GNU gettext tools, version 0.18.3

Native Language Support Library and Tools
Edition 0.18.3, 16 June 2013

日本語訳: **Ayanokoji Takesi**(綾小路剛志)
日本語訳: **Rue. SATOH**(佐藤竜一) / **Koichi KIMURA**(木村浩一)
Ulrich Drepper
Jim Meyering
François Pinard
Bruno Haible

Table of Contents

1 イントロダクション

このチャプターでは、GNU gettextが作られた目的と、フリーの翻訳プロジェクトについて説明します。それから、ネイティブ言語サポート (NLS: Native Language Support) にまつわる広義の概念をいくつか説明します。また国や文化の差異を他の側面から考慮した際に、翻訳したメッセージをプログラムに適用することが、どのような位置づけになるかを説明します。さらに翻訳に使用されるファイルを説明し、最初の翻訳において様々なツールがそれらのファイルをどのように生成、相互作用するのか、そして通常のメンテナンスサイクルにおいてどのように処理されるかについて説明します。

このマニュアルでは、プログラマーやメンテナーを指すときには彼という言葉を使用します。そして翻訳者を指すときは彼女、翻訳されたプログラムをインストールする人やエンドユーザーのことを指すときは彼らという言葉を使用します。これの目的はドキュメントを明解にするのが唯一の目的であり、決して各々の役割が男性、もしくは女性に適しているという意味ではまったくありません。あなたが想像するように、GNU gettextは性別、人種、宗教、国籍に関わらず、コンピュータを使用する人にとって有用なものなのです!

提案や訂正は下記にメールしてください:

Internet address:
bug-gnu-gettext@gnu.org

メールのメッセージに、マニュアルのバージョン番号と更新日付を記入してください。

1.1 GNU gettextの目的

プログラムは通常、英語でドキュメントされており、実行時にユーザーと相互作用する場合にも英語が使用されます。これは GNU ソフトウェアに限らず、多くのフリーソフトウェアにも当てはまる話です。開発者やメンテナー、すべての国々のユーザー同士でコミュニケーションする場合、共通の言語を使用するほうが便利です。一方、ほとんどの人は英語より自分の母国語を好み、日々の仕事にも可能な限り母国語を使用しています。単純に言うと、ほとんどの人は英語より母国語がコンピューターのスクリーンに表示されるのを愛するのです。

しかし多くの人にとって、これは時間をかけて考える価値がないと片付けてしまう夢かもしれません。彼らは結局、この夢を実現できる自信がないのです。しかし希望を失わずに、組織を作った人たちもいます。この翻訳プロジェクトは、これらの希望を作業可能な形に形式化して、私たちのすべてが真の多言語機能を兼ね揃えたプログラムと呼べるものを手にするチャンスなのです。

GNU gettextは、他の多くのステップを構築する資産であり、翻訳プロジェクトにとって重要なステップです。このパッケージは、プログラマー、翻訳者、そしてユーザーにたいして、統合されたツールとドキュメントを提供します。特に GNU gettextユーティリティーは、他のパッケージに多言語化されたメッセージを生成するためのフレームワークを提供するツール群です。このツールには以下のものが含まれます:

- メッセージカタログをサポートするために、プログラムをどのように記述すべきかの規則。
- メッセージカタログのディレクトリーやファイル名の命名方式。
- 翻訳されたメッセージの取得をサポートする実行時ライブラリー。
- 翻訳可能なメッセージや、既に翻訳されたメッセージを取り扱うための、スタンドアローンプログラム群。
- 翻訳可能なメッセージが含まれたファイルの解析、生成をサポートするライブラリー。

- これらのセットを準備して最新の状態に保つための、Emacs[1] 用のスペシャルモード。

GNU gettextは、プログラムのソースを国際化する際の影響を最小化し、その影響を可能な限り小さく保つようにデザインされています。プログラムソースを見たときに、その影響が軽微、または少なくとも軽微に見えることにより、国際化が成功する可能性が高くなります。

翻訳プロジェクトは、翻訳プロジェクトの構造や手法を記述する手段としても、GNU gettext ディストリビューションを使用しています。これは、この文書が GNU gettextの技術面に限定して適切に記述された文書であることを超えるものであることを意味します。そうすることにより、翻訳者は翻訳作業を適切に行うために知る必要のあるすべてのことを、可能な限り単一の場所で見つけることができます。この補足的な文書により、プログラマーさらに好奇心のあるユーザーは、GNU gettextが翻訳プロジェクト以外の場所とどのように関連しているかを理解し、大きな絵を垣間見ることができます。

1.2 i18n、l10n、などなど

プログラムによるネイティブ言語サポートを議論する際に、2 つの長い単語が出現します。これらの単語には正確な意味づけがあり、このドキュメントでも使用する単語なので、ここで説明しておきましょう。使用される 2 つの長い単語とは、インターナショナリゼーション *(internationalization)* とローカリゼーション *(localization)* です。これらの単語を何度も何度も書くうちに、多くの人たちが *i18n* や *l10n* と記述するようになりました。この表現は単語の最初と最後の文字と、それらの文字の間にある文字数によって、それぞれの単語を略記したものです。しかしこのマニュアルではわかりやすくするために、略記を用いずに正式な単語を使用することにします。

インターナショナリゼーション *(*国際化*)* とは、プログラムやパッケージに含まれるプログラムのセットが多言語を認識しサポートする操作のことを指します。インターナショナリゼーションとは、英語の文字列でしかプログラムがを呼び出せないとか、英語以外の特定の言語の持つ習慣に縛られるといったことなく、同じ操作を一般的な方法で結びつけさせるための汎化のプロセスです。プログラムの開発者は、彼のプログラムをインターナショナライズするために様々なテクニックを使うことでしょう。それらのいくつかは標準化されています。そのうちの 1 つが GNU gettextなのです。Chapter 11 [Programmers], page 109 を参照してください。

