not win *that* much speed. The hash table contains indices to the sorted array of strings in the MO file. Conflict resolution is done by double hashing. The precise hashing algorithm used is fairly dependent on GNU `gettext` code, and is not documented here.

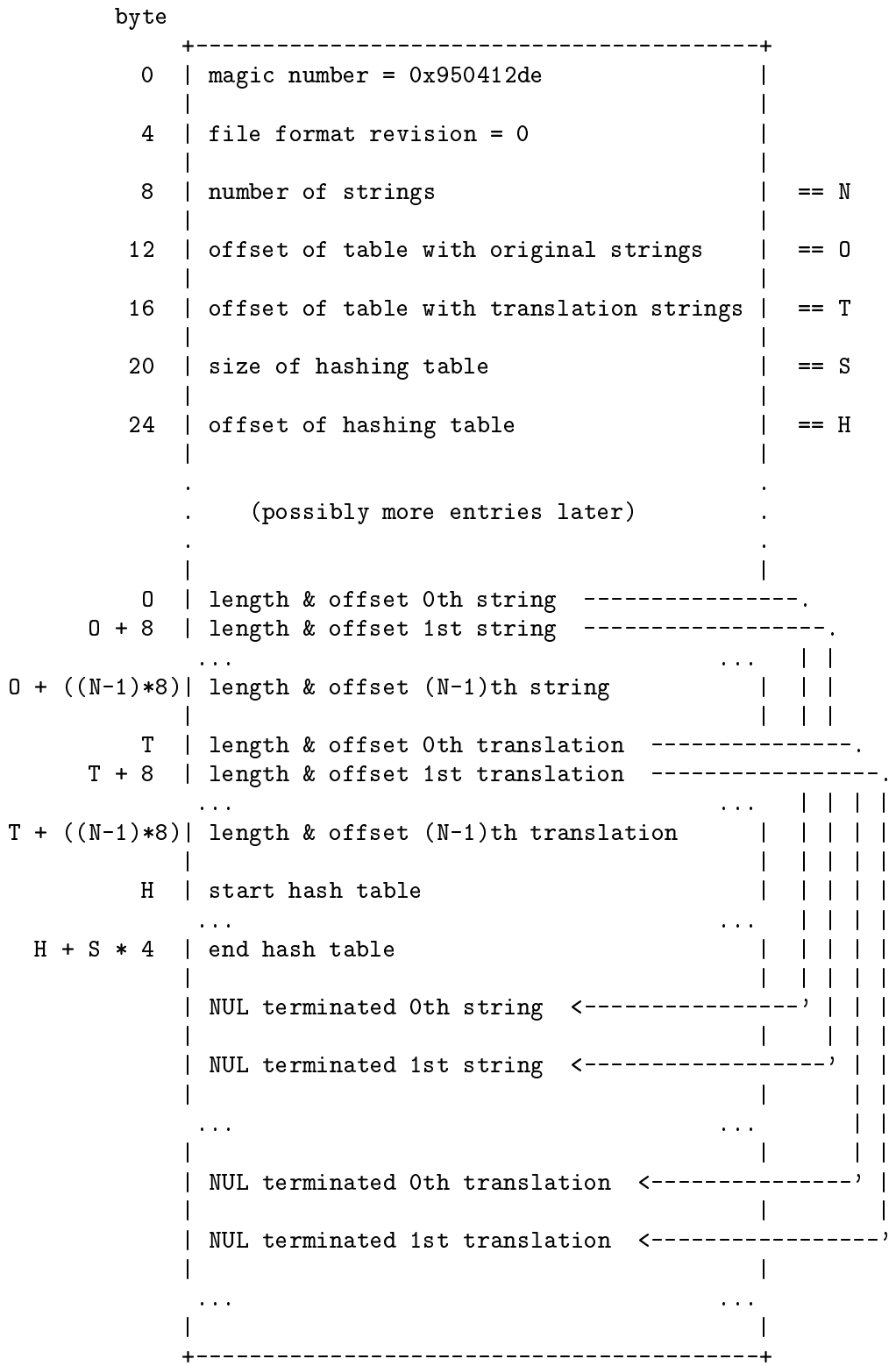
As for the strings themselves, they follow the hash file, and each is terminated with a ⟨NUL⟩, and this ⟨NUL⟩ is not counted in the length which appears in the string descriptor. The `msgfmt` program has an option selecting the alignment for MO file strings. With this option, each string is separately aligned so it starts at an offset which is a multiple of the alignment value. On some RISC machines, a correct alignment will speed things up.

Plural forms are stored by letting the plural of the original string follow the singular of the original string, separated through a ⟨NUL⟩ byte. The length which appears in the string descriptor includes both. However, only the singular of the original string takes part in the hash table lookup. The plural variants of the translation are all stored consecutively,

separated through a ⟨NUL⟩ byte. Here also, the length in the string descriptor includes all of them.

Nothing prevents a MO file from having embedded ⟨NUL⟩s in strings. However, the program interface currently used already presumes that strings are ⟨NUL⟩ terminated, so embedded ⟨NUL⟩s are somewhat useless. But the MO file format is general enough so other interfaces would be later possible, if for example, we ever want to implement wide characters right in MO files, where ⟨NUL⟩ bytes may accidently appear. (No, we don't want to have wide characters in MO files. They would make the file unnecessarily large, and the '`wchar_t`' type being platform dependent, MO files would be platform dependent as well.)

This particular issue has been strongly debated in the GNU `gettext` development forum, and it is expectable that MO file format will evolve or change over time. It is even possible that many formats may later be supported concurrently. But surely, we have to start somewhere, and the MO file format described here is a good start. Nothing is cast in concrete, and the format may later evolve fairly easily, so we should feel comfortable with the current approach.


byte
0
magic number = 0x950412de
4
file format revision = 0
8
number of strings
== N
12
offset of table with original strings
== O
16
offset of table with translation strings
== T
20
size of hashing table
== S
24
offset of hashing table
== H
(possibly more entries later)
O
length & offset 0th string
O + 8
length & offset 1st string
O + ((N-1)*8)
length & offset (N-1)th string
T
length & offset 0th translation
T + 8
length & offset 1st translation
T + ((N-1)*8)
length & offset (N-1)th translation
H
start hash table
H + S * 4
end hash table
NUL terminated 0th string
NUL terminated 1st string
NUL terminated 0th translation
NUL terminated 1st translation

9 The User's View

When GNU `gettext` will truly have reached its goal, average users should feel some kind of astonished pleasure, seeing the effect of that strange kind of magic that just makes their own native language appear everywhere on their screens. As for naive users, they would ideally have no special pleasure about it, merely taking their own language for *granted*, and becoming rather unhappy otherwise.

So, let's try to describe here how we would like the magic to operate, as we want the users' view to be the simplest, among all ways one could look at GNU `gettext`. All other software engineers: programmers, translators, maintainers, should work together in such a way that the magic becomes possible. This is a long and progressive undertaking, and information is available about the progress of the Translation Project.

When a package is distributed, there are two kinds of users: *installers* who fetch the distribution, unpack it, configure it, compile it and install it for themselves or others to use; and *end users* that call programs of the package, once these have been installed at their site. GNU `gettext` is offering magic for both installers and end users.

9.1 The Current 'ABOUT-NLS' Matrix

Languages are not equally supported in all packages using GNU `gettext`. To know if some package uses GNU `gettext`, one may check the distribution for the '`ABOUT-NLS`' information file, for some '*ll*`.po`' files, often kept together into some '`po/`' directory, or for an '`intl/`' directory. Internationalized packages have usually many '*ll*`.po`' files, where *ll* represents the language. Section 9.3 [End Users], page 79 for a complete description of the format for *ll*.

More generally, a matrix is available for showing the current state of the Translation Project, listing which packages are prepared for multi-lingual messages, and which languages are supported by each. Because this information changes often, this matrix is not kept within this GNU `gettext` manual. This information is often found in file '`ABOUT-NLS`' from various distributions, but is also as old as the distribution itself. A recent copy of this '`ABOUT-NLS`' file, containing up-to-date information, should generally be found on the Translation Project sites, and also on most GNU archive sites.

9.2 Magic for Installers

By default, packages fully using GNU `gettext`, internally, are installed in such a way that they to allow translation of messages. At *configuration* time, those packages should automatically detect whether the underlying host system already provides the GNU `gettext` functions. If not, the GNU `gettext` library should be automatically prepared and used. Installers may use special options at configuration time for changing this behavior. The command '`./configure --with-included-gettext`' bypasses system `gettext` to use the included GNU `gettext` instead, while '`./configure --disable-nls`' produces programs totally unable to translate messages.

Internationalized packages have usually many '*ll*`.po`' files. Unless translations are disabled, all those available are installed together with the package. However, the environment

variable `LINGUAS` may be set, prior to configuration, to limit the installed set. `LINGUAS` should then contain a space separated list of two-letter codes, stating which languages are allowed.

9.3 Magic for End Users

We consider here those packages using GNU `gettext` internally, and for which the installers did not disable translation at *configure* time. Then, users only have to set the `LANG` environment variable to the appropriate '*ll_CC*' combination prior to using the programs in the package. See Section 9.1 [Matrix], page 78. For example, let's presume a German site. At the shell prompt, users merely have to execute '`setenv LANG de_DE`' (in `csh`) or '`export LANG; LANG=de_DE`' (in `sh`). They could even do this from their '`.login`' or '`.profile`' file.

10 The Programmer's View

One aim of the current message catalog implementation provided by GNU `gettext` was to use the system's message catalog handling, if the installer wishes to do so. So we perhaps should first take a look at the solutions we know about. The people in the POSIX committee did not manage to agree on one of the semi-official standards which we'll describe below. In fact they couldn't agree on anything, so they decided only to include an example of an interface. The major Unix vendors are split in the usage of the two most important specifications: X/Open's catgets vs. Uniforum's gettext interface. We'll describe them both and later explain our solution of this dilemma.

10.1 About `catgets`

The `catgets` implementation is defined in the X/Open Portability Guide, Volume 3, XSI Supplementary Definitions, Chapter 5. But the process of creating this standard seemed to be too slow for some of the Unix vendors so they created their implementations on preliminary versions of the standard. Of course this leads again to problems while writing platform independent programs: even the usage of `catgets` does not guarantee a unique interface.

Another, personal comment on this that only a bunch of committee members could have made this interface. They never really tried to program using this interface. It is a fast, memory-saving implementation, an user can happily live with it. But programmers hate it (at least I and some others do...)

But we must not forget one point: after all the trouble with transfering the rights on Unix(tm) they at last came to X/Open, the very same who published this specification. This leads me to making the prediction that this interface will be in future Unix standards (e.g. Spec1170) and therefore part of all Unix implementation (implementations, which are *allowed* to wear this name).

10.1.1 The Interface

The interface to the `catgets` implementation consists of three functions which correspond to those used in file access: `catopen` to open the catalog for using, `catgets` for accessing the message tables, and `catclose` for closing after work is done. Prototypes for the functions and the needed definitions are in the `<nl_types.h>` header file.

`catopen` is used like in this:

```
nl_catd catd = catopen ("catalog_name", 0);
```

The function takes as the argument the name of the catalog. This usual refers to the name of the program or the package. The second parameter is not further specified in the standard. I don't even know whether it is implemented consistently among various systems. So the common advice is to use `0` as the value. The return value is a handle to the message catalog, equivalent to handles to file returned by `open`.

This handle is of course used in the `catgets` function which can be used like this:

```
char *translation = catgets (catd, set_no, msg_id, "original string");
```

The first parameter is this catalog descriptor. The second parameter specifies the set of messages in this catalog, in which the message described by `msg_id` is obtained. `catgets` therefore uses a three-stage addressing:

catalog name ⇒ set number ⇒ message ID ⇒ translation

The fourth argument is not used to address the translation. It is given as a default value in case when one of the addressing stages fail. One important thing to remember is that although the return type of catgets is `char *` the resulting string *must not* be changed. It should better be `const char *`, but the standard is published in 1988, one year before ANSI C.

The last of these function functions is used and behaves as expected:

```
catclose (catd);
```

After this no `catgets` call using the descriptor is legal anymore.

10.1.2 Problems with the `catgets` Interface?!

Now that this description seemed to be really easy — where are the problems we speak of? In fact the interface could be used in a reasonable way, but constructing the message catalogs is a pain. The reason for this lies in the third argument of `catgets`: the unique message ID. This has to be a numeric value for all messages in a single set. Perhaps you could imagine the problems keeping such a list while changing the source code. Add a new message here, remove one there. Of course there have been developed a lot of tools helping to organize this chaos but one as the other fails in one aspect or the other. We don't want to say that the other approach has no problems but they are far more easy to manage.

10.2 About `gettext`

The definition of the `gettext` interface comes from a Uniforum proposal and it is followed by at least one major Unix vendor (Sun) in its last developments. It is not specified in any official standard, though.

The main points about this solution is that it does not follow the method of normal file handling (open-use-close) and that it does not burden the programmer so many task, especially the unique key handling. Of course here also a unique key is needed, but this key is the message itself (how long or short it is). See Section 10.3 [Comparison], page 91 for a more detailed comparison of the two methods.

The following section contains a rather detailed description of the interface. We make it that detailed because this is the interface we chose for the GNU `gettext` Library. Programmers interested in using this library will be interested in this description.

10.2.1 The Interface

The minimal functionality an interface must have is a) to select a domain the strings are coming from (a single domain for all programs is not reasonable because its construction and maintenance is difficult, perhaps impossible) and b) to access a string in a selected domain.

This is principally the description of the `gettext` interface. It has a global domain which unqualified usages reference. Of course this domain is selectable by the user.

```
char *textdomain (const char *domain_name);
```

This provides the possibility to change or query the current status of the current global domain of the `LC_MESSAGE` category. The argument is a null-terminated string, whose characters must be legal in the use in filenames. If the *domain_name* argument is `NULL`, the function returns the current value. If no value has been set before, the name of the default domain is returned: *messages*. Please note that although the return value of `textdomain` is of type `char *` no changing is allowed. It is also important to know that no checks of the availability are made. If the name is not available you will see this by the fact that no translations are provided.

To use a domain set by `textdomain` the function

```
char *gettext (const char *msgid);
```

is to be used. This is the simplest reasonable form one can imagine. The translation of the string *msgid* is returned if it is available in the current domain. If not available the argument itself is returned. If the argument is `NULL` the result is undefined.

One things which should come into mind is that no explicit dependency to the used domain is given. The current value of the domain for the `LC_MESSAGES` locale is used. If this changes between two executions of the same `gettext` call in the program, both calls reference a different message catalog.

For the easiest case, which is normally used in internationalized packages, once at the beginning of execution a call to `textdomain` is issued, setting the domain to a unique name, normally the package name. In the following code all strings which have to be translated are filtered through the gettext function. That's all, the package speaks your language.

10.2.2 Solving Ambiguities

While this single name domain works well for most applications there might be the need to get translations from more than one domain. Of course one could switch between different domains with calls to `textdomain`, but this is really not convenient nor is it fast. A possible situation could be one case subject to discussion during this writing: all error messages of functions in the set of common used functions should go into a separate domain `error`. By this mean we would only need to translate them once. Another case are messages from a library, as these *have* to be independent of the current domain set by the application.

For this reasons there are two more functions to retrieve strings:

```
char *dgettext (const char *domain_name, const char *msgid);
char *dcgettext (const char *domain_name, const char *msgid,
                 int category);
```

Both take an additional argument at the first place, which corresponds to the argument of `textdomain`. The third argument of `dcgettext` allows to use another locale but `LC_MESSAGES`. But I really don't know where this can be useful. If the *domain_name* is `NULL` or *category* has an value beside the known ones, the result is undefined. It should also be noted that this function is not part of the second known implementation of this function family, the one found in Solaris.

A second ambiguity can arise by the fact, that perhaps more than one domain has the same name. This can be solved by specifying where the needed message catalog files can be found.

```
char *bindtextdomain (const char *domain_name,
                      const char *dir_name);
```

Calling this function binds the given domain to a file in the specified directory (how this file is determined follows below). Especially a file in the systems default place is not favored against the specified file anymore (as it would be by solely using `textdomain`). A `NULL` pointer for the *dir_name* parameter returns the binding associated with *domain_name*. If *domain_name* itself is `NULL` nothing happens and a `NULL` pointer is returned. Here again as for all the other functions is true that none of the return value must be changed!

It is important to remember that relative path names for the *dir_name* parameter can be trouble. Since the path is always computed relative to the current directory different results will be achieved when the program executes a `chdir` command. Relative paths should always be avoided to avoid dependencies and unreliabilities.

10.2.3 Locating Message Catalog Files

Because many different languages for many different packages have to be stored we need some way to add these information to file message catalog files. The way usually used in Unix environments is have this encoding in the file name. This is also done here. The directory name given in `bindtextdomain`s second argument (or the default directory), followed by the value and name of the locale and the domain name are concatenated:

dir_name/*locale*/`LC_`*category*/*domain_name*`.mo`

The default value for *dir_name* is system specific. For the GNU library, and for packages adhering to its conventions, it's:

`/usr/local/share/locale`

locale is the value of the locale whose name is this `LC_`*category*. For `gettext` and `dgettext` this `LC_`*category* is always `LC_MESSAGES`.[1] The value of the locale is determined through `setlocale (LC_`*category*`, NULL)`.[2] `dcgettext` specifies the locale category by the third argument.

10.2.4 How to specify the output character set `gettext` uses

`gettext` not only looks up a translation in a message catalog. It also converts the translation on the fly to the desired output character set. This is useful if the user is working in a different character set than the translator who created the message catalog, because it avoids distributing variants of message catalogs which differ only in the character set.

The output character set is, by default, the value of `nl_langinfo (CODESET)`, which depends on the `LC_CTYPE` part of the current locale. But programs which store strings in a locale independent way (e.g. UTF-8) can request that `gettext` and related functions return the translations in that encoding, by use of the `bind_textdomain_codeset` function.

---

[1] Some system, eg Ultrix, don't have `LC_MESSAGES`. Here we use a more or less arbitrary value for it, namely 1729, the smallest positive integer which can be represented in two different ways as the sum of two cubes.

[2] When the system does not support `setlocale` its behavior in setting the locale values is simulated by looking at the environment variables.

Note that the *msgid* argument to `gettext` is not subject to character set conversion. Also, when `gettext` does not find a translation for *msgid*, it returns *msgid* unchanged – independently of the current output character set. It is therefore recommended that all *msgid*s be US-ASCII strings.

`char *` **bind_textdomain_codeset** (`const char *`*domainname*`, const char *`*codeset*) Function

The `bind_textdomain_codeset` function can be used to specify the output character set for message catalogs for domain *domainname*. The *codeset* argument must be a valid codeset name which can be used for the `iconv_open` function, or a null pointer.

If the *codeset* parameter is the null pointer, `bind_textdomain_codeset` returns the currently selected codeset for the domain with the name *domainname*. It returns `NULL` if no codeset has yet been selected.

The `bind_textdomain_codeset` function can be used several times. If used multiple times with the same *domainname* argument, the later call overrides the settings made by the earlier one.

The `bind_textdomain_codeset` function returns a pointer to a string containing the name of the selected codeset. The string is allocated internally in the function and must not be changed by the user. If the system went out of core during the execution of `bind_textdomain_codeset`, the return value is `NULL` and the global variable *errno* is set accordingly.

10.2.5 Additional functions for plural forms

The functions of the `gettext` family described so far (and all the `catgets` functions as well) have one problem in the real world which have been neglected completely in all existing approaches. What is meant here is the handling of plural forms.