ローカリゼーション *(*地域化*)* とは、それを行うことによりすでにインターナショナライズされた一連のプログラムが必要とするすべての情報を受けとることができ、任意のネイティブ言語や文化的な習慣に従った方法による入出力に適応させることができるような操作のことを意味します。ローカリゼーションとは、一般的なメソッドを実装済みのインターナショナライズされたプログラムを特定の方法で使用する、特化のプロセスです。プログラミング環境はプログラマーにたいして、実行時に設定できるいくつかの機能を提供します。翻訳対象の単位として同じネイティブ言語同士をまとめた、任意の国がもつ特定の文化的な習慣の形式的な説明を、その言語や国の *locale* と呼びます。ローカライズされたプログラムを使用するユーザーは、プログラムを実行する前にプログラムがどの locale を使用するべきかを、特定の環境変数に設定します。

実際には locale メッセージのサポートとは、特定の locale を形成する文化的なデータの単一のコンポーネントのことです。インターナショナライズされたソフトウェアを開発するプログラマーを支援するために、特定の locale に保存されたデータにアクセスできるルーチンと関数を提供するライブラリーがあります。誰かが特定の locale を参照する場合には、特定の locale に保存されているデータを参照することになります。同様に、プログラマーが “locale ルーチンへのアクセス” を参照するということは、すべての locale 情報にアクセスできる完全なルーチンの一式を参照するのと同じです。

[1] このマニュアルで Emacs という言葉を使用するときは、FSF Emacs とも呼ばれる GNU Emacs と XEmacs、そして Lucid Emacs を指します。

ネイティブ言語サポート (*Native Language Support*、または単に NLS) という表現は、インターナショナリゼーションとローカリゼーションの両方により、プログラムが多言語と相互作用できる動作や機能の全体を指すときに使用します。一言で言うと、インターナショナリゼーションとはローカリゼーションを可能にする操作とも言えます。

大まかに言うと、多言語によるメッセージを処理する場合、インターナショナリゼーションはプログラマーによって処理され、ローカリゼーションは翻訳者によって処理されるとも言えます。

1.3 ネイティブ言語サポートの側面

完全な多言語ディストリビューションを配付するためには、出力メッセージの翻訳以外に多くの事柄があります。

- 現在の GNU `gettext`が提供するのは、C プログラムが出力するメッセージを翻訳するための完全なツールセットです。しかし perl スクリプトとシェルスクリプトも同様に翻訳される必要があります。現在これらを翻訳するための方法があったとしても、本来あるべき形での統合はされていません。
- `autoconf`や `bison`のような一部のプログラムは、他のプログラム (またはスクリプト) を生成することができます。たとえプログラムを生成するプログラムがインターナショナライズされていても、生成されたプログラムは独自にインターナショナリゼーションする必要がありますが、これらの間接的に行われるインターナショナリゼーションは生成プログラムで自動化できるはずです。しかし実際は生成するプログラムと生成されるプログラムは、それぞれ独自にインターナショナライズされるのが極めて一般的です。
- 数は多くありませんが、プログラム自体に含まれた文字列とは別に、翻訳が必要となるテキストのテーブルを含むプログラムもあります。例えば RFC 1345 は、`recode`プログラムが実行時に文字を再構築するための、文字に対応する説明的な英語名を提供します。これらの説明的な名前は RFC から機械的に取り出されるため、RFC 自体を事前に翻訳する必要があります。
- 大抵のプログラムにはオプションを指定することができますが、多くの場合は英語を読める人にたいして説明的なものが使用されています。このようプログラムのオプションについても、翻訳されたバージョンを提供することを考慮する必要があります。
- 多くのプログラムは、本質的には翻訳可能な何かを、読み込み、解釈、コンパイルしたり、そのようなキーワードや識別子のテキストを含む入力ファイルによって何かを行ったり、応答を返したりします。例えば `gcc`に識別子の文字を区別できるようにさせたり、‘`rm -i`’が ‘`y`’や ‘`n`’ではない、翻訳された応答をユーザーから受け取れるようにしたいと思うかもしれません。最終的にプログラムのほとんどの出力が他言語によるものだったとしても、入力の構文やオプションに指定できる値などを、ローカライズ可能かそうでないかを決定したいと思うかもしれません。
- パッケージに付随するマニュアル、及びディストリビューションのドキュメントファイルも同様に、翻訳される可能性があります。マニュアルの翻訳と、それ以降のアップデートは、一般的にはそれ自体が主要な作業です。

すでに述べたように、翻訳とは locale の 1 つの側面に過ぎません。インターナショナリゼーションの他の側面にはシステムのサービスがあり、これは GNU `libc`により処理されます。国々の文化的な慣習を定義するための多くの属性があります。これらの属性には、国々のネイティブ言語に即した日付や、時刻書式と数値表記、通貨記号などが含まれます。これらの地域的なルールは、その国の locale と呼ばれます。locale とは、その国のネイティブな属性をサポートするために必要となる知識を表します。

国ごとの差異にしたがって locale を記述しなければならない、主要な領域がいくつかあります。以下のリストは、locale に関連したその他のタスクの適切なコンテキストにおいて、多言語メッセージを配置する手助けになるでしょう。詳細については GNU `libc`のマニュアルを参照してください。

文字とコードセット
: 米国や世界中の、英語を話す地域で最も一般的に使用されるコードセットは、ASCII コードセットです。しかしこのコードセットには、様々な locale で必要とされる文字が含まれていません。8 ビット ISO 8859-1 コードセットは主要なヨーロッパの言語で処理する必要がある特殊文字をほとんど持っているにもかかわらず、主要なヨーロッパの通貨を処理することができない等、多くの場合は ISO 8859-1 を選択するだけでは十分ではないのです。したがってそれぞれの locale は、使用するコードセットの選択と、そしてそのコードセットに対処するための適切な文字列処理ルーチンが必要になります。

通貨
: 通貨記号は国ごとに異なり、それぞれの通貨記号の使用する位置も異なります。それぞれの locale にたいするネイティブモードで、ソフトウェアはそれを意識させずに通貨の数字を表示できる必要があります。

日付
: 日付の書式は locale ごとに異なります。例えば 1994 年のクリスマスは、米国では 12/25/94 と記述し、オーストラリアでは 25/12/94、それ以外の国では ISO 8601 の日付書式を使用する、といった具合です。

  1 日の中で使用される時刻も、*hh*:*mm*、*hh*.*mm*、などのように記述されます。ある locale では時刻は AM/PM ではなく、24 時間制で指定する必要があります。しかも夏時間の補正は国ごとに大きく異なります。

数値
: 数値の表記は locale ごとに異なります。以下はそれぞれの locale に対応する、正しい数値表記の例です:

```
12,345.67       English
12.345,67       German
 12345,67       French
1,2345.67       Asia
```

  メートル法とポンドヤード法のように異なる単位系を使用したり、それらの変種で数値表記する方法を採用しているプログラムもあります。

メッセージ
: locale による言語サポートにおいて、最も明確な領域です。GNU `gettext`は、locale でのメッセージのサポートという領域において、ソフトウェアがユーザーとコミュニケーションするときに使用する言語を、開発者とユーザーが簡単に変更する手段を提供します。

これらの文化的な慣習の領域は、*locale* カテゴリー *(locale categories)* と呼ばれます。この用語は、*locale* の側面 *(locale aspects)* や *locale* 機能のカテゴリー *(locale feature categories)* といった用語よりも劣っているのが残念です。なぜならそれぞれの “locale カテゴリー” は、ローカリゼーションが要求される、ある領域やタスクについて記述するからです。そのような領域や特定の文化にたいして、文化的な慣習を説明する具体的なデータも *locale* カテゴリーと呼ばれます。この意味では、locale とは、コードセットを定義する locale カテゴリー、数値の書式を定義する locale カテゴリー、翻訳されたメッセージを定義する locale カテゴリー、などのように、いくつかの locale カテゴリーから構成されているといえます。

メッセージ処理以外の locale コンポーネントは、標準 ISO C と POSIX:2001 標準 (SUSV3 specification とも呼ばれる) です。GNU `libc`はこれを完全に実装しており、その他の現代的なシステムも、欠けているコンポーネントにたいする、必要最小限のより実用的なサポートを提供しています。

1.4 翻訳を伝達するファイル

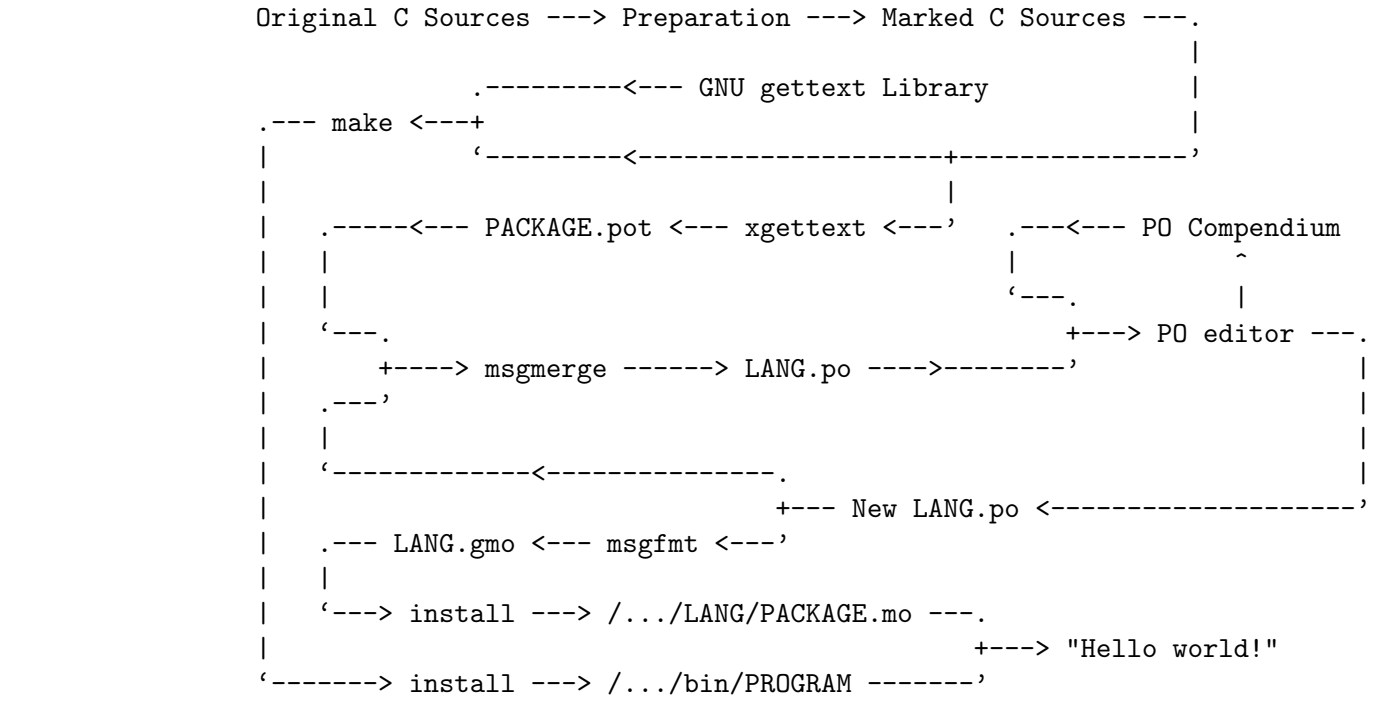
`.po`ファイルの PO は Portable Object の頭文字であり、`.mo`ファイルの MO は Machine Object の頭文字です。このパラダイムは PO ファイルのフォーマットと同様に、Uniforum で開発された NLS 標準にもとづき、Sun の Solaris システムで実装されたものです。