Looking through Unix source code before the time anybody thought about internationalization (and, sadly, even afterwards) one can often find code similar to the following:

```
printf ("%d file%s deleted", n, n == 1 ? "" : "s");
```

After the first complaints from people internationalizing the code people either completely avoided formulations like this or used strings like `"file(s)"`. Both look unnatural and should be avoided. First tries to solve the problem correctly looked like this:

```
if (n == 1)
  printf ("%d file deleted", n);
else
  printf ("%d files deleted", n);
```

But this does not solve the problem. It helps languages where the plural form of a noun is not simply constructed by adding an 's' but that is all. Once again people fell into the trap of believing the rules their language is using are universal. But the handling of plural forms differs widely between the language families. For example, Rafal Maszkowski `<rzm@mat.uni.torun.pl>` reports:

In Polish we use e.g. plik (file) this way:

```
1 plik
2,3,4 pliki
5-21 pliko'w
22-24 pliki
25-31 pliko'w
```

and so on (o' means 8859-2 oacute which should be rather okreska, similar to aogonek).

There are two things which can differ between languages (and even inside language families);

- The form how plural forms are built differs. This is a problem with languages which have many irregularities. German, for instance, is a drastic case. Though English and German are part of the same language family (Germanic), the almost regular forming of plural noun forms (appending an 's') is hardly found in German.
- The number of plural forms differ. This is somewhat surprising for those who only have experiences with Romanic and Germanic languages since here the number is the same (there are two).

  But other language families have only one form or many forms. More information on this in an extra section.

The consequence of this is that application writers should not try to solve the problem in their code. This would be localization since it is only usable for certain, hardcoded language environments. Instead the extended `gettext` interface should be used.

These extra functions are taking instead of the one key string two strings and a numerical argument. The idea behind this is that using the numerical argument and the first string as a key, the implementation can select using rules specified by the translator the right plural form. The two string arguments then will be used to provide a return value in case no message catalog is found (similar to the normal `gettext` behavior). In this case the rules for Germanic language is used and it is assumed that the first string argument is the singular form, the second the plural form.

This has the consequence that programs without language catalogs can display the correct strings only if the program itself is written using a Germanic language. This is a limitation but since the GNU C library (as well as the GNU `gettext` package) are written as part of the GNU package and the coding standards for the GNU project require program being written in English, this solution nevertheless fulfills its purpose.

`char * ngettext (const char *`*msgid1*`, const char *`*msgid2*`,` `unsigned long int` *n*`)` Function

The `ngettext` function is similar to the `gettext` function as it finds the message catalogs in the same way. But it takes two extra arguments. The *msgid1* parameter must contain the singular form of the string to be converted. It is also used as the key for the search in the catalog. The *msgid2* parameter is the plural form. The parameter *n* is used to determine the plural form. If no message catalog is found *msgid1* is returned if `n == 1`, otherwise `msgid2`.

An example for the use of this function is:

```
printf (ngettext ("%d file removed", "%d files removed", n), n);
```

Please note that the numeric value *n* has to be passed to the `printf` function as well. It is not sufficient to pass it only to `ngettext`.

`char *` **dngettext** (`const char *`*domain*`, const char *`*msgid1*`, const char *`*msgid2*`, unsigned long int` *n*) Function

The `dngettext` is similar to the `dgettext` function in the way the message catalog is selected. The difference is that it takes two extra parameter to provide the correct plural form. These two parameters are handled in the same way `ngettext` handles them.

`char *` **dcngettext** (`const char *`*domain*`, const char *`*msgid1*`, const char *`*msgid2*`, unsigned long int` *n*`, int` *category*) Function

The `dcngettext` is similar to the `dcgettext` function in the way the message catalog is selected. The difference is that it takes two extra parameter to provide the correct plural form. These two parameters are handled in the same way `ngettext` handles them.

Now, how do these functions solve the problem of the plural forms? Without the input of linguists (which was not available) it was not possible to determine whether there are only a few different forms in which plural forms are formed or whether the number can increase with every new supported language.

Therefore the solution implemented is to allow the translator to specify the rules of how to select the plural form. Since the formula varies with every language this is the only viable solution except for hardcoding the information in the code (which still would require the possibility of extensions to not prevent the use of new languages).

The information about the plural form selection has to be stored in the header entry of the PO file (the one with the empty `msgid` string). The plural form information looks like this:

```
Plural-Forms: nplurals=2; plural=n == 1 ? 0 : 1;
```

The `nplurals` value must be a decimal number which specifies how many different plural forms exist for this language. The string following `plural` is an expression which is using the C language syntax. Exceptions are that no negative numbers are allowed, numbers must be decimal, and the only variable allowed is `n`. This expression will be evaluated whenever one of the functions `ngettext`, `dngettext`, or `dcngettext` is called. The numeric value passed to these functions is then substituted for all uses of the variable `n` in the expression. The resulting value then must be greater or equal to zero and smaller than the value given as the value of `nplurals`.

The following rules are known at this point. The language with families are listed. But this does not necessarily mean the information can be generalized for the whole family (as can be easily seen in the table below).[3]

Only one form:

Some languages only require one single form. There is no distinction between the singular and plural form. An appropriate header entry would look like this:

[3] Additions are welcome. Send appropriate information to `bug-glibc-manual@gnu.org`.

```
Plural-Forms: nplurals=1; plural=0;
```

Languages with this property include:

Finno-Ugric family
: Hungarian

Asian family
: Japanese, Korean

Turkic/Altaic family
: Turkish

Two forms, singular used for one only
: This is the form used in most existing programs since it is what English is using. A header entry would look like this:

```
Plural-Forms: nplurals=2; plural=n != 1;
```

(Note: this uses the feature of C expressions that boolean expressions have to value zero or one.)

Languages with this property include:

Germanic family
: Danish, Dutch, English, German, Norwegian, Swedish

Finno-Ugric family
: Estonian, Finnish

Latin/Greek family
: Greek

Semitic family
: Hebrew

Romanic family
: Italian, Portuguese, Spanish

Artificial
: Esperanto

Two forms, singular used for zero and one
: Exceptional case in the language family. The header entry would be:

```
Plural-Forms: nplurals=2; plural=n>1;
```

Languages with this property include:

Romanic family
: French, Brazilian Portuguese

Three forms, special case for zero
: The header entry would be:

```
Plural-Forms: nplurals=3; plural=n%10==1 && n%100!=11 ? 0 : n != 0 ? 1 : 2;
```

Languages with this property include:

Baltic family
: Latvian

Three forms, special cases for one and two

The header entry would be:

```
Plural-Forms: nplurals=3; plural=n==1 ? 0 : n==2 ? 1 : 2;
```

Languages with this property include:

Celtic Gaeilge (Irish)

Three forms, special case for numbers ending in 1[2-9]

The header entry would look like this:

```
Plural-Forms: nplurals=3; \
    plural=n%10==1 && n%100!=11 ? 0 : \
           n%10>=2 && (n%100<10 || n%100>=20) ? 1 : 2;
```

Languages with this property include:

Baltic family
Lithuanian

Three forms, special cases for numbers ending in 1 and 2, 3, 4, except those ending in 1[1-4]

The header entry would look like this:

```
Plural-Forms: nplurals=3; \
    plural=n%10==1 && n%100!=11 ? 0 : \
           n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20) ? 1 : 2;
```

Languages with this property include:

Slavic family
Croatian, Czech, Russian, Slovak, Ukrainian

Three forms, special case for one and some numbers ending in 2, 3, or 4

The header entry would look like this:

```
Plural-Forms: nplurals=3; \
    plural=n==1 ? 0 : \
           n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20) ? 1 : 2;
```

Languages with this property include:

Slavic family
Polish

Four forms, special case for one and all numbers ending in 02, 03, or 04

The header entry would look like this:

```
Plural-Forms: nplurals=4; \
    plural=n%100==1 ? 0 : n%100==2 ? 1 : n%100==3 || n%100==4 ? 2 : 3;
```

Languages with this property include:

Slavic family
Slovenian

10.2.6 How to use `gettext` in GUI programs

One place where the `gettext` functions, if used normally, have big problems is within programs with graphical user interfaces (GUIs). The problem is that many of the strings which have to be translated are very short. They have to appear in pull-down menus which restricts the length. But strings which are not containing entire sentences or at least large fragments of a sentence may appear in more than one situation in the program but might have different translations. This is especially true for the one-word strings which are frequently used in GUI programs.

As a consequence many people say that the `gettext` approach is wrong and instead `catgets` should be used which indeed does not have this problem. But there is a very simple and powerful method to handle these kind of problems with the `gettext` functions.

As as example consider the following fictional situation. A GUI program has a menu bar with the following entries:

```
+------------+------------+--------------------------------------+
| File       | Printer    |                                      |
+------------+------------+--------------------------------------+
| Open     | | Select   |
| New      | | Open     |
+----------+ | Connect  |
             +----------+
```

To have the strings `File`, `Printer`, `Open`, `New`, `Select`, and `Connect` translated there has to be at some point in the code a call to a function of the `gettext` family. But in two places the string passed into the function would be `Open`. The translations might not be the same and therefore we are in the dilemma described above.

One solution to this problem is to artificially enlengthen the strings to make them unambiguous. But what would the program do if no translation is available? The enlengthened string is not what should be printed. So we should use a little bit modified version of the functions.

To enlengthen the strings a uniform method should be used. E.g., in the example above the strings could be chosen as

```
Menu|File
Menu|Printer
Menu|File|Open
Menu|File|New
Menu|Printer|Select
Menu|Printer|Open
Menu|Printer|Connect
```

Now all the strings are different and if now instead of `gettext` the following little wrapper function is used, everything works just fine:

```
char *
sgettext (const char *msgid)
{
  char *msgval = gettext (msgid);
  if (msgval == msgid)
```

```
      msgval = strrchr (msgid, '|') + 1;
    return msgval;
  }
```

What this little function does is to recognize the case when no translation is available. This can be done very efficiently by a pointer comparison since the return value is the input value. If there is no translation we know that the input string is in the format we used for the Menu entries and therefore contains a `|` character. We simply search for the last occurrence of this character and return a pointer to the character following it. That's it!

If one now consistently uses the enlengthened string form and replaces the `gettext` calls with calls to `sgettext` (this is normally limited to very few places in the GUI implementation) then it is possible to produce a program which can be internationalized.

The other `gettext` functions (`dgettext`, `dcgettext` and the `ngettext` equivalents) can and should have corresponding functions as well which look almost identical, except for the parameters and the call to the underlying function.

Now there is of course the question why such functions do not exist in the GNU gettext package? There are two parts of the answer to this question.

- They are easy to write and therefore can be provided by the project they are used in. This is not an answer by itself and must be seen together with the second part which is:
- There is no way the gettext package can contain a version which can work everywhere. The problem is the selection of the character to separate the prefix from the actual string in the enlenghtened string. The examples above used `|` which is a quite good choice because it resembles a notation frequently used in this context and it also is a character not often used in message strings.

  But what if the character is used in message strings? Or if the chose character is not available in the character set on the machine one compiles (e.g., `|` is not required to exist for ISO C; this is why the '`iso646.h`' file exists in ISO C programming environments).

There is only one more comment to be said. The wrapper function above requires that the translations strings are not enlengthened themselves. This is only logical. There is no need to disambiguate the strings (since they are never used as keys for a search) and one also saves quite some memory and disk space by doing this.

10.2.7 Optimization of the *gettext functions

At this point of the discussion we should talk about an advantage of the GNU `gettext` implementation. Some readers might have pointed out that an internationalized program might have a poor performance if some string has to be translated in an inner loop. While this is unavoidable when the string varies from one run of the loop to the other it is simply a waste of time when the string is always the same. Take the following example:

```
{
  while (...)
    {
      puts (gettext ("Hello world"));
    }
}
```

When the locale selection does not change between two runs the resulting string is always the same. One way to use this is:

```
{
  str = gettext ("Hello world");
  while (...)
    {
      puts (str);
    }
}
```

But this solution is not usable in all situation (e.g. when the locale selection changes) nor does it lead to legible code.

For this reason, GNU `gettext` caches previous translation results. When the same translation is requested twice, with no new message catalogs being loaded in between, `gettext` will, the second time, find the result through a single cache lookup.

10.3 Comparing the Two Interfaces

The following discussion is perhaps a little bit colored. As said above we implemented GNU `gettext` following the Uniforum proposal and this surely has its reasons. But it should show how we came to this decision.

First we take a look at the developing process. When we write an application using NLS provided by `gettext` we proceed as always. Only when we come to a string which might be seen by the users and thus has to be translated we use `gettext("...")` instead of `"..."`. At the beginning of each source file (or in a central header file) we define

```
#define gettext(String) (String)
```

Even this definition can be avoided when the system supports the `gettext` function in its C library. When we compile this code the result is the same as if no NLS code is used. When you take a look at the GNU `gettext` code you will see that we use `_("...")` instead of `gettext("...")`. This reduces the number of additional characters per translatable string to *3* (in words: three).

When now a production version of the program is needed we simply replace the definition

```
#define _(String) (String)
```

by

```
#include <libintl.h>
#define _(String) gettext (String)
```

Additionally we run the program '`xgettext`' on all source code file which contain translatable strings and that's it: we have a running program which does not depend on translations to be available, but which can use any that becomes available.

The same procedure can be done for the `gettext_noop` invocations (see Section 3.6 [Special cases], page 26). One usually defines `gettext_noop` as a no-op macro. So you should consider the following code for your project:

```
#define gettext_noop(String) String
#define N_(String) gettext_noop (String)
```

`N_` is a short form similar to `_`. The '`Makefile`' in the '`po/`' directory of GNU `gettext` knows by default both of the mentioned short forms so you are invited to follow this proposal for your own ease.

Now to `catgets`. The main problem is the work for the programmer. Every time he comes to a translatable string he has to define a number (or a symbolic constant) which has also be defined in the message catalog file. He also has to take care for duplicate entries, duplicate message IDs etc. If he wants to have the same quality in the message catalog as the GNU `gettext` program provides he also has to put the descriptive comments for the strings and the location in all source code files in the message catalog. This is nearly a Mission: Impossible.

But there are also some points people might call advantages speaking for `catgets`. If you have a single word in a string and this string is used in different contexts it is likely that in one or the other language the word has different translations. Example:

```
printf ("%s: %d", gettext ("number"), number_of_errors)

printf ("you should see %d %s", number_count,
        number_count == 1 ? gettext ("number") : gettext ("numbers"))
```

Here we have to translate two times the string `"number"`. Even if you do not speak a language beside English it might be possible to recognize that the two words have a different meaning. In German the first appearance has to be translated to `"Anzahl"` and the second to `"Zahl"`.

Now you can say that this example is really esoteric. And you are right! This is exactly how we felt about this problem and decide that it does not weight that much. The solution for the above problem could be very easy:

```
printf ("%s %d", gettext ("number:"), number_of_errors)

printf (number_count == 1 ? gettext ("you should see %d number")
                          : gettext ("you should see %d numbers"),
        number_count)
```

We believe that we can solve all conflicts with this method. If it is difficult one can also consider changing one of the conflicting string a little bit. But it is not impossible to overcome.

`catgets` allows same original entry to have different translations, but `gettext` has another, scalable approach for solving ambiguities of this kind: See Section 10.2.2 [Ambiguities], page 82.

10.4 Using libintl.a in own programs

Starting with version 0.9.4 the library `libintl.h` should be self-contained. I.e., you can use it in your own programs without providing additional functions. The '`Makefile`' will put the header and the library in directories selected using the `$(prefix)`.

10.5 Being a gettext grok

To fully exploit the functionality of the GNU `gettext` library it is surely helpful to read the source code. But for those who don't want to spend that much time in reading the (sometimes complicated) code here is a list comments:

- Changing the language at runtime

  For interactive programs it might be useful to offer a selection of the used language at runtime. To understand how to do this one need to know how the used language is determined while executing the `gettext` function. The method which is presented here only works correctly with the GNU implementation of the `gettext` functions.

  In the function `dcgettext` at every call the current setting of the highest priority environment variable is determined and used. Highest priority means here the following list with decreasing priority:

  1. `LANGUAGE`
  2. `LC_ALL`
  3. `LC_xxx`, according to selected locale
  4. `LANG`

  Afterwards the path is constructed using the found value and the translation file is loaded if available.

  What is now when the value for, say, `LANGUAGE` changes. According to the process explained above the new value of this variable is found as soon as the `dcgettext` function is called. But this also means the (perhaps) different message catalog file is loaded. In other words: the used language is changed.

  But there is one little hook. The code for gcc-2.7.0 and up provides some optimization. This optimization normally prevents the calling of the `dcgettext` function as long as no new catalog is loaded. But if `dcgettext` is not called the program also cannot find the `LANGUAGE` variable be changed (see Section 10.2.7 [Optimized gettext], page 90). A solution for this is very easy. Include the following code in the language switching function.

```
/* Change language.  */
setenv ("LANGUAGE", "fr", 1);

/* Make change known.  */
{
  extern int  _nl_msg_cat_cntr;
  ++_nl_msg_cat_cntr;
}
```

  The variable `_nl_msg_cat_cntr` is defined in '`loadmsgcat.c`'. You don't need to know what this is for. But it can be used to detect whether a `gettext` implementation is GNU gettext and not non-GNU system's native gettext implementation.

10.6 Temporary Notes for the Programmers Chapter

10.6.1 Temporary - Two Possible Implementations

There are two competing methods for language independent messages: the X/Open `catgets` method, and the Uniforum `gettext` method. The `catgets` method indexes messages by integers; the `gettext` method indexes them by their English translations. The `catgets` method has been around longer and is supported by more vendors. The `gettext` method is supported by Sun, and it has been heard that the COSE multi-vendor initiative is supporting it. Neither method is a POSIX standard; the POSIX.1 committee had a lot of disagreement in this area.

Neither one is in the POSIX standard. There was much disagreement in the POSIX.1 committee about using the `gettext` routines vs. `catgets` (XPG). In the end the committee couldn't agree on anything, so no messaging system was included as part of the standard. I believe the informative annex of the standard includes the XPG3 messaging interfaces, "...as an example of a messaging system that has been implemented..."

They were very careful not to say anywhere that you should use one set of interfaces over the other. For more on this topic please see the Programming for Internationalization FAQ.

10.6.2 Temporary - About `catgets`

There have been a few discussions of late on the use of `catgets` as a base. I think it important to present both sides of the argument and hence am opting to play devil's advocate for a little bit.

I'll not deny the fact that `catgets` could have been designed a lot better. It currently has quite a number of limitations and these have already been pointed out.

However there is a great deal to be said for consistency and standardization. A common recurring problem when writing Unix software is the myriad portability problems across Unix platforms. It seems as if every Unix vendor had a look at the operating system and found parts they could improve upon. Undoubtedly, these modifications are probably innovative and solve real problems. However, software developers have a hard time keeping up with all these changes across so many platforms.

And this has prompted the Unix vendors to begin to standardize their systems. Hence the impetus for Spec1170. Every major Unix vendor has committed to supporting this standard and every Unix software developer waits with glee the day they can write software to this standard and simply recompile (without having to use autoconf) across different platforms.

As I understand it, Spec1170 is roughly based upon version 4 of the X/Open Portability Guidelines (XPG4). Because `catgets` and friends are defined in XPG4, I'm led to believe that `catgets` is a part of Spec1170 and hence will become a standardized component of all Unix systems.