PO ファイルは人間が読んだり編集することを意図しており、あるパッケージの特定のターゲット言語のための、翻訳可能なオリジナル文字列の集まりです。1 つの言語にたいして、1 つの PO ファイルが割り当てられます。パッケージが複数の言語をサポートする場合、サポートする言語ごとに PO ファイルを持ち、パッケージごとにパッケージがサポートする言語ごとの PO ファイルを持っています。これらの PO ファイルは `xgettext`プログラムによって作成され、アップデートや更新は msgmerge プログラムによって行われます。`xgettext`プログラムは C ファイルからマークされたメッセージを抽出し、空の翻訳文字列で初期化された PO ファイルを作成します。`msgmerge`プログラムは、リリースの間に変更されたソースファイルにたいして、不要なエントリーのコメント化や新しい文字列の初期化、および参照するソース行を更新したりします。この種のファイルは配布物中にの`.pot`という拡張子のファイルとして含まれ、書式は PO ファイルと同じです。

MO ファイルは、プログラムが読み込むことを意図した、バイナリーのフォーマットのファイルです。ネイティブ言語サポートの一部として、MO ファイルを作成したり処理することのできる既存のシステムもいくつかありますが、MO ファイルのフォーマットがシステムごとに異なっているため可搬性がありません。また、これらのシステムが提供する既存のツールは、GNU `gettext`のすべての機能をサポートしていません。そのため GNU `gettext`は、MO ファイルに独自のフォーマットを使用しています。拡張子`.gmo`が実際の MO ファイルで、これらのファイルは GNU のフォーマットを使用しています。

1.5 GNU `gettext`の概要

以下は、GNU `gettext`で処理されるファイル群の関連と、それらを処理するツールをまとめたダイアグラムです。以降の詳細な説明は、このダイアグラムを見ながら読み進めてください。これらのファイルやツールの相互作用への明確な理解は、プログラマー、翻訳者、メンテナーにとって確実に役に立つものです。



プログラマーが自分のパッケージに GNU gettextを導入するときに最初に行うことは、C ソース中のどの文字列が翻訳可能で、どの文字列が翻訳不可かを識別することです。この退屈な作業は、emacs の PO mode を使用することにより多少は快適になりますが、C ソースを編集するのに、あなた自身が慣れ親しんだものを使用することもできます。その他に必要となる標準的な変更としては、翻訳用のライブラリーを正しく初期化することなどです。これらに関する詳細は Chapter 4 [Sources], page 19 を参照してください。

新しく記述するソフトウェアについては、ソフトウェアを記述するときにそのような文字列をマークできますし、そうするべきでしょう。このような文字列にたいする gettextのアプローチ方法としては、手始めに各ファイルの先頭または中心的なヘッダーファイルに、以下の行を追加するという、非常に簡単なものです:

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

これで、インターナショナリゼーションのためのソースの準備ができました。後で実際に gettextを使う準備ができたら、これらを以下の定義で置き換えてください:

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

libintl.aと libintl.soにリンクする必要もあります。GNU システムでは、gettextライブラリーの関数は GNU libc にすでに含まれているので、libintlにリンクする必要がないことに注意してください。

一度 C ソースが変更されると、翻訳可能なすべての文字列を検索、抽出して PO template ファイルに出力するのに、xgettextが使用されます。この *package*.potファイルには、オリジナルのプ

ログラムのすべての文字列が含まれています。これらの文字列は、その文字列が C ソース中で使用されている場所へのポインターを持っており、すべての翻訳文字列は空文字列に初期化されています。`.pot`の`t`という文字は、このファイルがテンプレート (Template) の PO ファイルであり、まだ特定の言語用ではないことを示します。どのように`xgettext`プログラムが呼び出されるかについては、Section 5.1 [xgettext Invocation], page 33 を参照してください。もしあなたが本当に怠け者の場合、少し手間をかけてディストリビューション全体をセットアップするのに興味があるかもしれません (Chapter 13 [Maintainers], page 135 を参照してください)。この方法では、`xgettext`コマンドをタイプするのを省略して`make`とタイプするだけで、自動的に適切なものを生成することができます。

最初はまだ *`lang`*`.po`がないので、`msgmerge`のステップはとばして、単に *`package`*`.pot`が *`lang`*`.po`としてコピーされます。ここで *lang* は対象となる言語です。詳細については、Chapter 6 [Creating], page 41 を参照してください。

次はメッセージの最初の翻訳です。翻訳それ自身が全体として今だ人手に頼らねばならないものであり、その複雑さはこのマニュアルの取り扱う範囲を超えるものです。翻訳チームに連絡したり、チームの一員になって、同じネイティブ言語を作業対象とする他の人たちとあなたの翻訳を共有する方法等、いくつかのヒントについては、このマニュアルの他のチャプターで触れています (Chapter 12 [Translators], page 126 を参照してください)。

PO ファイル *`lang`*`.po`に翻訳したメッセージを追加するときに、PO ファイル編集用のエディター (Chapter 8 [Editing], page 51 を参照してください) を使用していない場合は、PO ファイルのフォーマットに合わせて作業したり、文字列を引用符で括る規則など (Chapter 3 [PO Files], page 13 を参照してください) について自分で気を遣わなければなりません。これは不可能な作業ではなく、実際に 1995 年頃には多くの人が PO ファイルを取り扱っていた方法です。一方、PO ファイルエディターは、PO ファイルエディター自身の使い方を覚える必要はありますが、あなたにかわってエディターが PO ファイルに関する詳細を取り扱ってくれます。

既に何らかの翻訳が Compendium PO ファイルに保存されている場合、翻訳者は PO モードを使って翻訳されていないエントリーを Compendium から初期化したり、翻訳を選択して Compendium に保存したり更新することができます (Section 8.4 [Compendium], page 65 を参照してください)。Compendium ファイルは、翻訳チームのメンバー間で共有するように意図されたものです。

プログラムやプログラムのパッケージは、ユーザーがバグ報告や改良のための提案をして、メンテナーが様々な方法でプログラムを変更して対応するという、動的な性質を持ちます。パッケージがすでにインターナショナライズされているという事実により、メンテナーがパッケージに文字列を追加したり、すでに翻訳された文字列を変更することをためらうようにさせるべきではありません。彼らは、彼ら自身がスムーズに作業できるようにベストを尽くすだけです。メンテナーはすでに負荷の掛かった双肩に、翻訳に関する心配事を背負いこまないようにしてください。そして翻訳者はプログラミングの心配事からは自由でいるようにしてください。

メンテナーが心配すべきなのは、文字列が翻訳されるべきときに翻訳可能であるように文字列をマークすることであり、文字列がいつ翻訳されるかについては、適切な時がくれば翻訳されるものだと割り切るべきです。`xgettext`は、時間をかけて進化してきた *`package`*`.pot`を再び構築し、その結果、翻訳を含んだ *`lang`*`.po`は徐々に古くなっていきます。

翻訳者 (そしてメンテナー) にとって重要なのは、パッケージの翻訳はパッケージが誕生した時に 1 度行えばよいというものではなく、パッケージの生涯において繰り替えされる継続的なプロセスだと理解することです。あるパッケージにたいして最初の翻訳を行った後、時々手入れをすることが必要です。なぜなら翻訳が必要な新しい未翻訳の文字列が出現することにより、翻訳された文字列があちこちで古くなっていくからです。

`msgmerge`プログラムは、すでに存在する *`lang`*`.po`ファイルを、`xgettext`で最新の C ソースから抽出された、より新しい *`package`*`.pot`テンプレートファイルと比較して更新するという目的を持っ

ています。更新の処理はプログラムの変更により変更された、C ソース中の文字列の位置にたいする参照を調整します。同様に、msgmergeはすでに翻訳されているがプログラムのソースに存在しなくなった、古い翻訳のコメントアウトも行います (Section 8.3.8 [Obsolete Entries], page 58 を参照してください)。そして最後に新しい文字列を未翻訳の文字列として、結果である PO ファイルに挿入します (Section 8.3.7 [Untranslated Entries], page 57 を参照してください)。msgmerge が実際に何を行うかについては、Section 7.1 [msgmerge Invocation], page 47 を参照してください。

目的に至る経路と手段が何であれ、翻訳のためのすべての文字列を提供する更新された *lang*.po がゴールなのです。

PO ファイルが変動し流動する一時的なものであるという性質は、翻訳というゲームでの不可欠な部分であり、よく理解して受け入れる必要があります。翻訳プロジェクトに参加する人はこの性質に苦労し、他の翻訳プロジェクトのメンバーに苦労をかけることもあるのです! 特にメンテナーは、たとえ最近は更新されていないディストリビューションでも、翻訳チームに早く仕事を終えるようにプレッシャーを与えず、リラックスして利用可能でオフィシャルなすべての PO ファイルをディストリビューションに含めましょう。プレッシャーを与えるのはむしろ、特定の言語を話すコミュニティーのユーザーなので、メンテナー自身も安心して翻訳ファイルの妥当性を考慮するべきです。一方翻訳者は、パッケージがオフィシャルのディストリビューションに向けた事前テストを行っているときに、自分が担当する PO ファイルを合理的に更新する事を試みる必要があります。

1 度 PO ファイルが完成して信頼できる物になると、PO ファイルはmsgfmtプログラムによって、パッケージのプログラムが実行時に必要な時はいつでも効率的に翻訳を取得できるよう、マシン向けのフォーマットに変換されます (Section 10.3 [MO Files], page 106 を参照してください)。msgfmt プログラムのすべての実行モードについては、Section 10.1 [msgfmt Invocation], page 100 を参照してください。

最後に、変更およびマークされた C ソースがコンパイルされて、GNU gettextライブラリーとリンクされます。これは通常、プロジェクトのための適切な Makefileと共に、makeコマンドを実行することにより行われ、結果としてユーザーが見つけることのできる場所に実行可能ファイルがインストールされます。MO ファイル自身も適切にインストールされる必要があります。これで適切な環境変数 (Section 2.3 [Setting the POSIX Locale], page 10 を参照してください) をセットすると、プログラムを実行すればいつでも自分で自動的にローカライズするようになります。

このマニュアルの残りの部分では、上述の様々なステップを掘り下げて説明することを目的とします。

2 ユーザーの視点

最近では、ユーザーがコンピューターにログインしたときには通常、プログラムがネイティブ言語でメッセージを表示するのを目にすることができます – 少なくともフリーソフトウェアや GNU プロジェクトに関わる人が少ない Hindi や Filipino などの言語ではない、French や German などの言語による活発なフリーソフトウェアコミュニティーのユーザーは目にすることができるでしょう。