10.6.3 Temporary - Why a single implementation

Now it seems kind of wasteful to me to have two different systems installed for accessing message catalogs. If we do want to remedy `catgets` deficiencies why don't we try to expand `catgets` (in a compatible manner) rather than implement an entirely new system. Otherwise, we'll end up with two message catalog access systems installed with an operating system - one set of routines for packages using GNU `gettext` for their internationalization, and another set of routines (catgets) for all other software. Bloated?

Supposing another catalog access system is implemented. Which do we recommend? At least for Linux, we need to attract as many software developers as possible. Hence we need to make it as easy for them to port their software as possible. Which means supporting `catgets`. We will be implementing the `libintl` code within our `libc`, but does this mean we also have to incorporate another message catalog access scheme within our `libc` as well? And what about people who are going to be using the `libintl` + non-`catgets` routines. When they port their software to other platforms, they're now going to have to include the front-end (`libintl`) code plus the back-end code (the non-`catgets` access routines) with their software instead of just including the `libintl` code with their software.

Message catalog support is however only the tip of the iceberg. What about the data for the other locale categories. They also have a number of deficiencies. Are we going to abandon them as well and develop another duplicate set of routines (should `libintl` expand beyond message catalog support)?

Like many parts of Unix that can be improved upon, we're stuck with balancing compatibility with the past with useful improvements and innovations for the future.

10.6.4 Temporary - Notes

X/Open agreed very late on the standard form so that many implementations differ from the final form. Both of my system (old Linux catgets and Ultrix-4) have a strange variation.

OK. After incorporating the last changes I have to spend some time on making the GNU/Linux `libc gettext` functions. So in future Solaris is not the only system having `gettext`.

11 The Translator's View

11.1 Introduction 0

Free software is going international! The Translation Project is a way to get maintainers, translators and users all together, so free software will gradually become able to speak many native languages.

The GNU `gettext` tool set contains *everything* maintainers need for internationalizing their packages for messages. It also contains quite useful tools for helping translators at localizing messages to their native language, once a package has already been internationalized.

To achieve the Translation Project, we need many interested people who like their own language and write it well, and who are also able to synergize with other translators speaking the same language. If you'd like to volunteer to *work* at translating messages, please send mail to your translating team.

Each team has its own mailing list, courtesy of Linux International. You may reach your translating team at the address '*ll*`@li.org`', replacing *ll* by the two-letter ISO 639 code for your language. Language codes are *not* the same as country codes given in ISO 3166. The following translating teams exist:

> Chinese `zh`, Czech `cs`, Danish `da`, Dutch `nl`, Esperanto `eo`, Finnish `fi`, French `fr`, Irish `ga`, German `de`, Greek `el`, Italian `it`, Japanese `ja`, Indonesian `in`, Norwegian `no`, Polish `pl`, Portuguese `pt`, Russian `ru`, Spanish `es`, Swedish `sv` and Turkish `tr`.

For example, you may reach the Chinese translating team by writing to '`zh@li.org`'. When you become a member of the translating team for your own language, you may subscribe to its list. For example, Swedish people can send a message to '`sv-request@li.org`', having this message body:

```
subscribe
```

Keep in mind that team members should be interested in *working* at translations, or at solving translational difficulties, rather than merely lurking around. If your team does not exist yet and you want to start one, please write to '`translation@iro.umontreal.ca`'; you will then reach the coordinator for all translator teams.

A handful of GNU packages have already been adapted and provided with message translations for several languages. Translation teams have begun to organize, using these packages as a starting point. But there are many more packages and many languages for which we have no volunteer translators. If you would like to volunteer to work at translating messages, please send mail to '`translation@iro.umontreal.ca`' indicating what language(s) you can work on.

11.2 Introduction 1

This is now official, GNU is going international! Here is the announcement submitted for the January 1995 GNU Bulletin:

> A handful of GNU packages have already been adapted and provided with message translations for several languages. Translation teams have begun to organize, using these packages as a starting point. But there are many more packages and many languages for which we have no volunteer translators. If you'd like to volunteer to work at translating messages, please send mail to '`translation@iro.umontreal.ca`' indicating what language(s) you can work on.

This document should answer many questions for those who are curious about the process or would like to contribute. Please at least skim over it, hoping to cut down a little of the high volume of e-mail generated by this collective effort towards internationalization of free software.

Most free programming which is widely shared is done in English, and currently, English is used as the main communicating language between national communities collaborating to free software. This very document is written in English. This will not change in the foreseeable future.

However, there is a strong appetite from national communities for having more software able to write using national language and habits, and there is an on-going effort to modify free software in such a way that it becomes able to do so. The experiments driven so far raised an enthusiastic response from pretesters, so we believe that internationalization of free software is dedicated to succeed.

For suggestion clarifications, additions or corrections to this document, please e-mail to '`translation@iro.umontreal.ca`'.

11.3 Discussions

Facing this internationalization effort, a few users expressed their concerns. Some of these doubts are presented and discussed, here.

- Smaller groups

  Some languages are not spoken by a very large number of people, so people speaking them sometimes consider that there may not be all that much demand such versions of free software packages. Moreover, many people being *into computers*, in some countries, generally seem to prefer English versions of their software.

  On the other end, people might enjoy their own language a lot, and be very motivated at providing to themselves the pleasure of having their beloved free software speaking their mother tongue. They do themselves a personal favor, and do not pay that much attention to the number of people beneficiating of their work.

- Misinterpretation

  Other users are shy to push forward their own language, seeing in this some kind of misplaced propaganda. Someone thought there must be some users of the language over the networks pestering other people with it.

  But any spoken language is worth localization, because there are people behind the language for whom the language is important and dear to their hearts.

- Odd translations

  The biggest problem is to find the right translations so that everybody can understand the messages. Translations are usually a little odd. Some people get used to English, to

the extent they may find translations into their own language "rather pushy, obnoxious and sometimes even hilarious." As a French speaking man, I have the experience of those instruction manuals for goods, so poorly translated in French in Korea or Taiwan...

The fact is that we sometimes have to create a kind of national computer culture, and this is not easy without the collaboration of many people liking their mother tongue. This is why translations are better achieved by people knowing and loving their own language, and ready to work together at improving the results they obtain.

- Dependencies over the GPL or LGPL

  Some people wonder if using GNU `gettext` necessarily brings their package under the protective wing of the GNU General Public License or the GNU Library General Public License, when they do not want to make their program free, or want other kinds of freedom. The simplest answer is "normally not".

  The GNU `gettext` library, i.e. the contents of `libintl`, is covered by the GNU Library General Public License. The rest of the GNU `gettext` package is covered by the GNU General Public License.

  The mere marking of localizable strings in a package, or conditional inclusion of a few lines for initialization, is not really including GPL'ed or LGPL'ed code. However, since the localization routines in `libintl` are under the LGPL, the LGPL needs to be considered. It gives the right to distribute the complete unmodified source of `libintl` even with non-free programs. It also gives the right to use `libintl` as a shared library, even for non-free programs. But it gives the right to use `libintl` as a static library or to incorporate `libintl` into another library only to free software.

11.4 Organization

On a larger scale, the true solution would be to organize some kind of fairly precise set up in which volunteers could participate. I gave some thought to this idea lately, and realize there will be some touchy points. I thought of writing to Richard Stallman to launch such a project, but feel it might be good to shake out the ideas between ourselves first. Most probably that Linux International has some experience in the field already, or would like to orchestrate the volunteer work, maybe. Food for thought, in any case!

I guess we have to setup something early, somehow, that will help many possible contributors of the same language to interlock and avoid work duplication, and further be put in contact for solving together problems particular to their tongue (in most languages, there are many difficulties peculiar to translating technical English). My Swedish contributor acknowledged these difficulties, and I'm well aware of them for French.

This is surely not a technical issue, but we should manage so the effort of locale contributors be maximally useful, despite the national team layer interface between contributors and maintainers.

The Translation Project needs some setup for coordinating language coordinators. Localizing evolving programs will surely become a permanent and continuous activity in the free software community, once well started. The setup should be minimally completed and tested before GNU `gettext` becomes an official reality. The e-mail address

'`translation@iro.umontreal.ca`' has been setup for receiving offers from volunteers and general e-mail on these topics. This address reaches the Translation Project coordinator.

11.4.1 Central Coordination

I also think GNU will need sooner than it thinks, that someone setup a way to organize and coordinate these groups. Some kind of group of groups. My opinion is that it would be good that GNU delegates this task to a small group of collaborating volunteers, shortly. Perhaps in '`gnu.announce`' a list of this national committee's can be published.

My role as coordinator would simply be to refer to Ulrich any German speaking volunteer interested to localization of free software packages, and maybe helping national groups to initially organize, while maintaining national registries for until national groups are ready to take over. In fact, the coordinator should ease volunteers to get in contact with one another for creating national teams, which should then select one coordinator per language, or country (regionalized language). If well done, the coordination should be useful without being an overwhelming task, the time to put delegations in place.

11.4.2 National Teams

I suggest we look for volunteer coordinators/editors for individual languages. These people will scan contributions of translation files for various programs, for their own languages, and will ensure high and uniform standards of diction.

From my current experience with other people in these days, those who provide localizations are very enthusiastic about the process, and are more interested in the localization process than in the program they localize, and want to do many programs, not just one. This seems to confirm that having a coordinator/editor for each language is a good idea.

We need to choose someone who is good at writing clear and concise prose in the language in question. That is hard—we can't check it ourselves. So we need to ask a few people to judge each others' writing and select the one who is best.

I announce my prerelease to a few dozen people, and you would not believe all the discussions it generated already. I shudder to think what will happen when this will be launched, for true, officially, world wide. Who am I to arbitrate between two Czekolsovak users contradicting each other, for example?

I assume that your German is not much better than my French so that I would not be able to judge about these formulations. What I would suggest is that for each language there is a group for people who maintain the PO files and judge about changes. I suspect there will be cultural differences between how such groups of people will behave. Some will have relaxed ways, reach consensus easily, and have anyone of the group relate to the maintainers, while others will fight to death, organize heavy administrations up to national standards, and use strict channels.

The German team is putting out a good example. Right now, they are maybe half a dozen people revising translations of each other and discussing the linguistic issues. I do not even have all the names. Ulrich Drepper is taking care of coordinating the German team. He subscribed to all my pretest lists, so I do not even have to warn him specifically of incoming releases.

I'm sure, that is a good idea to get teams for each language working on translations. That will make the translations better and more consistent.

11.4.2.1 Sub-Cultures

Taking French for example, there are a few sub-cultures around computers which developed diverging vocabularies. Picking volunteers here and there without addressing this problem in an organized way, soon in the project, might produce a distasteful mix of internationalized programs, and possibly trigger endless quarrels among those who really care.

Keeping some kind of unity in the way French localization of internationalized programs is achieved is a difficult (and delicate) job. Knowing the latin character of French people (:-), if we take this the wrong way, we could end up nowhere, or spoil a lot of energies. Maybe we should begin to address this problem seriously *before* GNU `gettext` become officially published. And I suspect that this means soon!

11.4.2.2 Organizational Ideas

I expect the next big changes after the official release. Please note that I use the German translation of the short GPL message. We need to set a few good examples before the localization goes out for true in the free software community. Here are a few points to discuss:

- Each group should have one FTP server (at least one master).
- The files on the server should reflect the latest version (of course!) and it should also contain a RCS directory with the corresponding archives (I don't have this now).
- There should also be a ChangeLog file (this is more useful than the RCS archive but can be generated automatically from the later by Emacs).
- A *core group* should judge about questionable changes (for now this group consists solely by me but I ask some others occasionally; this also seems to work).

11.4.3 Mailing Lists

If we get any inquiries about GNU `gettext`, send them on to:

`'translation@iro.umontreal.ca'`

The '`*-pretest`' lists are quite useful to me, maybe the idea could be generalized to many GNU, and non-GNU packages. But each maintainer his/her way!

François, we have a mechanism in place here at '`gnu.ai.mit.edu`' to track teams, support mailing lists for them and log members. We have a slight preference that you use it. If this is OK with you, I can get you clued in.

Things are changing! A few years ago, when Daniel Fekete and I asked for a mailing list for GNU localization, nested at the FSF, we were politely invited to organize it anywhere else, and so did we. For communicating with my pretesters, I later made a handful of mailing lists located at iro.umontreal.ca and administrated by `majordomo`. These lists have been *very* dependable so far. . .

I suspect that the German team will organize itself a mailing list located in Germany, and so forth for other countries. But before they organize for true, it could surely be useful

to offer mailing lists located at the FSF to each national team. So yes, please explain me how I should proceed to create and handle them.

We should create temporary mailing lists, one per country, to help people organize. Temporary, because once regrouped and structured, it would be fair the volunteers from country bring back *their* list in there and manage it as they want. My feeling is that, in the long run, each team should run its own list, from within their country. There also should be some central list to which all teams could subscribe as they see fit, as long as each team is represented in it.

11.5 Information Flow

There will surely be some discussion about this messages after the packages are finally released. If people now send you some proposals for better messages, how do you proceed? Jim, please note that right now, as I put forward nearly a dozen of localizable programs, I receive both the translations and the coordination concerns about them.

If I put one of my things to pretest, Ulrich receives the announcement and passes it on to the German team, who make last minute revisions. Then he submits the translation files to me *as the maintainer*. For free packages I do not maintain, I would not even hear about it. This scheme could be made to work for the whole Translation Project, I think. For security reasons, maybe Ulrich (national coordinators, in fact) should update central registry kept at the Translation Project (Jim, me, or Len's recruits) once in a while.

In December/January, I was aggressively ready to internationalize all of GNU, giving myself the duty of one small GNU package per week or so, taking many weeks or months for bigger packages. But it does not work this way. I first did all the things I'm responsible for. I've nothing against some missionary work on other maintainers, but I'm also loosing a lot of energy over it—same debates over again.

And when the first localized packages are released we'll get a lot of responses about ugly translations :-). Surely, and we need to have beforehand a fairly good idea about how to handle the information flow between the national teams and the package maintainers.

Please start saving somewhere a quick history of each PO file. I know for sure that the file format will change, allowing for comments. It would be nice that each file has a kind of log, and references for those who want to submit comments or gripes, or otherwise contribute. I sent a proposal for a fast and flexible format, but it is not receiving acceptance yet by the GNU deciders. I'll tell you when I have more information about this.

12 The Maintainer's View

The maintainer of a package has many responsibilities. One of them is ensuring that the package will install easily on many platforms, and that the magic we described earlier (see Chapter 9 [Users], page 78) will work for installers and end users.

Of course, there are many possible ways by which GNU `gettext` might be integrated in a distribution, and this chapter does not cover them in all generality. Instead, it details one possible approach which is especially adequate for many free software distributions following GNU standards, or even better, Gnits standards, because GNU `gettext` is purposely for helping the internationalization of the whole GNU project, and as many other good free packages as possible. So, the maintainer's view presented here presumes that the package already has a '`configure.in`' file and uses GNU Autoconf.

Nevertheless, GNU `gettext` may surely be useful for free packages not following GNU standards and conventions, but the maintainers of such packages might have to show imagination and initiative in organizing their distributions so `gettext` work for them in all situations. There are surely many, out there.

Even if `gettext` methods are now stabilizing, slight adjustments might be needed between successive `gettext` versions, so you should ideally revise this chapter in subsequent releases, looking for changes.

12.1 Flat or Non-Flat Directory Structures

Some free software packages are distributed as `tar` files which unpack in a single directory, these are said to be *flat* distributions. Other free software packages have a one level hierarchy of subdirectories, using for example a subdirectory named '`doc/`' for the Texinfo manual and man pages, another called '`lib/`' for holding functions meant to replace or complement C libraries, and a subdirectory '`src/`' for holding the proper sources for the package. These other distributions are said to be *non-flat*.

We cannot say much about flat distributions. A flat directory structure has the disadvantage of increasing the difficulty of updating to a new version of GNU `gettext`. Also, if you have many PO files, this could somewhat pollute your single directory. Also, GNU `gettext`'s libintl sources consist of C sources, shell scripts, `sed` scripts and complicated Makefile rules, which don't fit well into an existing flat structure. For these reasons, we recommend to use non-flat approach in this case as well.

Maybe because GNU `gettext` itself has a non-flat structure, we have more experience with this approach, and this is what will be described in the remaining of this chapter. Some maintainers might use this as an opportunity to unflatten their package structure.

12.2 Prerequisite Works

There are some works which are required for using GNU `gettext` in one of your package. These works have some kind of generality that escape the point by point descriptions used in the remainder of this chapter. So, we describe them here.

- Before attempting to use `gettextize` you should install some other packages first. Ensure that recent versions of GNU `m4`, GNU Autoconf and GNU `gettext` are already

installed at your site, and if not, proceed to do this first. If you get to install these things, beware that GNU `m4` must be fully installed before GNU Autoconf is even *configured*.

To further ease the task of a package maintainer the `automake` package was designed and implemented. GNU `gettext` now uses this tool and the '`Makefile`'s in the '`intl/`' and '`po/`' therefore know about all the goals necessary for using `automake` and '`libintl`' in one project.

Those four packages are only needed by you, as a maintainer; the installers of your own package and end users do not really need any of GNU `m4`, GNU Autoconf, GNU `gettext`, or GNU `automake` for successfully installing and running your package, with messages properly translated. But this is not completely true if you provide internationalized shell scripts within your own package: GNU `gettext` shall then be installed at the user site if the end users want to see the translation of shell script messages.

- Your package should use Autoconf and have a '`configure.in`' or '`configure.ac`' file. If it does not, you have to learn how. The Autoconf documentation is quite well written, it is a good idea that you print it and get familiar with it.
- Your C sources should have already been modified according to instructions given earlier in this manual. See Chapter 3 [Sources], page 18.
- Your '`po/`' directory should receive all PO files submitted to you by the translator teams, each having '*ll*`.po`' as a name. This is not usually easy to get translation work done before your package gets internationalized and available! Since the cycle has to start somewhere, the easiest for the maintainer is to start with absolutely no PO files, and wait until various translator teams get interested in your package, and submit PO files.

It is worth adding here a few words about how the maintainer should ideally behave with PO files submissions. As a maintainer, your role is to authenticate the origin of the submission as being the representative of the appropriate translating teams of the Translation Project (forward the submission to '`translation@iro.umontreal.ca`' in case of doubt), to ensure that the PO file format is not severely broken and does not prevent successful installation, and for the rest, to merely put these PO files in '`po/`' for distribution.

As a maintainer, you do not have to take on your shoulders the responsibility of checking if the translations are adequate or complete, and should avoid diving into linguistic matters. Translation teams drive themselves and are fully responsible of their linguistic choices for the Translation Project. Keep in mind that translator teams are *not* driven by maintainers. You can help by carefully redirecting all communications and reports from users about linguistic matters to the appropriate translation team, or explain users how to reach or join their team. The simplest might be to send them the '`ABOUT-NLS`' file.

Maintainers should *never ever* apply PO file bug reports themselves, short-cutting translation teams. If some translator has difficulty to get some of her points through her team, it should not be an option for her to directly negotiate translations with maintainers. Teams ought to settle their problems themselves, if any. If you, as a maintainer, ever think there is a real problem with a team, please never try to *solve* a team's problem on your own.

12.3 Invoking the `gettextize` Program

Some files are consistently and identically needed in every package internationalized through GNU `gettext`. As a matter of convenience, the `gettextize` program puts all these files right in your package. This program has the following synopsis:

`gettextize [` *option*`... ] [` *directory* `]`

and accepts the following options:

'`-c`'
'`--copy`' Copy the needed files instead of making symbolic links. Using links would allow the package to always use the latest `gettext` code available on the system, but it might disturb some mechanism the maintainer is used to apply to the sources. Because running `gettextize` is easy there shouldn't be problems with using copies.