これはどのような仕組みで動くのでしょう? ユーザーはどのようにして、プログラムで使用する言語にたいして影響を与えることができるのでしょうか? このチャプターではこれらの疑問にお答えします。

2.1 オペレーティングシステムのインストール

デフォルトで使用する言語は、すでにオペレーティングシステムのインストールの時点で決定されている場合があります。オペレーティングシステムがインストールされるときは通常、インストーラーがインストール中に使用する言語とは別に、インストールされるシステムで使用する言語を尋ねます。言語を 1 度しか尋ねない OS インストーラーもあります。

これにより、すべてのユーザーにたいする、システム全体でのデフォルト言語が決定されます。追加の言語としてデフォルト言語以外のローカリゼーションを指定できるインストーラーもあります。たとえば KDE(K Desktop Environment) のローカリゼーションや OpenOffice.org は、言語ごとにインストールできるパッケージが個別にバンドルされています。

これはマシンの使用目的を考える、よい機会です。個人的に使用するマシンの場合、追加のローカリゼーションはおそらく必要ありません。国際的なつながりをもつ組織や企業で使用するマシンの場合、ゲストユーザーのことも考えられます。海外から 1 週間程度の予定でゲストを迎える場合、彼のお好みの locale は何でしょうか? そのコストがディスクスペースを少し余分に消費するだけならば、あらかじめ追加のローカリゼーションをインストールする価値があるかもしれません。

システム全体のデフォルト言語は、新しいアカウントを作成するときの locale 設定で使用されます。しかしユーザーは同じマシンの他のユーザーとは異なる、自分自身の locale 設定を持つことができます。次のセクションで説明するようにユーザーは通常、最初のログイン後に自分の locale を指定することができます。

2.2 GUI プログラムを使用した locale のセッティング

すぐに利用可能なプログラムは、“デスクトップ環境” とも呼ばれるユーザーのデスクトップで、それにはウィンドウマネージャーやウェブブラウザー、それにテキストエディターなどが含まれます。一般的なデスクトップとしては KDE、GNOME、Xfce などがあります。

デスクトップ環境の GUI プログラムで使用される locale は、“control center”、“language settings”、“country settings” などと呼ばれる設定画面で指定できます。

デスクトップ環境に属さない個別の GUI プログラムは、設定パネルや環境変数を通じて、自身の locale を持つことができます。

環境変数を通じて locale を指定できるプログラムには、デスクトップの locale とは異なる locale を指定できるものもあります。これはプログラムをメニューやファイルシステムから起動するかわりに、コマンドラインから環境変数を指定した後にプログラムを起動するということです。環境変数の設定については、次のセクション (Section 2.3 [Setting the POSIX Locale], page 10) で説明します。ただし KDE のあるバージョンでは、locale を `LANG`や `LC_ALL`ではなく `KDE_LANG`で設定します。

2.3 環境変数を通じた locale のセッティング

もっとも単純なケースは、あなたの言語がこのパッケージにしたがってインストールされている場合で、環境変数 LANGに適切な '*ll_CC*'の組み合わせを指定するだけです。たとえばあなたが Germany に住んでいて German を話すとしましょう。この場合はシェルプロンプトで単に、'setenv LANG de_DE'(cshの場合)、'export LANG=de_DE'(shの場合)、'export LANG=de_DE'(bashの場合) と実行します。1 度これを.loginや.profileに記述しておけば、毎回適用することができます。

2.3.1 locale 名

locale の名前は通常、'*ll_CC*'という形式で表されます。ここで'*ll*'はISO 639 による 2 文字の言語コード、'*CC*'はISO 3166 による国コードを記述します。たとえば国が Germany、言語が German の場合、*ll*はde、*CC*はDEとなります。言語コードと国コードについては、付表 Appendix A [Language Codes], page 198 と付表 Appendix B [Country Codes], page 206 を参照してください。

国コードも指定するのは冗長だと思うかもしれません。しかし、実際にいくつかの言語は国ごとに方言をもつものがあります。たとえば、'de_AT'は Austria、'pt_BR'は Brazil で使用されます。国コードはこの様な方言を区別するのに役立つのです。

多くの locale 名は'*ll_CC*.*encoding*'という拡張形式で文字のエンコーディングを指定できます。これは多くのユーザーが 2000 年から 2005 年にかけて UTF-8 に移行したためです。たとえば現在の glibc システムの German locale は'de_DE.UTF-8'です。'de_DE'という古い locale 名は、2000 年時点で使用されていた ISO-8859-1(ユーロ通貨記号を持たない) が格納された文字列を参照するのに現在も使用されます。

'*ll_CC*'のかわりに、'*ll_CC*.@*variant*'を使う locale 名もあります。'@*variant*'により、言語 (*ll*) と国 (*CC*) では提供できないような特性を示すことができます。これにより特定の通貨単位を示すことができます。たとえば glibc システムでは'de_DE@euro'は、2002 年以前の通貨記号で使用されていた'de_DE'ではなく、ユーロ通貨を使用する locale を示します。また、言語の方言や筆記に使用される方法 (たとえば'sr_RS@latin'は、'sr_RS'により Serbian を Cyrillic で筆記するのに、Latin 筆記を使用することを示す)、正書法 (orthography rule) を使用するか、などを示すことができます。

その他のシステムでは、単に'*ll*'と指定したりする等、このスキームの様々なバリエーションが使用されています。あなたの言語でサポートされている locale の一覧は、'locale -a | grep ''^*ll*''を実行して取得することができます。