'`-f`'
'`--force`' Force replacement of files which already exist.

'`--intl`' Install the libintl sources in a subdirectory named '`intl/`'. This libintl will be used to provide internationalization on systems that don't have GNU libintl installed. If this option is omitted, the call to `AM_GNU_GETTEXT` in '`configure.in`' should read: '`AM_GNU_GETTEXT([external])`', and internationalization will not be enabled on systems lacking GNU gettext.

'`--no-changelog`'
Don't update or create ChangeLog files. By default, `gettextize` logs all changes (file additions, modifications ans removals) in a file called '`ChangeLog`' in each affected directory.

'`--help`' Display this help and exit.

'`--version`'
Output version information and exit.

If *directory* is given, this is the top level directory of a package to prepare for using GNU `gettext`. If not given, it is assumed that the current directory is the top level directory of such a package.

The program `gettextize` provides the following files. However, no existing file will be replaced unless the option `--force` (`-f`) is specified.

1. The '`ABOUT-NLS`' file is copied in the main directory of your package, the one being at the top level. This file gives the main indications about how to install and use the Native Language Support features of your program. You might elect to use a more recent copy of this '`ABOUT-NLS`' file than the one provided through `gettextize`, if you have one handy. You may also fetch a more recent copy of file '`ABOUT-NLS`' from Translation Project sites, and from most GNU archive sites.
2. A '`po/`' directory is created for eventually holding all translation files, but initially only containing the file '`po/Makefile.in.in`' from the GNU `gettext` distribution. (beware the double '`.in`' in the file name) and a few auxiliary files. If the '`po/`' directory already exists, it will be preserved along with the files it contains, and only '`Makefile.in.in`' and the auxiliary files will be overwritten.

3. Only if '`--intl`' has been specified: A '`intl/`' directory is created and filled with most of the files originally in the '`intl/`' directory of the GNU `gettext` distribution. Also, if option `--force` (`-f`) is given, the '`intl/`' directory is emptied first.
4. The '`config.rpath`' file is copied into the directory containing configuration support files. It is needed by the `AM_GNU_GETTEXT` autoconf macro.

If your site support symbolic links, `gettextize` will not actually copy the files into your package, but establish symbolic links instead. This avoids duplicating the disk space needed in all packages. Merely using the '`-h`' option while creating the `tar` archive of your distribution will resolve each link by an actual copy in the distribution archive. So, to insist, you really should use '`-h`' option with `tar` within your `dist` goal of your main '`Makefile.in`'.

It is interesting to understand that most new files for supporting GNU `gettext` facilities in one package go in '`intl/`' and '`po/`' subdirectories. One distinction between these two directories is that '`intl/`' is meant to be completely identical in all packages using GNU `gettext`, while all newly created files, which have to be different, go into '`po/`'. There is a common '`Makefile.in.in`' in '`po/`', because the '`po/`' directory needs its own '`Makefile`', and it has been designed so it can be identical in all packages.

12.4 Files You Must Create or Alter

Besides files which are automatically added through `gettextize`, there are many files needing revision for properly interacting with GNU `gettext`. If you are closely following GNU standards for Makefile engineering and auto-configuration, the adaptations should be easier to achieve. Here is a point by point description of the changes needed in each.

So, here comes a list of files, each one followed by a description of all alterations it needs. Many examples are taken out from the GNU `gettext` 0.11.2 distribution itself, or from the GNU `hello` distribution (`http://www.franken.de/users/gnu/ke/hello` or `http://www.gnu.franken.de/ke/hello/`) You may indeed refer to the source code of the GNU `gettext` and GNU `hello` packages, as they are intended to be good examples for using GNU gettext functionality.

12.4.1 '`POTFILES.in`' in '`po/`'

The '`po/`' directory should receive a file named '`POTFILES.in`'. This file tells which files, among all program sources, have marked strings needing translation. Here is an example of such a file:

```
# List of source files containing translatable strings.
# Copyright (C) 1995 Free Software Foundation, Inc.

# Common library files
lib/error.c
lib/getopt.c
lib/xmalloc.c

# Package source files
src/gettext.c
src/msgfmt.c
src/xgettext.c
```

Hash-marked comments and white lines are ignored. All other lines list those source files containing strings marked for translation (see Section 3.3 [Mark Keywords], page 21), in a notation relative to the top level of your whole distribution, rather than the location of the '`POTFILES.in`' file itself.

When a C file is automatically generated by a tool, like `flex` or `bison`, that doesn't introduce translatable strings by itself, it is recommended to list in '`po/POTFILES.in`' the real source file (ending in '`.l`' in the case of `flex`, or in '`.y`' in the case of `bison`), not the generated C file.

12.4.2 'LINGUAS' in 'po/'

The '`po/`' directory should also receive a file named '`LINGUAS`'. This file contains the list of available translations. It is a whitespace separated list. Hash-marked comments and white lines are ignored. Here is an example file:

```
# Set of available languages.
de fr
```

This example means that German and French PO files are available, so that these languages are currently supported by your package. If you want to further restrict, at installation time, the set of installed languages, this should not be done by modifying the '`LINGUAS`' file, but rather by using the `LINGUAS` environment variable (see Section 9.2 [Installers], page 78).

12.4.3 'Makefile' pieces in 'po/'

The '`po/`' directory also has a file named '`Makevars`'. It can be left unmodified if your package has a single message domain and, accordingly, a single '`po/`' directory. Only packages which have multiple '`po/`' directories at different locations need to adjust the three variables defined in '`Makevars`'.

'`po/Makevars`' gets inserted into the '`po/Makefile`' when the latter is created. At the same time, all files called '`Rules-*`' in the '`po/`' directory get appended to the '`po/Makefile`'. They present an opportunity to add rules for special PO files to the Makefile, without needing to mess with '`po/Makefile.in.in`'.

GNU gettext comes with a '`Rules-quot`' file, containing rules for building catalogs '`en@quot.po`' and '`en@boldquot.po`'. The effect of '`en@quot.po`' is that people who set their `LANGUAGE` environment variable to '`en@quot`' will get messages with proper looking

symmetric Unicode quotation marks instead of abusing the ASCII grave accent and the ASCII apostrophe for indicating quotations. To enable this catalog, simply add `en@quot` to the '`po/LINGUAS`' file. The effect of '`en@boldquot.po`' is that people who set `LANGUAGE` to '`en@boldquot`' will get not only proper quotation marks, but also the quoted text will be shown in a bold font on terminals and consoles. This catalog is useful only for command-line programs, not GUI programs. To enable it, similarly add `en@boldquot` to the '`po/LINGUAS`' file.

12.4.4 '`configure.in`' at top level

'`configure.in`' or '`configure.ac`' - this is the source from which `autoconf` generates the '`configure`' script.

1. Declare the package and version.

   This is done by a set of lines like these:

   ```
   PACKAGE=gettext
   VERSION=0.11.2
   AC_DEFINE_UNQUOTED(PACKAGE, "$PACKAGE")
   AC_DEFINE_UNQUOTED(VERSION, "$VERSION")
   AC_SUBST(PACKAGE)
   AC_SUBST(VERSION)
   ```

   or, if you are using GNU `automake`, by a line like this:

   ```
   AM_INIT_AUTOMAKE(gettext, 0.11.2)
   ```

   Of course, you replace '`gettext`' with the name of your package, and '`0.11.2`' by its version numbers, exactly as they should appear in the packaged `tar` file name of your distribution ('`gettext-0.11.2.tar.gz`', here).

2. Check for internationalization support.

   Here is the main `m4` macro for triggering internationalization support. Just add this line to '`configure.in`':

   ```
   AM_GNU_GETTEXT
   ```

   This call is purposely simple, even if it generates a lot of configure time checking and actions.

   If you have suppressed the '`intl/`' subdirectory by calling `gettextize` without '`--intl`' option, this call should read

   ```
   AM_GNU_GETTEXT([external])
   ```

3. Have output files created.

   The `AC_OUTPUT` directive, at the end of your '`configure.in`' file, needs to be modified in two ways:

   `AC_OUTPUT([`*existing configuration files* `intl/Makefile po/Makefile.in],`
   `[`*existing additional actions*`])`

   The modification to the first argument to `AC_OUTPUT` asks for substitution in the '`intl/`' and '`po/`' directories. Note the '`.in`' suffix used for '`po/`' only. This is because the distributed file is really '`po/Makefile.in.in`'.

   If you have suppressed the '`intl/`' subdirectory by calling `gettextize` without '`--intl`' option, then you don't need to add `intl/Makefile` to the `AC_OUTPUT` line.

12.4.5 'config.guess', 'config.sub' at top level

If you haven't suppressed the 'intl/' subdirectory, you need to add the GNU 'config.guess' and 'config.sub' files to your distribution. They are needed because the 'intl/' directory has platform dependent support for determining the locale's character encoding and therefore needs to identify the platform.

You can obtain the newest version of 'config.guess' and 'config.sub' from 'ftp://ftp.gnu.org/pub/gnu
Less recent versions are also contained in the GNU automake and GNU libtool packages.

Normally, 'config.guess' and 'config.sub' are put at the top level of a distribution. But it is also possible to put them in a subdirectory, altogether with other configuration support files like 'install-sh', 'ltconfig', 'ltmain.sh', 'mkinstalldirs' or 'missing'. All you need to do, other than moving the files, is to add the following line to your 'configure.in'.

```
AC_CONFIG_AUX_DIR([subdir])
```

12.4.6 'aclocal.m4' at top level

If you do not have an 'aclocal.m4' file in your distribution, the simplest is to concatenate the files 'codeset.m4', 'gettext.m4', 'glibc21.m4', 'iconv.m4', 'isc-posix.m4', 'lcmessage.m4', 'lib-ld.m4', 'lib-link.m4', 'lib-prefix.m4', 'progtest.m4' from GNU gettext's 'm4/' directory into a single file. If you have suppressed the 'intl/' directory, only 'gettext.m4', 'iconv.m4', 'lib-ld.m4', 'lib-link.m4', 'lib-prefix.m4', 'progtest.m4' need to be concatenated.

If you already have an 'aclocal.m4' file, then you will have to merge the said macro files into your 'aclocal.m4'. Note that if you are upgrading from a previous release of GNU gettext, you should most probably *replace* the macros (AM_GNU_GETTEXT, etc.), as they usually change a little from one release of GNU gettext to the next. Their contents may vary as we get more experience with strange systems out there.

If you are using GNU automake 1.5 or newer, it is enough to put these macro files into a subdirectory named 'm4/' and add the line

```
ACLOCAL_AMFLAGS = -I m4
```

to your top level 'Makefile.am'.

These macros check for the internationalization support functions and related informations. Hopefully, once stabilized, these macros might be integrated in the standard Autoconf set, because this piece of m4 code will be the same for all projects using GNU gettext.

12.4.7 'acconfig.h' at top level

Earlier GNU gettext releases required to put definitions for ENABLE_NLS, HAVE_GETTEXT and HAVE_LC_MESSAGES, HAVE_STPCPY, PACKAGE and VERSION into an 'acconfig.h' file. This is not needed any more; you can remove them from your 'acconfig.h' file unless your package uses them independently from the 'intl/' directory.

12.4.8 'Makefile.in' at top level

Here are a few modifications you need to make to your main, top-level 'Makefile.in' file.

1. Add the following lines near the beginning of your 'Makefile.in', so the 'dist:' goal will work properly (as explained further down):

   ```
   PACKAGE = @PACKAGE@
   VERSION = @VERSION@
   ```

2. Add file 'ABOUT-NLS' to the DISTFILES definition, so the file gets distributed.

3. Wherever you process subdirectories in your 'Makefile.in', be sure you also process dir subdirectories 'intl' and 'po'. Special rules in the 'Makefiles' take care for the case where no internationalization is wanted.

   If you are using Makefiles, either generated by automake, or hand-written so they carefully follow the GNU coding standards, the effected goals for which the new subdirectories must be handled include 'installdirs', 'install', 'uninstall', 'clean', 'distclean'.

   Here is an example of a canonical order of processing. In this example, we also define SUBDIRS in Makefile.in for it to be further used in the 'dist:' goal.

   ```
   SUBDIRS = doc intl lib src po
   ```

   Note that you must arrange for 'make' to descend into the intl directory before descending into other directories containing code which make use of the libintl.h header file. For this reason, here we mention intl before lib and src.

4. A delicate point is the 'dist:' goal, as both 'intl/Makefile' and 'po/Makefile' will later assume that the proper directory has been set up from the main 'Makefile'. Here is an example at what the 'dist:' goal might look like:

   ```
   distdir = $(PACKAGE)-$(VERSION)
   dist: Makefile
   rm -fr $(distdir)
   mkdir $(distdir)
   chmod 777 $(distdir)
   for file in $(DISTFILES); do \
     ln $$file $(distdir) 2>/dev/null || cp -p $$file $(distdir); \
   done
   for subdir in $(SUBDIRS); do \
     mkdir $(distdir)/$$subdir || exit 1; \
     chmod 777 $(distdir)/$$subdir; \
     (cd $$subdir && $(MAKE) $@) || exit 1; \
   done
   tar chozf $(distdir).tar.gz $(distdir)
   rm -fr $(distdir)
   ```

Note that if you are using GNU automake, 'Makefile.in' is automatically generated from 'Makefile.am', and all needed changes to 'Makefile.am' are already made by running 'gettextize'.

12.4.9 'Makefile.in' in 'src/'

Some of the modifications made in the main 'Makefile.in' will also be needed in the 'Makefile.in' from your package sources, which we assume here to be in the 'src/' subdirectory. Here are all the modifications needed in 'src/Makefile.in':

1. In view of the 'dist:' goal, you should have these lines near the beginning of 'src/Makefile.in':

   ```
   PACKAGE = @PACKAGE@
   VERSION = @VERSION@
   ```

2. If not done already, you should guarantee that top_srcdir gets defined. This will serve for cpp include files. Just add the line:

   ```
   top_srcdir = @top_srcdir@
   ```

3. You might also want to define subdir as 'src', later allowing for almost uniform 'dist:' goals in all your 'Makefile.in'. At list, the 'dist:' goal below assume that you used:

   ```
   subdir = src
   ```

4. The main function of your program will normally call bindtextdomain (see see Section 3.1 [Triggering], page 18), like this:

   ```
   bindtextdomain (PACKAGE, LOCALEDIR);
   ```

   To make LOCALEDIR known to the program, add the following lines to Makefile.in:

   ```
   datadir = @datadir@
   localedir = $(datadir)/locale
   DEFS = -DLOCALEDIR=\"$(localedir)\" @DEFS@
   ```

   Note that @datadir@ defaults to '$(prefix)/share', thus $(localedir) defaults to '$(prefix)/share/locale'.

5. You should ensure that the final linking will use @LIBINTL@ or @LTLIBINTL@ as a library. @LIBINTL@ is for use without libtool, @LTLIBINTL@ is for use with libtool. An easy way to achieve this is to manage that it gets into LIBS, like this:

   ```
   LIBS = @LIBINTL@ @LIBS@
   ```

   In most packages internationalized with GNU gettext, one will find a directory 'lib/' in which a library containing some helper functions will be build. (You need at least the few functions which the GNU gettext Library itself needs.) However some of the functions in the 'lib/' also give messages to the user which of course should be translated, too. Taking care of this, the support library (say 'libsupport.a') should be placed before @LIBINTL@ and @LIBS@ in the above example. So one has to write this:

   ```
   LIBS = ../lib/libsupport.a @LIBINTL@ @LIBS@
   ```

6. You should also ensure that directory 'intl/' will be searched for C preprocessor include files in all circumstances. So, you have to manage so both '-I../intl' and '-I$(top_srcdir)/intl' will be given to the C compiler.

7. Your 'dist:' goal has to conform with others. Here is a reasonable definition for it:

   ```
   distdir = ../$(PACKAGE)-$(VERSION)/$(subdir)
   dist: Makefile $(DISTFILES)
   for file in $(DISTFILES); do \
   ```

```
  ln $$file $(distdir) 2>/dev/null || cp -p $$file $(distdir); \
done
```

12.4.10 'gettext.h' in 'lib/'

Internationalization of packages, as provided by GNU gettext, is optional. It can be turned off in two situations:

- When the installer has specified './configure --disable-nls'. This can be useful when small binaries are more important than features, for example when building utilities for boot diskettes. It can also be useful in order to get some specific C compiler warnings about code quality with some older versions of GCC (older than 3.0).
- When the package does not include the intl/ subdirectory, and the libintl.h header (with its associated libintl library, if any) is not already installed on the system, it is preferrable that the package builds without internationalization support, rather than to give a compilation error.

A C preprocessor macro can be used to detect these two cases. Usually, when libintl.h was found and not explicitly disabled, the ENABLE_NLS macro will be defined to 1 in the autoconf generated configuration file (usually called 'config.h'). In the two negative situations, however, this macro will not be defined, thus it will evaluate to 0 in C preprocessor expressions.

'gettext.h' is a convenience header file for conditional use of '<libintl.h>', depending on the ENABLE_NLS macro. If ENABLE_NLS is set, it includes '<libintl.h>'; otherwise it defines no-op substitutes for the libintl.h functions. We recommend the use of "gettext.h" over direct use of '<libintl.h>', so that portability to older systems is guaranteed and installers can turn off internationalization if they want to. In the C code, you will then write

```
#include "gettext.h"
```

instead of

```
#include <libintl.h>
```

The location of gettext.h is usually in a directory containing auxiliary include files. In many GNU packages, there is a directory 'lib/' containing helper functions; 'gettext.h' fits there. In other packages, it can go into the 'src' directory.

Do not install the gettext.h file in public locations. Every package that needs it should contain a copy of it on its own.

12.5 Autoconf macros for use in 'configure.in'

GNU gettext installs macros for use in a package's 'configure.in' or 'configure.ac'. See section "Introduction" in *The Autoconf Manual*. The primary macro is, of course, AM_GNU_GETTEXT.

12.5.1 AM_GNU_GETTEXT in 'gettext.m4'

The AM_GNU_GETTEXT macro tests for the presence of the GNU gettext function family in either the C library or a separate libintl library (shared or static libraries are both supported) or in the package's 'intl/' directory.

`AM_GNU_GETTEXT` accepts up to three optional arguments. The general syntax is

```
AM_GNU_GETTEXT([intlsymbol], [needsymbol], [intldir])
```

intlsymbol can be one of '`external`', '`no-libtool`', '`use-libtool`'. The default (if it is not specified or empty) is '`no-libtool`'. *intlsymbol* should be '`external`' for packages with no '`intl/`' directory, and '`no-libtool`' or '`use-libtool`' for packages with an '`intl/`' directory. If *intlsymbol* is '`use-libtool`', then a libtool library `$(top_builddir)/intl/libintl.la` will be created (shared and/or static, depending on `--{enable,disable}-{shared,static}` and on the presence of `AM_DISABLE_SHARED`). If *intlsymbol* is '`no-libtool`', a static library `$(top_builddir)/intl/libintl.a` will be created.

If *needsymbol* is specified and is '`need-ngettext`', then GNU gettext implementations (in libc or libintl) without the `ngettext()` function will be ignored.

intldir is used to find the intl libraries. If empty, the value '`$(top_builddir)/intl/`' is used.

The `AM_GNU_GETTEXT` macro determines whether GNU gettext is available and should be used. If so, it sets the `USE_NLS` variable to '`yes`'; it defines `ENABLE_NLS` to 1 in the autoconf generated configuration file (usually called '`config.h`'); it sets the variables `LIBINTL` and `LTLIBINTL` to the linker options for use in a Makefile (`LIBINTL` for use without libtool, `LTLIBINTL` for use with libtool); it adds an '`-I`' option to `CPPFLAGS` if necessary. In the negative case, it sets `USE_NLS` to '`no`'; it sets `LIBINTL` and `LTLIBINTL` to empty and doesn't change `CPPFLAGS`.

The complexities that `AM_GNU_GETTEXT` deals with are the following:

- Some operating systems have `gettext` in the C library, for example glibc. Some have it in a separate library `libintl`. GNU `libintl` might have been installed as part of the GNU `gettext` package.
- GNU `libintl`, if installed, is not necessarily already in the search path (`CPPFLAGS` for the include file search path, `LDFLAGS` for the library search path).
- Except for glibc, the operating system's native `gettext` cannot exploit the GNU mo files, doesn't have the necessary locale dependency features, and cannot convert messages from the catalog's text encoding to the user's locale encoding.
- GNU `libintl`, if installed, is not necessarily already in the run time library search path. To avoid the need for setting an environment variable like `LD_LIBRARY_PATH`, the macro adds the appropriate run time search path options to the `LIBINTL` and `LTLIBINTL` variables. This works on most systems, but not on some operating systems with limited shared library support, like SCO.
- GNU `libintl` relies on POSIX `iconv`. The macro checks for linker options needed to use iconv and appends them to the `LIBINTL` and `LTLIBINTL` variables.

12.5.2 AM_ICONV in '`iconv.m4`'

The `AM_ICONV` macro tests for the presence of the POSIX `iconv` function family in either the C library or a separate `libiconv` library. If found, it sets the `am_cv_func_iconv` variable to '`yes`'; it defines `HAVE_ICONV` to 1 in the autoconf generated configuration file (usually called '`config.h`'); it defines `ICONV_CONST` to '`const`' or to empty, depending on

whether the second argument of `iconv()` is of type '`const char **`' or '`char **`'; it sets the variables `LIBICONV` and `LTLIBICONV` to the linker options for use in a Makefile (`LIBICONV` for use without libtool, `LTLIBICONV` for use with libtool); it adds an '`-I`' option to `CPPFLAGS` if necessary. If not found, it sets `LIBICONV` and `LTLIBICONV` to empty and doesn't change `CPPFLAGS`.

The complexities that `AM_ICONV` deals with are the following:

- Some operating systems have `iconv` in the C library, for example glibc. Some have it in a separate library `libiconv`, for example OSF/1 or FreeBSD. Regardless of the operating system, GNU `libiconv` might have been installed. In that case, it should be used instead of the operating system's native `iconv`.
- GNU `libiconv`, if installed, is not necessarily already in the search path (`CPPFLAGS` for the include file search path, `LDFLAGS` for the library search path).
- GNU `libiconv` is binary incompatible with some operating system's native `iconv`, for example on FreeBSD. Use of an '`iconv.h`' and '`libiconv.so`' that don't fit together would produce program crashes.
- GNU `libiconv`, if installed, is not necessarily already in the run time library search path. To avoid the need for setting an environment variable like `LD_LIBRARY_PATH`, the macro adds the appropriate run time search path options to the `LIBICONV` variable. This works on most systems, but not on some operating systems with limited shared library support, like SCO.

'`iconv.m4`' is distributed with the GNU gettext package because '`gettext.m4`' relies on it.

13 Other Programming Languages

While the presentation of `gettext` focuses mostly on C and implicitly applies to C++ as well, its scope is far broader than that: Many programming languages, scripting languages and other textual data like GUI resources or package descriptions can make use of the gettext approach.

13.1 The Language Implementor's View

All programming and scripting languages that have the notion of strings are eligible to supporting `gettext`. Supporting `gettext` means the following:

1. You should add to the language a syntax for translatable strings. In principle, a function call of `gettext` would do, but a shorthand syntax helps keeping the legibility of internationalized programs. For example, in C we use the syntax `_("string")`, in bash we use the syntax `$"string"`, and in GNU awk we use the shorthand `_"string"`.
2. You should arrange that evaluation of such a translatable string at runtime calls the `gettext` function, or performs equivalent processing.
3. Similarly, you should make the functions `ngettext`, `dcgettext`, `dcngettext` available from within the language. These functions are less often used, but are nevertheless necessary for particular purposes: `ngettext` for correct plural handling, and `dcgettext` and `dcngettext` for obeying other locale environment variables than `LC_MESSAGES`, such as `LC_TIME` or `LC_MONETARY`. For these latter functions, you need to make the `LC_*` constants, available in the C header `<locale.h>`, referenceable from within the language, usually either as enumeration values or as strings.
4. You should allow the programmer to designate a message domain, either by making the `textdomain` function available from within the language, or by introducing a magic variable called `TEXTDOMAIN`. Similarly, you should allow the programmer to designate where to search for message catalogs, by providing access to the `bindtextdomain` function.
5. You should either perform a `setlocale (LC_ALL, "")` call during the startup of your language runtime, or allow the programmer to do so. Remember that gettext will act as a no-op if the `LC_MESSAGES` and `LC_CTYPE` locale facets are not both set.
6. A programmer should have a way to extract translatable strings from a program into a PO file. The GNU `xgettext` program is being extended to support very different programming languages. Please contact the GNU `gettext` maintainers to help them doing this. If the string extractor is best integrated into your language's parser, GNU `xgettext` can function as a front end to your string extractor.
7. The language's library should have a string formatting facility where the arguments of a format string are denoted by a positional number or a name. This is needed because for some languages and some messages with more than one substitutable argument, the translation will need to output the substituted arguments in different order. See Section 3.5 [c-format], page 25.
8. If the language has more than one implementation, and not all of the implementations use `gettext`, but the programs should be portable across implementations, you should

provide a no-i18n emulation, that makes the other implementations accept programs written for yours, without actually translating the strings.

9. To help the programmer in the task of marking translatable strings, which is usually performed using the Emacs PO mode, you are welcome to contact the GNU `gettext` maintainers, so they can add support for your language to '`po-mode.el`'.

On the implementation side, three approaches are possible, with different effects on portability and copyright:

- You may integrate the GNU `gettext`'s '`intl/`' directory in your package, as described in Chapter 12 [Maintainers], page 102. This allows you to have internationalization on all kinds of platforms. Note that when you then distribute your package, it legally falls under the GNU General Public License, and the GNU project will be glad about your contribution to the Free Software pool.
- You may link against GNU `gettext` functions if they are found in the C library. For example, an autoconf test for `gettext()` and `ngettext()` will detect this situation. For the moment, this test will succeed on GNU systems and not on other platforms. No severe copyright restrictions apply.
- You may emulate or reimplement the GNU `gettext` functionality. This has the advantage of full portability and no copyright restrictions, but also the drawback that you have to reimplement the GNU `gettext` features (such as the `LANGUAGE` environment variable, the locale aliases database, the automatic charset conversion, and plural handling).

13.2 The Programmer's View

For the programmer, the general procedure is the same as for the C language. The Emacs PO mode supports other languages, and the GNU `xgettext` string extractor recognizes other languages based on the file extension or a command-line option. In some languages, `setlocale` is not needed because it is already performed by the underlying language runtime.

13.3 The Translator's View

The translator works exactly as in the C language case. The only difference is that when translating format strings, she has to be aware of the language's particular syntax for positional arguments in format strings.

13.4 The Maintainer's View

For the maintainer, the general procedure differs from the C language case in two ways.

- For those languages that don't use GNU gettext, the '`intl/`' directory is not needed and can be omitted. This means that the maintainer calls the `gettextize` program without the '`--intl`' option, and that he invokes the `AM_GNU_GETTEXT` autoconf macro via '`AM_GNU_GETTEXT([external])`'.

- If only a single programming language is used, the `XGETTEXT_OPTIONS` variable in '`po/Makevars`' (see Section 12.4.3 [po/Makevars], page 106) should be adjusted to match the `xgettext` options for that particular programming language. If the package uses more than one programming language with `gettext` support, it becomes necessary to change the POT file construction rule in '`po/Makefile.in.in`'. It is recommended to make one `xgettext` invocation per programming language, each with the options appropriate for that language, and to combine the resulting files using `msgcat`.

13.5 Individual Programming Languages

13.5.1 C, C++, Objective C

RPMs gcc, gpp, gobjc, glibc, gettext

File extension
For C: `c`, `h`.
For C++: `C`, `c++`, `cc`, `cxx`, `cpp`, `hpp`.
For Objective C: `m`.

String syntax
`"abc"`

gettext shorthand
`_("abc")`

gettext/ngettext functions
`gettext`, `dgettext`, `dcgettext`, `ngettext`, `dngettext`, `dcngettext`

textdomain
`textdomain` function

bindtextdomain
`bindtextdomain` function

setlocale Programmer must call `setlocale (LC_ALL, "")`

Prerequisite
```
#include <libintl.h>
#include <locale.h>
#define _(string) gettext (string)
```

Use or emulate GNU gettext
Use

Extractor `xgettext -k_`

Formatting with positions
`fprintf "%2$d %1$d"` (POSIX but not C 99)

Portability
autoconf (gettext.m4) and #if ENABLE_NLS

po-mode marking
yes

13.5.2 sh - Shell Script

RPMs bash, gettext

File extension
`sh`

String syntax
`"abc"`, `'abc'`, `abc`

gettext shorthand
`"‘gettext "abc"‘"`

gettext/ngettext functions
`gettext`, `ngettext` programs

textdomain
environment variable `TEXTDOMAIN`

bindtextdomain
environment variable `TEXTDOMAINDIR`

setlocale automatic

Prerequisite
—

Use or emulate GNU gettext
use

Extractor —

Formatting with positions
—

Portability
—

po-mode marking
—

13.5.3 bash - Bourne-Again Shell Script

RPMs bash 2.0 or newer, gettext

File extension
`sh`

String syntax
`"abc"`, `'abc'`, `abc`

gettext shorthand
`$"abc"`

gettext/ngettext functions
`gettext`, `ngettext` programs

textdomain
: environment variable `TEXTDOMAIN`

bindtextdomain
: environment variable `TEXTDOMAINDIR`

setlocale
: automatic

Prerequisite
: —

Use or emulate GNU gettext
: use

Extractor
: `bash --dump-po-strings`

Formatting with positions
: —

Portability
: —

po-mode marking
: —

13.5.4 Python

RPMs
: python

File extension
: `py`

String syntax
: `'abc'`, `u'abc'`, `r'abc'`, `ur'abc'`,
`"abc"`, `u"abc"`, `r"abc"`, `ur"abc"`,
`'''abc'''`, `u'''abc'''`, `r'''abc'''`, `ur'''abc'''`,
`"""abc"""`, `u"""abc"""`, `r"""abc"""`, `ur"""abc"""`

gettext shorthand
: `_('abc')` etc.

gettext/ngettext functions
: `gettext.gettext`, `gettext.dgettext`, also `ugettext`

textdomain
: `gettext.textdomain` function, or `gettext.install(`*domain*`)` function

bindtextdomain
: `gettext.bindtextdomain` function, or `gettext.install(`*domain*`,`*localedir*`)` function

setlocale
: not used by the gettext emulation

Prerequisite
: `import gettext`

Use or emulate GNU gettext
: emulate. Bug: uses only the first found .mo file, not all of them

Extractor `xgettext`

Formatting with positions
: `'...%(ident)d...' % { 'ident': value }`

Portability
: fully portable

po-mode marking
: —

13.5.5 GNU clisp - Common Lisp

RPMs clisp 2.28 or newer

File extension
: `lisp`

String syntax
: `"abc"`

gettext shorthand
: `(_ "abc")`, `(ENGLISH "abc")`

gettext/ngettext functions
: `i18n:gettext`, `i18n:ngettext`

textdomain
: `i18n:textdomain`

bindtextdomain
: `i18n:textdomaindir`

setlocale automatic

Prerequisite
: —

Use or emulate GNU gettext
: use

Extractor `xgettext -k_ -kENGLISH`

Formatting with positions
: `format "~1@*~D ~0@*~D"`

Portability
: On platforms without gettext, no translation.

po-mode marking
: —

13.5.6 GNU clisp C sources

RPMs clisp

File extension
`d`

String syntax
`"abc"`

gettext shorthand
`ENGLISH ? "abc" : ""`
`GETTEXT("abc")`
`GETTEXTL("abc")`

gettext/ngettext functions
`clgettext`, `clgettextl`

textdomain
—

bindtextdomain
—

setlocale automatic

Prerequisite
`#include "lispbibl.c"`

Use or emulate GNU gettext
use

Extractor `clisp-xgettext`

Formatting with positions
`fprintf "%2$d %1$d"` (POSIX but not C 99)

Portability
On platforms without gettext, no translation.

po-mode marking
—

13.5.7 Emacs Lisp

RPMs emacs, xemacs

File extension
`el`

String syntax
`"abc"`

gettext shorthand
`(_"abc")`

gettext/ngettext functions
: `gettext`, `dgettext` (xemacs only)

textdomain
: `domain` special form (xemacs only)

bindtextdomain
: `bind-text-domain` function (xemacs only)

setlocale automatic

Prerequisite
: —

Use or emulate GNU gettext
: use

Extractor `xgettext`

Formatting with positions
: `format "%2$d %1$d"`

Portability
: Only XEmacs. Without `I18N3` defined at build time, no translation.

po-mode marking
: —

13.5.8 librep

RPMs librep 0.15.3 or newer

File extension
: `jl`

String syntax
: `"abc"`

gettext shorthand
: `(_"abc")`

gettext/ngettext functions
: `gettext`

textdomain
: `textdomain` function

bindtextdomain
: `bindtextdomain` function

setlocale —

Prerequisite
: `(require 'rep.i18n.gettext)`

Use or emulate GNU gettext
: use

Extractor `xgettext`

Formatting with positions
`format "%2$d %1$d"`

Portability
On platforms without gettext, no translation.

po-mode marking
—

13.5.9 GNU Smalltalk

RPMs smalltalk

File extension
`st`

String syntax
`''abc''`

gettext shorthand
`NLS? ''abc''`
`self? ''abc''`

gettext/ngettext functions
`LcMessagesDomain>>#at:`, `LcMessagesDomain>>#at:plural:with:`

textdomain
`LcMessages>>#?` (returns a `LcMessagesDomain` object).
Example: `Locale default messages ? 'gettext'`

bindtextdomain
`LcMessages>>#domain:directory:` (returns a `LcMessagesDomain` object)

setlocale You can obtain any `Locale` object from `Locale` class methods such as `#fromString:` or `#default`.
Example: `Locale default messages` gives the `LcMessages` object for the default locale.

Prerequisite
The gettext code is contained in the '`I18N`' package.

Use or emulate GNU gettext
emulate

Extractor —

Formatting with positions
`'%1 %2' bindWith: 'Hello' with: 'world'`

Portability
fully portable

po-mode marking
—

13.5.10 Java

RPMs java, java2

File extension
`java`

String syntax
"abc"

gettext shorthand
_("abc")

gettext/ngettext functions
`GettextResource.gettext`, `GettextResource.ngettext`

textdomain
—, use `ResourceBundle.getResource` instead

bindtextdomain
—, use CLASSPATH instead

setlocale automatic

Prerequisite
—

Use or emulate GNU gettext
—, uses a Java specific message catalog format

Extractor `xgettext -k_`

Formatting with positions
`MessageFormat.format "{1,number} {0,number}"`

Portability
fully portable

po-mode marking
—

Before marking strings as internationalizable, uses of the string concatenation operator need to be converted to `MessageFormat` applications. For example, `"file "+filename+" not found"` becomes `MessageFormat.format("file {0} not found", new Object[] { filename })`. Only after this is done, can the strings be marked and extracted.

GNU gettext uses the native Java internationalization mechanism, namely `ResourceBundles`. To convert a PO file to a ResourceBundle, the `msgfmt` program can be used with the option `--java` or `--java2`. To convert a ResourceBundle back to a PO file, the `msgunfmt` program can be used with the option `--java`.

Two different programmatic APIs can be used to access ResourceBundles. Note that both APIs work with all kinds of ResourceBundles, whether GNU gettext generated classes, or other `.class` or `.properties` files.

1. The `java.util.ResourceBundle` API.

   In particular, its `getString` function returns a string translation. Note that a missing translation yields a `MissingResourceException`.

   This has the advantage of being the standard API. And it does not require any additional libraries, only the `msgfmt` generated `.class` files. But it cannot do plural handling, even if the resource was generated from a PO file with plural handling.

2. The `gnu.gettext.GettextResource` API.

   Reference documentation in Javadoc 1.1 style format is in the javadoc1 directory (`javadoc1/tree.html`) and in Javadoc 2 style format in the javadoc2 directory (`javadoc2/index.html`).

   Its `gettext` function returns a string translation. Note that when a translation is missing, the *msgid* argument is returned unchanged.

   This has the advantage of having the `ngettext` function for plural handling.

   To use this API, one needs the `libintl.jar` file which is part of the GNU gettext package and distributed under the LGPL.

13.5.11 GNU awk

RPMs gawk 3.1 or newer

File extension
`awk`

String syntax
`"abc"`

gettext shorthand
`_"abc"`

gettext/ngettext functions
`dcgettext`, missing `dcngettext` in gawk-3.1.0

textdomain
`TEXTDOMAIN` variable

bindtextdomain
`bindtextdomain` function

setlocale automatic, but missing `setlocale (LC_MESSAGES, "")` in gawk-3.1.0

Prerequisite
—

Use or emulate GNU gettext
use

Extractor `xgettext`

Formatting with positions
`printf "%2$d %1$d"` (GNU awk only)

Portability
: On platforms without gettext, no translation. On non-GNU awks, you must define `dcgettext`, `dcngettext` and `bindtextdomain` yourself.

po-mode marking
: —

13.5.12 Pascal - Free Pascal Compiler

RPMs fpk

File extension
: `pp`, `pas`

String syntax
: `'abc'`

gettext shorthand
: automatic

gettext/ngettext functions
: —, use `ResourceString` data type instead

textdomain
: —, use `TranslateResourceStrings` function instead

bindtextdomain
: —, use `TranslateResourceStrings` function instead

setlocale automatic, but uses only LANG, not LC_MESSAGES or LC_ALL

Prerequisite
: `{$mode delphi}` or `{$mode objfpc}`
`uses gettext;`

Use or emulate GNU gettext
: emulate partially

Extractor `ppc386` followed by `xgettext` or `rstconv`

Formatting with positions
: `uses sysutils;`
`format "%1:d %0:d"`

Portability
: ?

po-mode marking
: —

The Pascal compiler has special support for the `ResourceString` data type. It generates a `.rst` file. This is then converted to a `.pot` file by use of `xgettext` or `rstconv`. At runtime, a `.mo` file corresponding to translations of this `.pot` file can be loaded using the `TranslateResourceStrings` function in the `gettext` unit.