'C'と呼ばれる特別な locale もあります。これはすべての locale を無効にするときに使用します。この locale では、すべてのプログラムが POSIX 標準で指定された英語のメッセージと、指定されていない不特定の文字 (たいていは US-ASCII ですが、オペレーティングシステムによっては ISO-8859-1 や UTF-8 のときもあります) を使用します。

2.3.2 locale の環境変数

locale は複数の *locale* カテゴリー *(locale categories)* から構成されています (Section 1.3 [Aspects], page 3 を参照してください)。プログラムが locale に依存する値を参照する場合は、以下の環境変数を優先度順に参照します:

1. LANGUAGE
2. LC_ALL
3. LC_xxxは、xxx に対応する locale カテゴリーです: LC_CTYPE, LC_NUMERIC, LC_TIME, LC_COLLATE, LC_MONETARY, LC_MESSAGES, ...
4. LANG

変数に空の値がセットされている場合は、無視されます。

LANGはlocaleを指定するときに通常使われる環境変数で、通常はユーザーもこの変数にlocaleを設定します (他の変数がシステムにより設定されていなければ、/etc/profileまたはそれに類する初期化ファイルで設定します)。

LC_CTYPE、LC_NUMERIC、LC_TIME、LC_COLLATE、LC_MONETARY、LC_MESSAGES等は、対応するlocaleのカテゴリーでLANGの設定をオーバーライドするときに使用されます。たとえば、あなたがSpainに住むSwedishのユーザーで、プログラムに数値や日付についてはSpanishの規則で表示し、メッセージだけをSwedishで表示させたいと仮定します。その場合にはlocaledefプログラムで、'sv_ES'または'sv_ES.UTF-8'という名前のlocaleを作成する必要があります。しかし、単にLANG変数にes_ES.UTF-8を設定し、LC_MESSAGES変数にsv_SE.UTF-8という、オペレーティングシステムに事前にインストールされている2つのlocaleを設定することで、同じ効果を得ることができます。

LC_ALLは、これらのすべてをオーバーライドするための変数です。これは通常、特定のプログラムを実行するスクリプトで使用されます。たとえばGNU autoconfにより生成されたLC_ALLスクリプトは、configurationのテストがlocaleに依存した方法で行われないように、LC_ALLを使用します。

残念ながらいくつかのシステムでは、/etc/profile等の初期化ファイルでLC_ALLが設定されています。したがってLANGを設定する場合、ユーザーはまずこの設定を解除し、必要なら他のLC_xxxも解除しなければなりません。

LANGUAGE変数については、つぎのセクションで説明します。

2.3.3 言語の優先リストを指定する

すべてのプログラムが、すべての言語に翻訳されている訳ではありません。翻訳されたメッセージが存在しない場合、デフォルトでは英語のメッセージが表示されます。あなたが他の言語を理解できる場合は、言語の優先順位のリストを設定することができます。これはLANGUAGEと呼ばれる環境変数で行います。GNU gettextはメッセージを処理するために、LC_ALLやLANGに加え、LANGUAGEによる設定を提供します。しかし他のシステムライブラリーで必要となるため、LANG(またはLC_ALL)によるプライマリー言語の設定は、依然として必要です。たとえば、あるSwedishのユーザーは、Swedishの翻訳が存在しないとき、EnglishよりもGermanに翻訳されたものが読みたいとします。そのような場合は、LANGの値は'sv_SE'のまま、LANGUAGEの値を'sv:de'に設定します。

Norwegianのユーザーのためのアドバイス: Norwegian言語 (bokmål) は最近 (2003年)、'no'から'nb'に変更されました。移行期間中、いくつかのこの言語のメッセージカタログは、'nb'と、古い'no'にインストールされるので、Norwegianユーザーは新旧どちらの翻訳も使用できるように、LANGUAGEを'nb:no'に設定することをお勧めします。

他の環境変数とは異なり、LANGUAGE環境変数では、'*ll_CC*'を'*ll*'と省略することにより、その言語で主に使用される方言であることを示します。この事情により、たとえば'de'は'de_DE'(Germanyで話されるGermany)、'pt'は'pt_PT'(Portugalで話されるPortuguese)と同義です。

注意: LANGUAGE変数は、localeが'C'に設定されている場合は無視されます。つまり、最初にローカリゼーションを利用可能にする時に、LANGUAGE変数で言語の優先順位リストを使用する前に、まずLANG(またはLC_ALL)を'C'以外に設定する必要があります。

2.4 特定のプログラムにたいして翻訳をインストールする

GNU gettextを使用するすべてのパッケージが、各言語にたいして同様のサポートを提供している訳ではなく、翻訳は時間をかけて個々に追加されます。パッケージをインストールした後は通常、

オペレーティングシステムか個々のパッケージに同梱された翻訳を使用することになります。しかし新しいローカリゼーションを直接インストールすることもできます。これを行うには、ローカリゼーションの各ファイルがファイルシステム上でどのように保存されているかを理解する必要があります。

翻訳プロジェクトに参加しているプログラムは、`http://translationproject.org/team/index.html`で見つけることができます。この情報のスナップショットは、GNU gettext に同梱されている `ABOUT-NLS`で確認することもできます。

KDE プロジェクトのプログラムを探す場合の出発点: `http://i18n.kde.org/`

GNOME プロジェクトのプログラムを探す場合の出発点: `http://www.gnome.org/i18n/`

他のプログラムに関しては、プログラムのソースコードパッケージに *ll*`.po`のようなファイルが含まれているかどうかでチェックすることができます (`po/`というディレクトリーに保存されているときもあります)。各 *ll*`.po`には、*ll*で略記された言語の翻訳されたメッセージが含まれています。

3 PO ファイルのフォーマット

GNU gettextツールセットは、プログラマーや翻訳者が翻訳のためのファイルを生成、更新、使用する手助けをし、それらの PO ファイルは主としてテキスト形式で編集可能なファイルです。このチャプターでは、PO ファイルのフォーマットについて説明します。