13.5.13 wxWindows library

RPMs wxGTK, gettext

File extension
`cpp`

String syntax
`"abc"`

gettext shorthand
`_("abc")`

gettext/ngettext functions
`wxLocale::GetString`, `wxGetTranslation`

textdomain
`wxLocale::AddCatalog`

bindtextdomain
`wxLocale::AddCatalogLookupPathPrefix`

setlocale `wxLocale::Init`, `wxSetLocale`

Prerequisite
`#include <wx/intl.h>`

Use or emulate GNU gettext
emulate, see `include/wx/intl.h` and `src/common/intl.cpp`

Extractor `xgettext`

Formatting with positions
—

Portability
fully portable

po-mode marking
yes

13.5.14 YCP - YaST2 scripting language

RPMs libycp, libycp-devel, yast2-core-translator

File extension
`ycp`

String syntax
`"abc"`

gettext shorthand
`_("abc")`

gettext/ngettext functions
`_()` with 1 or 3 arguments

textdomain
: `textdomain` statement

bindtextdomain
: —

setlocale
: —

Prerequisite
: —

Use or emulate GNU gettext
: use maps instead

Extractor
: `xgettext`

Formatting with positions
: `sformat "%2 %1"`

Portability
: fully portable

po-mode marking
: —

13.5.15 Tcl - Tk's scripting language

RPMs
: tcl

File extension
: `tcl`

String syntax
: `"abc"`

gettext shorthand
: `[_ "abc"]`

gettext/ngettext functions
: `::msgcat::mc`

textdomain
: —

bindtextdomain
: —, use `::msgcat::mcload` instead

setlocale
: automatic, uses LANG, but ignores LC_MESSAGES and LC_ALL

Prerequisite
: `package require msgcat`
`proc _ {s} {return [::msgcat::mc $s]}`

Use or emulate GNU gettext
: —, uses a Tcl specific message catalog format

Extractor `xgettext -k_`

Formatting with positions
`format "%2\$d %1\$d"`

Portability
fully portable

po-mode marking
—

Before marking strings as internationalizable, substitutions of variables into the string need to be converted to `format` applications. For example, `"file $filename not found"` becomes `[format "file %s not found" $filename]`. Only after this is done, can the strings be marked and extracted. After marking, this example becomes `[format [_ "file %s not found"] $filename]` or `[msgcat::mc "file %s not found" $filename]`. Note that the `msgcat::mc` function implicitly calls `format` when more than one argument is given.

13.5.16 Perl

RPMs perl, perl-gettext

File extension
`pl`, `PL`

String syntax
`"abc"`

gettext shorthand
—

gettext/ngettext functions
`gettext`, `dgettext`, `dcgettext`

textdomain
`textdomain` function

bindtextdomain
`bindtextdomain` function

setlocale Use `setlocale (LC_ALL, "");`

Prerequisite
```
use POSIX;
use Locale::gettext;
```

Use or emulate GNU gettext
use

Extractor ?

Formatting with positions
—

Portability
: ?

po-mode marking
: —

13.5.17 PHP Hypertext Preprocessor

RPMs mod_php4, phplib, phpdoc

File extension
: `php`, `php3`, `php4`

String syntax
: `"abc"`

gettext shorthand
: `_("abc")`

gettext/ngettext functions
: `gettext`, `dgettext`, `dcgettext`

textdomain
: `textdomain` function

bindtextdomain
: `bindtextdomain` function

setlocale `setlocale` function

Prerequisite
: —

Use or emulate GNU gettext
: use

Extractor —

Formatting with positions
: —

Portability
: On platforms without gettext, the functions are not available.

po-mode marking
: —

13.5.18 Pike

RPMs roxen

File extension
: `pike`

String syntax
: `"abc"`

gettext shorthand
: —

gettext/ngettext functions
: `gettext`, `dgettext`, `dcgettext`

textdomain
: `textdomain` function

bindtextdomain
: `bindtextdomain` function

setlocale
: `setlocale` function

Prerequisite
: `import Locale.Gettext;`

Use or emulate GNU gettext
: use

Extractor
: —

Formatting with positions
: —

Portability
: On platforms without gettext, the functions are not available.

po-mode marking
: —

13.6 Internationalizable Data

Here is a list of other data formats which can be internationalized using GNU gettext.

13.6.1 POT - Portable Object Template

RPMs
: gettext

File extension
: `pot`, `po`

Extractor
: `xgettext`

13.6.2 Resource String Table

RPMs
: fpk

File extension
: `rst`

Extractor
: `xgettext`, `rstconv`

13.6.3 Glade - GNOME user interface description

RPMs glade, libglade, xml-i18n-tools

File extension
`glade`

Extractor `xgettext`, `libglade-xgettext`

14 Concluding Remarks

We would like to conclude this GNU `gettext` manual by presenting an history of the Translation Project so far. We finally give a few pointers for those who want to do further research or readings about Native Language Support matters.

14.1 History of GNU `gettext`

Internationalization concerns and algorithms have been informally and casually discussed for years in GNU, sometimes around GNU `libc`, maybe around the incoming `Hurd`, or otherwise (nobody clearly remembers). And even then, when the work started for real, this was somewhat independently of these previous discussions.

This all began in July 1994, when Patrick D'Cruze had the idea and initiative of internationalizing version 3.9.2 of GNU `fileutils`. He then asked Jim Meyering, the maintainer, how to get those changes folded into an official release. That first draft was full of `#ifdef`s and somewhat disconcerting, and Jim wanted to find nicer ways. Patrick and Jim shared some tries and experimentations in this area. Then, feeling that this might eventually have a deeper impact on GNU, Jim wanted to know what standards were, and contacted Richard Stallman, who very quickly and verbally described an overall design for what was meant to become `glocale`, at that time.

Jim implemented `glocale` and got a lot of exhausting feedback from Patrick and Richard, of course, but also from Mitchum DSouza (who wrote a `catgets`-like package), Roland McGrath, maybe David MacKenzie, François Pinard, and Paul Eggert, all pushing and pulling in various directions, not always compatible, to the extent that after a couple of test releases, `glocale` was torn apart.

While Jim took some distance and time and became dad for a second time, Roland wanted to get GNU `libc` internationalized, and got Ulrich Drepper involved in that project. Instead of starting from `glocale`, Ulrich rewrote something from scratch, but more conformant to the set of guidelines who emerged out of the `glocale` effort. Then, Ulrich got people from the previous forum to involve themselves into this new project, and the switch from `glocale` to what was first named `msgutils`, renamed `nlsutils`, and later `gettext`, became officially accepted by Richard in May 1995 or so.

Let's summarize by saying that Ulrich Drepper wrote GNU `gettext` in April 1995. The first official release of the package, including PO mode, occurred in July 1995, and was numbered 0.7. Other people contributed to the effort by providing a discussion forum around Ulrich, writing little pieces of code, or testing. These are quoted in the `THANKS` file which comes with the GNU `gettext` distribution.

While this was being done, François adapted half a dozen of GNU packages to `glocale` first, then later to `gettext`, putting them in pretest, so providing along the way an effective user environment for fine tuning the evolving tools. He also took the responsibility of organizing and coordinating the Translation Project. After nearly a year of informal exchanges between people from many countries, translator teams started to exist in May 1995, through the creation and support by Patrick D'Cruze of twenty unmoderated mailing lists for that many native languages, and two moderated lists: one for reaching all teams at once, the other for reaching all willing maintainers of internationalized free software packages.

François also wrote PO mode in June 1995 with the collaboration of Greg McGary, as a kind of contribution to Ulrich's package. He also gave a hand with the GNU `gettext` Texinfo manual.

In 1997, Ulrich Drepper released the GNU libc 2.0, which included the `gettext`, `textdomain` and `bindtextdomain` functions.

In 2000, Ulrich Drepper added plural form handling (the `ngettext` function) to GNU libc. Later, in 2001, he released GNU libc 2.2.x, which is the first free C library with full internationalization support.

Ulrich being quite busy in his role of General Maintainer of GNU libc, he handed over the GNU `gettext` maintenance to Bruno Haible in 2000. Bruno added the plural form handling to the tools as well, added support for UTF-8 and CJK locales, and wrote a few new tools for manipulating PO files.

14.2 Related Readings

Eugene H. Dorr (‘`dorre@well.com`’) maintains an interesting bibliography on internationalization matters, called *Internationalization Reference List*, which is available as:

```
ftp://ftp.ora.com/pub/examples/nutshell/ujip/doc/i18n-books.txt
```

Michael Gschwind (‘`mike@vlsivie.tuwien.ac.at`’) maintains a Frequently Asked Questions (FAQ) list, entitled *Programming for Internationalisation*. This FAQ discusses writing programs which can handle different language conventions, character sets, etc.; and is applicable to all character set encodings, with particular emphasis on ISO 8859-1. It is regularly published in Usenet groups ‘`comp.unix.questions`’, ‘`comp.std.internat`’, ‘`comp.software.international`’, ‘`comp.lang.c`’, ‘`comp.windows.x`’, ‘`comp.std.c`’, ‘`comp.answers`’ and ‘`news.answers`’. The home location of this document is:

```
ftp://ftp.vlsivie.tuwien.ac.at/pub/8bit/ISO-programming
```

Patrick D'Cruze (‘`pdcruze@li.org`’) wrote a tutorial about NLS matters, and Jochen Hein (‘`Hein@student.tu-clausthal.de`’) took over the responsibility of maintaining it. It may be found as:

```
ftp://sunsite.unc.edu/pub/Linux/utils/nls/catalogs/Incoming/...
     ...locale-tutorial-0.8.txt.gz
```

This site is mirrored in:

```
ftp://ftp.ibp.fr/pub/linux/sunsite/
```

A French version of the same tutorial should be findable at:

```
ftp://ftp.ibp.fr/pub/linux/french/docs/
```

together with French translations of many Linux-related documents.

Appendix A Language Codes

The ISO 639 standard defines two character codes for many languages. All abbreviations for languages used in the Translation Project should come from this standard.

'aa' Afar.

'ab' Abkhazian.

'ae' Avestan.

'af' Afrikaans.

'am' Amharic.

'ar' Arabic.

'as' Assamese.

'ay' Aymara.

'az' Azerbaijani.

'ba' Bashkir.

'be' Byelorussian; Belarusian.

'bg' Bulgarian.

'bh' Bihari.

'bi' Bislama.

'bn' Bengali; Bangla.

'bo' Tibetan.

'br' Breton.

'bs' Bosnian.

'ca' Catalan.

'ce' Chechen.

'ch' Chamorro.

'co' Corsican.

'cs' Czech.

'cu' Church Slavic.

'cv' Chuvash.

'cy' Welsh.

'da' Danish.

'de' German.

'dz' Dzongkha; Bhutani.

‘el’ Greek.

‘en’ English.

‘eo’ Esperanto.

‘es’ Spanish.

‘et’ Estonian.

‘eu’ Basque.

‘fa’ Persian.

‘fi’ Finnish.

‘fj’ Fijian; Fiji.

‘fo’ Faroese.

‘fr’ French.

‘fy’ Frisian.

‘ga’ Irish.

‘gd’ Scots; Gaelic.

‘gl’ Gallegan; Galician.

‘gn’ Guarani.

‘gu’ Gujarati.

‘gv’ Manx.

‘ha’ Hausa (?).

‘he’ Hebrew (formerly iw).

‘hi’ Hindi.

‘ho’ Hiri Motu.

‘hr’ Croatian.

‘hu’ Hungarian.

‘hy’ Armenian.

‘hz’ Herero.

‘ia’ Interlingua.

‘id’ Indonesian (formerly in).

‘ie’ Interlingue.

‘ik’ Inupiak.

‘io’ Ido.

‘is’ Icelandic.

‘`it`’ Italian.

‘`iu`’ Inuktitut.

‘`ja`’ Japanese.

‘`jv`’ Javanese.

‘`ka`’ Georgian.

‘`ki`’ Kikuyu.

‘`kj`’ Kuanyama.

‘`kk`’ Kazakh.

‘`kl`’ Kalaallisut; Greenlandic.

‘`km`’ Khmer; Cambodian.

‘`kn`’ Kannada.

‘`ko`’ Korean.

‘`ks`’ Kashmiri.

‘`ku`’ Kurdish.

‘`kv`’ Komi.

‘`kw`’ Cornish.

‘`ky`’ Kirghiz.

‘`la`’ Latin.

‘`lb`’ Letzeburgesch.

‘`ln`’ Lingala.

‘`lo`’ Lao; Laotian.

‘`lt`’ Lithuanian.

‘`lv`’ Latvian; Lettish.

‘`mg`’ Malagasy.

‘`mh`’ Marshall.

‘`mi`’ Maori.

‘`mk`’ Macedonian.

‘`ml`’ Malayalam.

‘`mn`’ Mongolian.

‘`mo`’ Moldavian.

‘`mr`’ Marathi.

‘`ms`’ Malay.

‘mt’ Maltese.

‘my’ Burmese.

‘na’ Nauru.

‘nb’ Norwegian Bokmål.

‘nd’ Ndebele, North.

‘ne’ Nepali.

‘ng’ Ndonga.

‘nl’ Dutch.

‘nn’ Norwegian Nynorsk.

‘no’ Norwegian.

‘nr’ Ndebele, South.

‘nv’ Navajo.

‘ny’ Chichewa; Nyanja.

‘oc’ Occitan; Provençal.

‘om’ (Afan) Oromo.

‘or’ Oriya.

‘os’ Ossetian; Ossetic.

‘pa’ Panjabi; Punjabi.

‘pi’ Pali.

‘pl’ Polish.

‘ps’ Pashto, Pushto.

‘pt’ Portuguese.

‘qu’ Quechua.

‘rm’ Rhaeto-Romance.

‘rn’ Rundi; Kirundi.

‘ro’ Romanian.

‘ru’ Russian.

‘rw’ Kinyarwanda.

‘sa’ Sanskrit.

‘sc’ Sardinian.

‘sd’ Sindhi.

‘se’ Northern Sami.

‘sg’ Sango; Sangro.

‘si’ Sinhalese.

‘sk’ Slovak.

‘sl’ Slovenian.

‘sm’ Samoan.

‘sn’ Shona.

‘so’ Somali.

‘sq’ Albanian.

‘sr’ Serbian.

‘ss’ Swati; Siswati.

‘st’ Sesotho; Sotho, Southern.

‘su’ Sundanese.

‘sv’ Swedish.

‘sw’ Swahili.

‘ta’ Tamil.

‘te’ Telugu.

‘tg’ Tajik.

‘th’ Thai.

‘ti’ Tigrinya.

‘tk’ Turkmen.

‘tl’ Tagalog.

‘tn’ Tswana; Setswana.

‘to’ Tonga (?).

‘tr’ Turkish.

‘ts’ Tsonga.

‘tt’ Tatar.

‘tw’ Twi.

‘ty’ Tahitian.

‘ug’ Uighur.

‘uk’ Ukrainian.

‘ur’ Urdu.

‘uz’ Uzbek.

‘`vi`’ Vietnamese.

‘`vo`’ Volapük; Volapuk.

‘`wa`’ Walloon.

‘`wo`’ Wolof.

‘`xh`’ Xhosa.

‘`yi`’ Yiddish (formerly ji).

‘`yo`’ Yoruba.

‘`za`’ Zhuang.

‘`zh`’ Chinese.

‘`zu`’ Zulu.

Appendix B Country Codes

The ISO 3166 standard defines two character codes for many countries and territories. All abbreviations for countries used in the Translation Project should come from this standard.

'AD' Andorra.

'AE' United Arab Emirates.

'AF' Afghanistan.

'AG' Antigua and Barbuda.

'AI' Anguilla.

'AL' Albania.

'AM' Armenia.

'AN' Netherlands Antilles.

'AO' Angola.

'AQ' Antarctica.

'AR' Argentina.

'AS' Samoa (American).

'AT' Austria.

'AU' Australia.

'AW' Aruba.

'AZ' Azerbaijan.

'BA' Bosnia and Herzegovina.

'BB' Barbados.

'BD' Bangladesh.

'BE' Belgium.

'BF' Burkina Faso.

'BG' Bulgaria.

'BH' Bahrain.

'BI' Burundi.

'BJ' Benin.

'BM' Bermuda.

'BN' Brunei.

'BO' Bolivia.

'BR'	Brazil.
'BS'	Bahamas.
'BT'	Bhutan.
'BV'	Bouvet Island.
'BW'	Botswana.
'BY'	Belarus.
'BZ'	Belize.
'CA'	Canada.
'CC'	Cocos (Keeling) Islands.
'CD'	Congo (Dem. Rep.).
'CF'	Central African Rep..
'CG'	Congo (Rep.).
'CH'	Switzerland.
'CI'	Cote d'Ivoire.
'CK'	Cook Islands.
'CL'	Chile.
'CM'	Cameroon.
'CN'	China.
'CO'	Colombia.
'CR'	Costa Rica.
'CU'	Cuba.
'CV'	Cape Verde.
'CX'	Christmas Island.
'CY'	Cyprus.
'CZ'	Czech Republic.
'DE'	Germany.
'DJ'	Djibouti.
'DK'	Denmark.
'DM'	Dominica.
'DO'	Dominican Republic.
'DZ'	Algeria.
'EC'	Ecuador.

'EE' Estonia.

'EG' Egypt.

'EH' Western Sahara.

'ER' Eritrea.

'ES' Spain.

'ET' Ethiopia.

'FI' Finland.

'FJ' Fiji.

'FK' Falkland Islands.

'FM' Micronesia.

'FO' Faeroe Islands.

'FR' France.

'GA' Gabon.

'GB' Britain (UK).

'GD' Grenada.

'GE' Georgia.

'GF' French Guiana.

'GH' Ghana.

'GI' Gibraltar.

'GL' Greenland.

'GM' Gambia.

'GN' Guinea.

'GP' Guadeloupe.

'GQ' Equatorial Guinea.

'GR' Greece.

'GS' South Georgia and the South Sandwich Islands.

'GT' Guatemala.

'GU' Guam.

'GW' Guinea-Bissau.

'GY' Guyana.

'HK' Hong Kong.

'HM' Heard Island and McDonald Islands.

'HN' Honduras.

'HR' Croatia.

'HT' Haiti.

'HU' Hungary.

'ID' Indonesia.

'IE' Ireland.

'IL' Israel.

'IN' India.

'IO' British Indian Ocean Territory.

'IQ' Iraq.

'IR' Iran.

'IS' Iceland.

'IT' Italy.

'JM' Jamaica.

'JO' Jordan.

'JP' Japan.

'KE' Kenya.

'KG' Kyrgyzstan.

'KH' Cambodia.

'KI' Kiribati.

'KM' Comoros.

'KN' St Kitts and Nevis.

'KP' Korea (North).

'KR' Korea (South).

'KW' Kuwait.

'KY' Cayman Islands.

'KZ' Kazakhstan.

'LA' Laos.

'LB' Lebanon.

'LC' St Lucia.

'LI' Liechtenstein.

'LK' Sri Lanka.

'LR' Liberia.

'LS' Lesotho.

'LT' Lithuania.

'LU' Luxembourg.

'LV' Latvia.

'LY' Libya.

'MA' Morocco.

'MC' Monaco.

'MD' Moldova.

'MG' Madagascar.

'MH' Marshall Islands.

'MK' Macedonia.

'ML' Mali.

'MM' Myanmar (Burma).

'MN' Mongolia.

'MO' Macao.

'MP' Northern Mariana Islands.

'MQ' Martinique.

'MR' Mauritania.

'MS' Montserrat.

'MT' Malta.

'MU' Mauritius.

'MV' Maldives.

'MW' Malawi.

'MX' Mexico.

'MY' Malaysia.

'MZ' Mozambique.

'NA' Namibia.

'NC' New Caledonia.

'NE' Niger.

'NF' Norfolk Island.

'NG' Nigeria.

'NI'	Nicaragua.
'NL'	Netherlands.
'NO'	Norway.
'NP'	Nepal.
'NR'	Nauru.
'NU'	Niue.
'NZ'	New Zealand.
'OM'	Oman.
'PA'	Panama.
'PE'	Peru.
'PF'	French Polynesia.
'PG'	Papua New Guinea.
'PH'	Philippines.
'PK'	Pakistan.
'PL'	Poland.
'PM'	St Pierre and Miquelon.
'PN'	Pitcairn.
'PR'	Puerto Rico.
'PS'	Palestine.
'PT'	Portugal.
'PW'	Palau.
'PY'	Paraguay.
'QA'	Qatar.
'RE'	Reunion.
'RO'	Romania.
'RU'	Russia.
'RW'	Rwanda.
'SA'	Saudi Arabia.
'SB'	Solomon Islands.
'SC'	Seychelles.
'SD'	Sudan.
'SE'	Sweden.

'SG'	Singapore.
'SH'	St Helena.
'SI'	Slovenia.
'SJ'	Svalbard and Jan Mayen.
'SK'	Slovakia.
'SL'	Sierra Leone.
'SM'	San Marino.
'SN'	Senegal.
'SO'	Somalia.
'SR'	Suriname.
'ST'	Sao Tome and Principe.
'SV'	El Salvador.
'SY'	Syria.
'SZ'	Swaziland.
'TC'	Turks and Caicos Is.
'TD'	Chad.
'TF'	French Southern and Antarctic Lands.
'TG'	Togo.
'TH'	Thailand.
'TJ'	Tajikistan.
'TK'	Tokelau.
'TM'	Turkmenistan.
'TN'	Tunisia.
'TO'	Tonga.
'TP'	East Timor.
'TR'	Turkey.
'TT'	Trinidad and Tobago.
'TV'	Tuvalu.
'TW'	Taiwan.
'TZ'	Tanzania.
'UA'	Ukraine.
'UG'	Uganda.

'UM'	US minor outlying islands.
'US'	United States.
'UY'	Uruguay.
'UZ'	Uzbekistan.
'VA'	Vatican City.
'VC'	St Vincent.
'VE'	Venezuela.
'VG'	Virgin Islands (UK).
'VI'	Virgin Islands (US).
'VN'	Vietnam.
'VU'	Vanuatu.
'WF'	Wallis and Futuna.
'WS'	Samoa (Western).
'YE'	Yemen.
'YT'	Mayotte.
'YU'	Yugoslavia.
'ZA'	South Africa.
'ZM'	Zambia.
'ZW'	Zimbabwe.

Program Index

G

gettext 117
gettextize 104

M

msgattrib 65
msgcat 51
msgcmp 64
msgcomm 62
msgconv 54
msgen 67
msgexec 69
msgfilter 58
msgfmt 70
msggrep 55
msginit 32
msgmerge 36
msgunfmt 73
msguniq 60

N

ngettext 117

X

xgettext 28

Option Index

-

--add-comments, xgettext option 29
--add-location, msgattrib option 66
--add-location, msgcat option 53
--add-location, msgcomm option 63
--add-location, msgconv option 55
--add-location, msgen option 68
--add-location, msgfilter option 59
--add-location, msggrep option 57
--add-location, msgmerge option 37
--add-location, msguniq option 61
--add-location, xgettext option 30
--alignment, msgfmt option 72
--backup, msgmerge option 37
--c++, xgettext option 29
--check, msgfmt option 71
--check-accelerators, msgfmt option 72
--check-compatibility, msgfmt option 72
--check-domain, msgfmt option 72
--check-format, msgfmt option 71
--check-header, msgfmt option 72
--clear-fuzzy, msgattrib option 66
--clear-obsolete, msgattrib option 66
--compendium, msgmerge option 36
--copy, gettextize option 104
--copyright-holder, xgettext option 31
--debug, xgettext option 29
--default-domain, xgettext option 28
--directory, msgattrib option 65
--directory, msgcat option 52
--directory, msgcmp option 64
--directory, msgcomm option 62
--directory, msgconv option 54
--directory, msgen option 67
--directory, msgexec option 69
--directory, msgfilter option 58
--directory, msgfmt option 70
--directory, msggrep option 55
--directory, msgmerge option 36
--directory, msguniq option 60
--directory, xgettext option 28
--domain, msggrep option 56
--exclude-file, xgettext option 29
--expression, msgfilter option 59
--extended-regexp, msggrep option 56
--extract-all, xgettext option 29
--file, msgfilter option 59
--file, msggrep option 57
--files-from, msgcat option 52
--files-from, msgcomm option 62
--files-from, xgettext option 28
--fixed-strings, msggrep option 57
--force, gettextize option 104
--force-po, msgattrib option 66
--force-po, msgcat option 53
--force-po, msgcomm option 63
--force-po, msgconv option 54
--force-po, msgen option 68
--force-po, msgfilter option 59
--force-po, msggrep option 57
--force-po, msgmerge option 37
--force-po, msgunfmt option 74
--force-po, msguniq option 61
--force-po, xgettext option 30
--foreign-user, xgettext option 31
--fuzzy, msgattrib option 66
--help, gettextize option 104
--help, msgattrib option 67
--help, msgcat option 53
--help, msgcmp option 65
--help, msgcomm option 64
--help, msgconv option 55
--help, msgen option 68
--help, msgexec option 69
--help, msgfilter option 60
--help, msgfmt option 72
--help, msggrep option 58
--help, msginit option 33
--help, msgmerge option 38
--help, msgunfmt option 74
--help, msguniq option 62
--help, xgettext option 31
--ignore-case, msggrep option 57
--indent, msgattrib option 66
--indent, msgcat option 53
--indent, msgcomm option 63
--indent, msgconv option 54
--indent, msgen option 68
--indent, msgfilter option 59
--indent, msggrep option 57
--indent, msgmerge option 37
--indent, msgunfmt option 74
--indent, msguniq option 61
--indent, xgettext option 30

--input, msgexec option ... 69
--input, msgfilter option ... 58
--input, msginit option ... 32
--intl, gettextize option ... 104
--java, msgfmt option ... 70
--java, msgunfmt option ... 73
--java2, msgfmt option ... 70
--join-existing, xgettext option ... 29
--keep-header, msgfilter option ... 59
--keyword, xgettext option ... 29
--language, xgettext option ... 28
--less-than, msgcat option ... 52
--less-than, msgcomm option ... 63
--locale, msgfmt option ... 71
--locale, msginit option ... 32
--locale, msgunfmt option ... 73
--location, msggrep option ... 56
--more-than, msgcat option ... 52
--more-than, msgcomm option ... 63
--msgid, msggrep option ... 56
--msgstr, msggrep option ... 56
--msgstr-prefix, xgettext option ... 31
--msgstr-suffix, xgettext option ... 31
--multi-domain, msgcmp option ... 64
--multi-domain, msgmerge option ... 37
--no-changelog, gettextize option ... 104
--no-fuzzy, msgattrib option ... 65
--no-hash, msgfmt option ... 72
--no-location, msgattrib option ... 66
--no-location, msgcat option ... 53
--no-location, msgcomm option ... 63
--no-location, msgconv option ... 54
--no-location, msgen option ... 68
--no-location, msgfilter option ... 59
--no-location, msggrep option ... 57
--no-location, msgmerge option ... 37
--no-location, msguniq option ... 61
--no-location, xgettext option ... 30
--no-obsolete, msgattrib option ... 65
--no-translator, msginit option ... 32
--no-wrap, msgattrib option ... 67
--no-wrap, msgcat option ... 53
--no-wrap, msgcomm option ... 63
--no-wrap, msgconv option ... 55
--no-wrap, msgen option ... 68
--no-wrap, msgfilter option ... 60
--no-wrap, msggrep option ... 57
--no-wrap, msginit option ... 33
--no-wrap, msgmerge option ... 38
--no-wrap, msgunfmt option ... 74
--no-wrap, msguniq option ... 62
--no-wrap, xgettext option ... 30
--obsolete, msgattrib option ... 66
--omit-header, msgcomm option ... 64
--omit-header, xgettext option ... 30
--only-fuzzy, msgattrib option ... 65
--only-obsolete, msgattrib option ... 66
--output, xgettext option ... 28
--output-dir, xgettext option ... 28
--output-file, msgattrib option ... 65
--output-file, msgcat option ... 52
--output-file, msgcomm option ... 63
--output-file, msgconv option ... 54
--output-file, msgen option ... 67
--output-file, msgfilter option ... 58
--output-file, msgfmt option ... 70
--output-file, msggrep option ... 56
--output-file, msginit option ... 32
--output-file, msgmerge option ... 36
--output-file, msgunfmt option ... 74
--output-file, msguniq option ... 61
--quiet, msgfilter option ... 59
--quiet, msgmerge option ... 38
--regexp=, msggrep option ... 57
--repeated, msguniq option ... 61
--resource, msgfmt option ... 71
--resource, msgunfmt option ... 73
--set-fuzzy, msgattrib option ... 66
--set-obsolete, msgattrib option ... 66
--silent, msgfilter option ... 59
--silent, msgmerge option ... 38
--sort-by-file, msgattrib option ... 67
--sort-by-file, msgcat option ... 53
--sort-by-file, msgcomm option ... 64
--sort-by-file, msgconv option ... 55
--sort-by-file, msgen option ... 68
--sort-by-file, msgfilter option ... 60
--sort-by-file, msggrep option ... 57
--sort-by-file, msgmerge option ... 38
--sort-by-file, msguniq option ... 62
--sort-by-file, xgettext option ... 30
--sort-output, msgattrib option ... 67
--sort-output, msgcat option ... 53
--sort-output, msgcomm option ... 64
--sort-output, msgconv option ... 55
--sort-output, msgen option ... 68
--sort-output, msgfilter option ... 60
--sort-output, msggrep option ... 57
--sort-output, msgmerge option ... 38
--sort-output, msgunfmt option ... 74

--sort-output, msguniq option ... 62
--sort-output, xgettext option ... 30
--statistics, msgfmt option ... 73
--strict, msgattrib option ... 66
--strict, msgcat option ... 53
--strict, msgcomm option ... 63
--strict, msgconv option ... 55
--strict, msgen option ... 68
--strict, msgfilter option ... 59
--strict, msgfmt option ... 70
--strict, msggrep option ... 57
--strict, msgmerge option ... 37
--strict, msgunfmt option ... 74
--strict, msguniq option ... 61
--strict, xgettext option ... 30
--suffix, msgmerge option ... 37
--tcl, msgfmt option ... 70
--tcl, msgunfmt option ... 73
--to-code, msgcat option ... 53
--to-code, msgconv option ... 54
--to-code, msguniq option ... 61
--translated, msgattrib option ... 65
--trigraphs, xgettext option ... 29
--unique, msgcat option ... 52
--unique, msgcomm option ... 63
--unique, msguniq option ... 61
--untranslated, msgattrib option ... 65
--update, msgmerge option ... 36
--use-first, msgcat option ... 53
--use-first, msguniq option ... 61
--use-fuzzy, msgfmt option ... 72
--verbose, msgfmt option ... 73
--verbose, msgmerge option ... 38
--verbose, msgunfmt option ... 74
--version, gettextize option ... 104
--version, msgattrib option ... 67
--version, msgcat option ... 54
--version, msgcmp option ... 65
--version, msgcomm option ... 64
--version, msgconv option ... 55
--version, msgen option ... 68
--version, msgexec option ... 69
--version, msgfilter option ... 60
--version, msgfmt option ... 72
--version, msggrep option ... 58
--version, msginit option ... 33
--version, msgmerge option ... 38
--version, msgunfmt option ... 74
--version, msguniq option ... 62
--version, xgettext option ... 31
--width, msgattrib option ... 66
--width, msgcat option ... 53
--width, msgcomm option ... 63
--width, msgconv option ... 55
--width, msgen option ... 68
--width, msgfilter option ... 59
--width, msggrep option ... 57
--width, msginit option ... 33
--width, msgmerge option ... 38
--width, msgunfmt option ... 74
--width, msguniq option ... 61
--width, xgettext option ... 30
->, msgcat option ... 52
->, msgcomm option ... 63
-<, msgcat option ... 52
-<, msgcomm option ... 63
-a, msgfmt option ... 72
-a, xgettext option ... 29
-c, gettextize option ... 104
-c, msgfmt option ... 71
-C, msgfmt option ... 72
-C, msgmerge option ... 36
-c, xgettext option ... 29
-C, xgettext option ... 29
-D, msgattrib option ... 65
-D, msgcat option ... 52
-D, msgcmp option ... 64
-D, msgcomm option ... 62
-D, msgconv option ... 54
-D, msgen option ... 67
-D, msgexec option ... 69
-D, msgfilter option ... 58
-d, msgfmt option ... 71
-D, msgfmt option ... 70
-D, msggrep option ... 55
-D, msgmerge option ... 36
-d, msgunfmt option ... 73
-d, msguniq option ... 61
-D, msguniq option ... 60
-d, xgettext option ... 28
-D, xgettext option ... 28
-e, msgfilter option ... 59
-e, msggrep option ... 57
-E, msggrep option ... 56
-f, gettextize option ... 104
-F, msgattrib option ... 67
-f, msgcat option ... 52
-F, msgcat option ... 53
-f, msgcomm option ... 62
-F, msgcomm option ... 64

-F, msgconv option 55
-F, msgen option 68
-f, msgfilter option 59
-F, msgfilter option 60
-f, msgfmt option 72
-f, msggrep option 57
-F, msggrep option 57
-F, msgmerge option 38
-F, msguniq option 62
-f, xgettext option 28
-F, xgettext option 30
-h, msgattrib option 67
-h, msgcat option 53
-h, msgcmp option 65
-h, msgcomm option 64
-h, msgconv option 55
-h, msgen option 68
-h, msgexec option 69
-h, msgfilter option 60
-h, msgfmt option 72
-h, msggrep option 58
-h, msginit option 33
-h, msgmerge option 38
-h, msgunfmt option 74
-h, msguniq option 62
-h, xgettext option 31
-i, msgattrib option 66
-i, msgcat option 53
-i, msgcomm option 63
-i, msgconv option 54
-i, msgen option 68
-i, msgexec option 69
-i, msgfilter option 58
-i, msggrep option 57
-i, msginit option 32
-i, msgmerge option 37
-i, msgunfmt option 74
-i, msguniq option 61
-i, xgettext option 30
-j, msgfmt option 70
-j, msgunfmt option 73
-j, xgettext option 29
-K, msggrep option 56
-k, xgettext option 29
-l, msgfmt option 71
-l, msginit option 32
-l, msgunfmt option 73
-L, xgettext option 28
-m, msgcmp option 64
-M, msggrep option 56
-m, msgmerge option 37
-m, xgettext option 31
-M, xgettext option 31
-n, msgattrib option 66
-n, msgcat option 53
-n, msgcomm option 63
-n, msgfilter option 59
-N, msggrep option 56
-n, msguniq option 61
-n, xgettext option 30
-o, msgattrib option 65
-o, msgcat option 52
-o, msgcomm option 63
-o, msgconv option 54
-o, msgen option 67
-o, msgfilter option 58
-o, msgfmt option 70
-o, msggrep option 56
-o, msginit option 32
-o, msgmerge option 36
-o, msgunfmt option 74
-o, msguniq option 61
-o, xgettext option 28
-p, xgettext option 28
-q, msgmerge option 38
-r, msgfmt option 71
-r, msgunfmt option 73
-s, msgattrib option 67
-s, msgcat option 53
-s, msgcomm option 64
-s, msgconv option 55
-s, msgen option 68
-s, msgfilter option 60
-s, msgmerge option 38
-s, msgunfmt option 74
-s, msguniq option 62
-s, xgettext option 30
-t, msgcat option 53
-t, msgconv option 54
-T, msggrep option 56
-t, msguniq option 61
-T, xgettext option 29
-u, msgcat option 52
-u, msgcomm option 63
-U, msgmerge option 36
-u, msguniq option 61
-V, msgattrib option 67
-V, msgcat option 54
-V, msgcmp option 65
-V, msgcomm option 64

-V, msgconv option 55
-V, msgen option 68
-V, msgexec option 69
-V, msgfilter option 60
-v, msgfmt option 73
-V, msgfmt option 72
-V, msggrep option 58
-V, msginit option 33
-v, msgmerge option 38
-V, msgmerge option 38
-v, msgunfmt option 74
-V, msgunfmt option 74
-V, msguniq option 62
-V, xgettext option 31
-w, msgattrib option 66
-w, msgcat option 53
-w, msgcomm option 63
-w, msgconv option 55
-w, msgen option 68
-w, msgfilter option 59
-w, msggrep option 57
-w, msginit option 33
-w, msgmerge option 38
-w, msgunfmt option 74
-w, msguniq option 61
-w, xgettext option 30
-x, xgettext option 29

Variable Index

L

`LANG`, environment variable 79, 93
`LANGUAGE`, environment variable 93, 106
`LC_ALL`, environment variable 93
`LC_COLLATE`, environment variable 93
`LC_CTYPE`, environment variable 93
`LC_MESSAGES`, environment variable 93
`LC_MONETARY`, environment variable 93
`LC_NUMERIC`, environment variable 93
`LC_TIME`, environment variable 93
`LINGUAS`, environment variable 78

M

`MSGEXEC_LOCATION`, environment variable 69
`MSGEXEC_MSGID`, environment variable 69

T

`TEXTDOMAIN`, environment variable 117, 118
`TEXTDOMAINDIR`, environment variable 117, 118

PO Mode Index

#

`#`, PO Mode command 44

,

`,`, PO Mode command 23

.

`.`, PO Mode command 14, 15
‘`.emacs`’ customizations 10

=

`=`, PO Mode command 13, 14

?

`?`, PO Mode command 13, 14

_

`_`, PO Mode command 13

>

`>`, PO Mode command 15

<

`<`, PO Mode command 15

0

`0`, PO Mode command 13, 14

A

`a`, PO Mode command 48
`A`, PO Mode command 48
auxiliary PO file 48

C

`C-c C-a`, PO Mode command 45, 48
`C-c C-c`, PO Mode command 45
`C-c C-k`, PO Mode command 45
`C-j`, PO Mode command 42
commands 13
comment out PO file entry 41
consulting program sources 46
consulting translations to other languages 48
current entry of a PO file 14
cut and paste for translated strings 42

D

`DEL`, PO Mode command 39, 41

E

editing comments 44
editing multiple entries 46
editing translations 41
`etags`, using for marking strings 23
exiting PO subedit 45

F

find source fragment for a PO file entry 46

H

`h`, PO Mode command 13, 14

I

installing PO mode 10

K

`k`, PO Mode command 40, 42
`K`, PO Mode command 44

L

`LFD`, PO Mode command 42
looking at the source to aid translation 46

M

`m`, PO Mode command 15
`M-,`, PO Mode command 23
`M-.`, PO Mode command 23
`M-A`, PO Mode command 48
`M-s`, PO Mode command 47
`M-S`, PO Mode command 47
marking strings for translation 22
moving by fuzzy entries 39
moving by obsolete entries 41
moving by translated entries 38
moving by untranslated entries 40
moving through a PO file 14

N

`n`, PO Mode command 14, 15
`next-error`, stepping through PO file validation results 14
`normalize`, PO Mode command 48

O

o, PO Mode command 41
O, PO Mode command 41
obsolete active entry 41

P

p, PO Mode command 14, 15
pending subedits 46
po-auto-edit-with-msgid, PO Mode variable 42
po-auto-fuzzy-on-edit, PO Mode variable 39
po-auto-select-on-unfuzzy, PO Mode variable 39
po-confirm-and-quit, PO Mode command 13
po-consider-as-auxiliary, PO Mode command 48
po-consider-source-path, PO Mode command 47
po-current-entry, PO Mode command 15
po-cycle-auxiliary, PO Mode command 48
po-cycle-source-reference, PO Mode command 47
po-edit-comment, PO Mode command 44
po-edit-msgstr, PO Mode command 42
po-exchange-location, PO Mode command ... 16
po-fade-out-entry, PO Mode command ... 39, 41
po-first-entry, PO Mode command 15
po-help, PO Mode command 14
po-ignore-as-auxiliary, PO Mode command 48
po-ignore-source-path, PO Mode command .. 47
po-kill-comment, PO Mode command 44
po-kill-msgstr, PO Mode command 40, 42
po-kill-ring-save-comment, PO Mode command 44
po-kill-ring-save-msgstr, PO Mode command 42
po-last-entry, PO Mode command 15
po-mark-translatable, PO Mode command ... 24
po-msgid-to-msgstr, PO Mode command 42
po-next-entry, PO Mode command 15
po-next-fuzzy-entry, PO Mode command 39
po-next-obsolete-entry, PO Mode command 41
po-next-translated-entry, PO Mode command 39
po-next-untranslated-entry, PO Mode command 40
po-normalize, PO Mode command 12, 16
po-other-window, PO Mode command 14
po-pop-location, PO Mode command 15
po-previous-entry, PO Mode command 15
po-previous-fuzzy-entry, PO Mode command 39
po-previous-obsolete-entry, PO Mode command 41
po-previous-translated-entry, PO Mode command 39
po-previous-untransted-entry, PO Mode command 40
po-push-location, PO Mode command 15
po-quit, PO Mode command 13
po-select-auxiliary, PO Mode command 48
po-select-mark-and-mark, PO Mode command 24
po-select-source-reference, PO Mode command 47
po-statistics, PO Mode command 14
po-subedit-abort, PO Mode command 45
po-subedit-cycle-auxiliary, PO Mode command 45
po-subedit-exit, PO Mode command 45
po-subedit-mode-hook, PO Mode variable 44
po-tags-search, PO Mode command 23
po-undo, PO Mode command 13
po-unfuzzy, PO Mode command 39
po-validate, PO Mode command 14
po-yank-comment, PO Mode command 44
po-yank-msgstr, PO Mode command 43

Q

q, PO Mode command 13
Q, PO Mode command 13

R

r, PO Mode command 15
RET, PO Mode command 42

S

s, PO Mode command 47
S, PO Mode command 47
starting a string translation 42
string normalization in entries 16
subedit minor mode 45

T

t, PO Mode command 38, 39
T, PO Mode command 38, 39
TAB, PO Mode command 39
'TAGS', and marking translatable strings 23

U

u, PO Mode command 40
U, PO Mode command 40
use the source, Luke 46
using obsolete translations to make new entries 43
using translation compendia 49

V

V, PO Mode command 13, 14

W

w, PO Mode command 42
W, PO Mode command 44

X

x, PO Mode command 15, 16

Y

y, PO Mode command 42, 43
Y, PO Mode command 44

Z

z, PO Mode command 39
Z, PO Mode command 39

Autoconf Macro Index

A

AM_GNU_GETTEXT 111
AM_ICONV 112

General Index

_

_, a macro to mark strings for translation 22
_nl_msg_cat_cntr 93

A

'ABOUT-NLS' file 78
'acconfig.h' file 108
accumulating translations 49
'aclocal.m4' file 108
adding keywords, xgettext 29
ambiguities 19
apply a filter to translations 58
apply command to all translations in a catalog 69
attribute manipulation 65
attribute, fuzzy 39
attributes of a PO file entry 39
attributes, manipulating 51
autoconf macros for gettext 111
auxiliary PO file 48
available translations 78
awk 124

B

backup old file, and msgmerge program 37
bash 117
bibliography 133
big picture 5
bind_textdomain_codeset 84
bug report address 1

C

C and C-like languages 116
C trigraphs 29
c-format flag 11
c-format, and xgettext 25
catalog encoding and msgexec output 69
catclose, a catgets function 81
catgets, a catgets function 80
catgets, X/Open specification 80
catopen, a catgets function 80
character encoding 4
charset conversion at runtime 83
charset of PO files 34
check format strings 71
checking of translations 51
clisp 119
clisp C sources 120
codeset 4
comments in PO files 12
Common Lisp 119
compare PO files 64
comparison of interfaces 91
compatibility with X/Open msgfmt 72
compendium 49
compendium, creating 49
concatenate PO files 51
concatenating PO files into a compendium 49
concatenation of strings 20
convert binary message catalog into PO file 73
convert translations to a different encoding 54
converting a package to use gettext 102
country codes 140
create new PO file 32
creating a new PO file 32
creating compendia 49
currency symbols 4

D

date format 4
dcngettext 86
debugging messages marked as format strings 29
dialect 51
disabling NLS 111
dngettext 86
domain ambiguities 82
duplicate elimination 51
duplicate removal 60

E

editing comments in PO files 44
editing translations 41
Emacs Lisp 120
encoding 4
encoding conversion 51
encoding conversion at runtime 83
encoding for your language 34
encoding list 34
encoding of PO files 34
evolution of packages 8
extracting parts of a PO file into a compendium 50

F

file format, '.mo' ... 75
file format, '.po' ... 10
files, '.po' and '.mo' ... 5
files, '.pot' ... 7
filter messages according to attributes ... 65
find common messages ... 62
force use of fuzzy entries ... 72
format strings ... 25
Free Pascal ... 125
fuzzy entries ... 39
fuzzy flag ... 11

G

gawk ... 124
generate binary message catalog from PO file ... 70
generate translation catalog in English ... 67
gettext files ... 105
gettext installation ... 10
gettext interface ... 81
gettext vs catgets ... 91
gettext, a programmer's view ... 81
'gettext.h' file ... 111
gettextize program, usage ... 104
GUI programs ... 89

H

hash table, inside MO files ... 75
he, she, and they ... 1
header entry of a PO file ... 33
help option ... 20
history of GNU gettext ... 132

I

i18n ... 2
importing PO files ... 17
include file 'libintl.h' ... 6, 18, 91, 111
initialization ... 18
initialize new PO file ... 32
initialize translations from a compendium ... 50
installing gettext ... 10
interface to catgets ... 80
internationalization ... 2
inttypes.h ... 21
ISO 3166 ... 140
ISO 639 ... 134

J

Java ... 123
Java mode, and msgfmt program ... 70
Java mode, and msgunfmt program ... 73
Java, string concatenation ... 21

K

keyboard accelerator checking ... 72

L

l10n ... 2
language codes ... 134
language selection ... 79
language selection at runtime ... 93
large package ... 82
libiconv library ... 113
libintl for Java ... 124
libintl library ... 112
librep Lisp ... 121
'LINGUAS' file ... 106
link with 'libintl' ... 7
Linux ... 5, 7, 34
Lisp ... 119
list of translation teams, where to find ... 33
locale facet, LC_ALL ... 18
locale facet, LC_COLLATE ... 19
locale facet, LC_CTYPE ... 4, 18, 19
locale facet, LC_MESSAGES ... 5, 19
locale facet, LC_MONETARY ... 4, 19
locale facet, LC_NUMERIC ... 4, 19
locale facet, LC_RESPONSES ... 19
locale facet, LC_TIME ... 4, 19
locale facets ... 4
locale program ... 34
localization ... 2

M

magic signature of MO files ... 75
'Makevars' file ... 106
manipulating PO files ... 51
marking string initializers ... 26
marking strings that require translation ... 21
marking strings, preparations ... 19
marking translatable strings ... 6
menu entries ... 89
menu, keyboard accelerator support ... 72
merge PO files ... 51
merging two PO files ... 51
message catalog files location ... 83
messages ... 5
migration from earlier versions of gettext ... 102
mnemonics of menu entries ... 72
MO file's format ... 75

modify message attrributes 66
`msgattrib` program, usage 65
`msgcat` program, usage 51
`msgcmp` program, usage 64
`msgcomm` program, usage 62
`msgconv` program, usage 54
`msgen` program, usage 67
`msgexec` program, usage 69
`msgfilter` filter and catalog encoding 58
`msgfilter` program, usage 58
`msgfmt` program, usage 70
`msggrep` program, usage 55
`msgid` 11
`msgid_plural` 12
`msginit` program, usage 32
`msgmerge` program, usage 36
`msgstr` 11
`msgunfmt` program, usage 73
`msguniq` program, usage 60
multi-line strings 17

N

`N_`, a convenience macro 91
Native Language Support 3
Natural Language Support 3
newlines in PO files 12
`ngettext` 85
NLS 3
`no-c-format` flag 11
`no-c-format`, and `xgettext` 25
`nplurals`, in a PO file header 86
number format 4

O

Object Pascal 125
obsolete entries 40
optimization of `gettext` functions 90
orthography 51
output to stdout, `xgettext` 28
overview of `gettext` 5

P

package and version declaration in '`configure.in`' 107
package build and installation options 78
package maintainer's view of `gettext` 102
paragraphs 20
Pascal 125
Perl 128
PHP 129
Pike 129
plural form formulas 86
plural forms 84
plural forms, in MO files 75
`plural`, in a PO file header 86
PO files' format 10
PO mode (Emacs) commands 13
PO template file 28
portability problems with `sed` 58
'`POTFILES.in`' file 105
preparing programs for translation 18
problems with `catgets` interface 81
programming languages 114
Python 118

Q

quotation marks 35, 106
quote characters, use in PO files 35

R

related reading 133
RST 130

S

scripting languages 114
search messages in a catalog 55
selecting message language 79
sentences 19
setting up `gettext` at build time 78
setting up `gettext` at run time 79
several domains 82
sex 1
sgettext 89
she, he, and they 1
shell scripts 117
Smalltalk 122
sorting `msgcat` output 53
sorting `msgmerge` output 38
sorting `msgunfmt` output 74
sorting output of `xgettext` 30
specifying plural form in a PO file 86
standard output, and `msgcat` 52
standard output, and `msgmerge` program 36
string concatenation 20
string normalization in entries 16
style 19
supported languages, `xgettext` 28

T

Tcl 127

Tcl mode, and `msgfmt` program 70
Tcl mode, and `msgunfmt` program 73
template PO file 7
testing '`.po`' files for equivalence 30
Tk's scripting language 127
translated entries 38
translating menu entries 89
translation aspects 3
Translation Matrix 78
Translation Project 1
turning off NLS support 111
tutorial of `gettext` usage 5

U

unify duplicate translations 60
untranslated entries 40
update translations from a compendium 50
upgrading to new versions of `gettext` 102

V

version control for backup files, `msgmerge` 37

W

`wxWindows` library 126

X

`xargs`, and output from `msgexec` 69
`xgettext` program, usage 28
`xmodmap` program, and typing quotation marks 35

Y

YaST2 scripting language 126
YCP 126

Table of Contents

1 Introduction 1

1.1 The Purpose of GNU `gettext` 1
1.2 I18n, L10n, and Such 2
1.3 Aspects in Native Language Support 3
1.4 Files Conveying Translations 5
1.5 Overview of GNU `gettext` 5

2 PO Files and PO Mode Basics 10

2.1 Completing GNU `gettext` Installation 10
2.2 The Format of PO Files 10
2.3 Main PO mode Commands 13
2.4 Entry Positioning 14
2.5 Normalizing Strings in Entries 16

3 Preparing Program Sources 18

3.1 Triggering `gettext` Operations 18
3.2 Preparing Translatable Strings 19
3.3 How Marks Appear in Sources 21
3.4 Marking Translatable Strings 22
3.5 Special Comments preceding Keywords 25
3.6 Special Cases of Translatable Strings 26

4 Making the PO Template File 28

4.1 Invoking the `xgettext` Program 28
4.1.1 Input file location 28
4.1.2 Output file location 28
4.1.3 Choice of input file language 28
4.1.4 Operation mode 29
4.1.5 Language=C/C++ specific options 29
4.1.6 Output details 30
4.1.7 Informative output 31

5 Creating a New PO File 32

5.1 Invoking the `msginit` Program 32
5.1.1 Input file location 32
5.1.2 Output file location 32
5.1.3 Output details 32
5.1.4 Informative output 33
5.2 Filling in the Header Entry 33

6 Updating Existing PO Files 36

6.1 Invoking the `msgmerge` Program 36
6.1.1 Input file location 36
6.1.2 Operation mode 36
6.1.3 Output file location 36
6.1.4 Output file location in update mode 37
6.1.5 Operation modifiers 37
6.1.6 Output details 37
6.1.7 Informative output 38
6.2 Translated Entries 38
6.3 Fuzzy Entries 39
6.4 Untranslated Entries 40
6.5 Obsolete Entries 40
6.6 Modifying Translations 41
6.7 Modifying Comments 44
6.8 Details of Sub Edition 45
6.9 C Sources Context 46
6.10 Consulting Auxiliary PO Files 48
6.11 Using Translation Compendia 49
6.11.1 Creating Compendia 49
6.11.1.1 Concatenate PO Files 49
6.11.1.2 Extract a Message Subset from a PO File 50
6.11.2 Using Compendia 50
6.11.2.1 Initialize a New Translation File 50
6.11.2.2 Update an Existing Translation File ... 50

7 Manipulating PO Files 51

7.1 Invoking the `msgcat` Program 51
7.1.1 Input file location 52
7.1.2 Output file location 52
7.1.3 Message selection 52
7.1.4 Output details 53
7.1.5 Informative output 53
7.2 Invoking the `msgconv` Program 54
7.2.1 Input file location 54
7.2.2 Output file location 54
7.2.3 Conversion target 54
7.2.4 Output details 54
7.2.5 Informative output 55
7.3 Invoking the `msggrep` Program 55
7.3.1 Input file location 55
7.3.2 Output file location 56
7.3.3 Message selection 56
7.3.4 Output details 57
7.3.5 Informative output 58
7.4 Invoking the `msgfilter` Program 58
7.4.1 Input file location 58

7.4.2 Output file location 58
7.4.3 The filter 58
7.4.4 Useful *filter-option*s when the *filter* is 'sed' 59
7.4.5 Output details 59
7.4.6 Informative output 60
7.5 Invoking the msguniq Program 60
7.5.1 Input file location 60
7.5.2 Output file location 61
7.5.3 Message selection 61
7.5.4 Output details 61
7.5.5 Informative output 62
7.6 Invoking the msgcomm Program 62
7.6.1 Input file location 62
7.6.2 Output file location 63
7.6.3 Message selection 63
7.6.4 Output details 63
7.6.5 Informative output 64
7.7 Invoking the msgcmp Program 64
7.7.1 Input file location 64
7.7.2 Operation modifiers 64
7.7.3 Informative output 65
7.8 Invoking the msgattrib Program 65
7.8.1 Input file location 65
7.8.2 Output file location 65
7.8.3 Message selection 65
7.8.4 Attribute manipulation 66
7.8.5 Output details 66
7.8.6 Informative output 67
7.9 Invoking the msgen Program 67
7.9.1 Input file location 67
7.9.2 Output file location 67
7.9.3 Output details 68
7.9.4 Informative output 68
7.10 Invoking the msgexec Program 69
7.10.1 Input file location 69
7.10.2 Informative output 69

8 Producing Binary MO Files 70

8.1 Invoking the msgfmt Program 70
8.1.1 Input file location 70
8.1.2 Operation mode 70
8.1.3 Output file location 70
8.1.4 Output file location in Java mode 71
8.1.5 Output file location in Tcl mode 71
8.1.6 Input file interpretation 71
8.1.7 Output details 72
8.1.8 Informative output 72
8.2 Invoking the msgunfmt Program 73

8.2.1 Operation mode ... 73
8.2.2 Input file location ... 73
8.2.3 Input file location in Java mode ... 73
8.2.4 Input file location in Tcl mode ... 73
8.2.5 Output file location ... 74
8.2.6 Output details ... 74
8.2.7 Informative output ... 74
8.3 The Format of GNU MO Files ... 75

9 The User's View ... 78

9.1 The Current 'ABOUT-NLS' Matrix ... 78
9.2 Magic for Installers ... 78
9.3 Magic for End Users ... 79

10 The Programmer's View ... 80

10.1 About catgets ... 80
10.1.1 The Interface ... 80
10.1.2 Problems with the catgets Interface?! ... 81
10.2 About gettext ... 81
10.2.1 The Interface ... 81
10.2.2 Solving Ambiguities ... 82
10.2.3 Locating Message Catalog Files ... 83
10.2.4 How to specify the output character set gettext uses ... 83
10.2.5 Additional functions for plural forms ... 84
10.2.6 How to use gettext in GUI programs ... 89
10.2.7 Optimization of the *gettext functions ... 90
10.3 Comparing the Two Interfaces ... 91
10.4 Using libintl.a in own programs ... 92
10.5 Being a gettext grok ... 93
10.6 Temporary Notes for the Programmers Chapter ... 94
10.6.1 Temporary - Two Possible Implementations ... 94
10.6.2 Temporary - About catgets ... 94
10.6.3 Temporary - Why a single implementation ... 95
10.6.4 Temporary - Notes ... 95

11 The Translator's View ... 96

11.1 Introduction 0 ... 96
11.2 Introduction 1 ... 96
11.3 Discussions ... 97
11.4 Organization ... 98
11.4.1 Central Coordination ... 99
11.4.2 National Teams ... 99
11.4.2.1 Sub-Cultures ... 100
11.4.2.2 Organizational Ideas ... 100
11.4.3 Mailing Lists ... 100
11.5 Information Flow ... 101

12 The Maintainer's View 102

12.1 Flat or Non-Flat Directory Structures 102
12.2 Prerequisite Works 102
12.3 Invoking the `gettextize` Program 104
12.4 Files You Must Create or Alter 105
12.4.1 '`POTFILES.in`' in '`po/`' 105
12.4.2 '`LINGUAS`' in '`po/`' 106
12.4.3 '`Makefile`' pieces in '`po/`' 106
12.4.4 '`configure.in`' at top level 107
12.4.5 '`config.guess`', '`config.sub`' at top level 108
12.4.6 '`aclocal.m4`' at top level 108
12.4.7 '`acconfig.h`' at top level 108
12.4.8 '`Makefile.in`' at top level 109
12.4.9 '`Makefile.in`' in '`src/`' 110
12.4.10 '`gettext.h`' in '`lib/`' 111
12.5 Autoconf macros for use in '`configure.in`' 111
12.5.1 AM_GNU_GETTEXT in '`gettext.m4`' 111
12.5.2 AM_ICONV in '`iconv.m4`' 112

13 Other Programming Languages 114

13.1 The Language Implementor's View 114
13.2 The Programmer's View 115
13.3 The Translator's View 115
13.4 The Maintainer's View 115
13.5 Individual Programming Languages 116
13.5.1 C, C++, Objective C 116
13.5.2 sh - Shell Script 117
13.5.3 bash - Bourne-Again Shell Script 117
13.5.4 Python 118
13.5.5 GNU clisp - Common Lisp 119
13.5.6 GNU clisp C sources 120
13.5.7 Emacs Lisp 120
13.5.8 librep 121
13.5.9 GNU Smalltalk 122
13.5.10 Java 123
13.5.11 GNU awk 124
13.5.12 Pascal - Free Pascal Compiler 125
13.5.13 wxWindows library 126
13.5.14 YCP - YaST2 scripting language 126
13.5.15 Tcl - Tk's scripting language 127
13.5.16 Perl 128
13.5.17 PHP Hypertext Preprocessor 129
13.5.18 Pike 129
13.6 Internationalizable Data 130
13.6.1 POT - Portable Object Template 130
13.6.2 Resource String Table 130
13.6.3 Glade - GNOME user interface description 131

14 Concluding Remarks 132
14.1 History of GNU `gettext` 132
14.2 Related Readings 133

Appendix A Language Codes 134

Appendix B Country Codes 140

Program Index 148

Option Index 149

Variable Index 154

PO Mode Index 155

Autoconf Macro Index 158

General Index 159