PO ファイルは多くのエントリーから成り立っており、それぞれのエントリーには翻訳される前の原文の文字列と、それに対応する翻訳された文字列との関連が保持されています。ある PO ファイルに含まれるすべてのエントリーは通常、ひとつのプロジェクトに関連し、翻訳されたすべての文字列もひとつの言語を対象に翻訳されたものです。一般的な PO ファイルのエントリーは、以下のような構造を持ちます:

```
white-space
#  translator-comments
#. extracted-comments
#: reference...
#, flag...
#| msgid previous-untranslated-string
msgid untranslated-string
msgstr translated-string
```

翻訳者は、PO ファイルの一般的な構造を十分に理解する必要があります。emacs の PO モードを使用すれば、フォーマットの詳細に関する最小限の知識を理解するだけで、あとは PO モードが彼女にかわって面倒を見てくれます。

以下は簡単なエントリの例です：

```
#: lib/error.c:116
msgid "Unknown system error"
msgstr "Error desconegut del sistema"
```

エントリーは任意の個数の空白文字から開始することができます。GNU gettextツールで生成された場合、エントリーとエントリーの間には通常、1 つの空行があります。#の文字で始まる行はすべてコメント行として扱われます。コメントには 2 種類のコメントがあります。1 つ目は *translator comments(*翻訳者コメント*)*で、#の直後にいくつかの空白文字があり、これらのコメントは翻訳者により保守、保守されます。2 つ目のコメントは *automatic comments(*自動コメント*)*で、これらのコメントは GNU gettextツールにより自動的に挿入、保守されるもので、#の直後に空白文字以外の文字があります。#.で始まるコメントはプログラマーによる翻訳者に向けたコメントを含んでいます。これらのコメントは xgettextプログラムによりプログラムのソースから抽出 (extract) されるため、*extracted comments(*抽出コメント*)*と呼ばれます。#:で始まるコメントには、プログラムのソースコードへの参照 (references) が含まれます。#,で始まるコメントにはフラグ (flags) が含まれています。これらのフラグについては以下で説明します。#|で始まるコメントには、以前のバージョンの翻訳済みメッセージに対応する翻訳前の文字列 (previous untranslated string) が含まれています。

すべてのコメントは、オプションです。

空白文字とコメントの後には、2 つの文字列を表すための文字列があります。最初の文字列は翻訳前の文字列で、これらの文字列のオリジナルはプログラムのソース中に出現する文字列です。その次の文字列は、この翻訳前の文字列に対応する翻訳後の文字列です。オリジナルの文字列は msgidというキーワードで識別され、翻訳は msgstrというキーワードで識別されます。これらの翻訳前と翻訳後の 2 つの文字列は、PO ファイル中で"区切りや\エスケープにより、様々な方法で引用されていますが、文字列の引用などについては PO モードが彼女にかわり面倒をすべて見てくれるので、翻訳者はそれらの正確な引用形式に注意を払う必要がなくなります。

msgid の文字列も自動生成されたコメントと同様、GNU gettextの他のツールにより生成、管理されるので、PO モードは翻訳者がそれらを変更するような操作を提供しません。それらの文字列にたいして彼女にできることは、単にそれを削除することだけで、しかもエントリー全体を削除できるだけです。一方、msgstrの文字列については、実際に翻訳者が編集するための翻訳者コメントなので、PO モードは彼女の必要に応じて、完全な制御を提供します。

#,で始まるコメントは、一般的なコメントとは違いプログラムにより完全に無視されるものではないという点で、特別なコメントです。カンマで区切られた *flag*のリストは、ユーザーのためにより良い診断メッセージを提供するために、msgfmtプログラムにより使用されます。現時点では 2 つの形式の flag が定義されています:

`fuzzy` このフラグは msgmergeプログラムにより生成されるか、翻訳者自身により挿入され、msgstrの文字列が、(もはや) 正しい翻訳ではないことを示します。翻訳をさらに修正する必要があるか、そのまま適用できるかは、翻訳者だけが判断できます。翻訳を完成したら、彼女は fuzzy属性を取り除きます。msgmergeは、あいまい検索 (fuzzy search) により msgidと msgstrエントリーを結びつけた場合のみ、このフラグを挿入します。Section 8.3.6 [Fuzzy Entries], page 56 を参照してください。

`c-format`
`no-c-format`
: これらは人間によって追加されるフラグではなく、xgettextプログラムだけが挿入するフラグです。ここで提案しているような PO ファイルを自動生成するシステムでは、ユーザーが変更を行っても、xgettextプログラムが新しいテンプレートファイルを生成するたびに、変更は上書きされてしまいます。

  c-formatフラグは、翻訳前の文字列と翻訳された文字列が、C 形式の文字列であることを示します。no-c-formatフラグは逆に、文字列が一見して ('%'ディレクティブによる)C 形式の文字列に見えても、C 形式ではないことを示します。

  文字列に c-formatフラグが設定されていると、msgfmtプログラムは、翻訳にたいして妥当性チェックのテストを追加で行います。Section 10.1 [msgfmt Invocation], page 100、および Section 4.6 [c-format Flag], page 26 と Section 15.3.1 [c-format], page 155 を参照してください。

`objc-format`
`no-objc-format`
: Objective C の場合も同様です。Section 15.3.2 [objc-format], page 156 を参照してください。

`sh-format`
`no-sh-format`
: shell の場合も同様です。Section 15.3.3 [sh-format], page 156 を参照してください。

`python-format`
`no-python-format`
: Python の場合も同様です。Section 15.3.4 [python-format], page 156 を参照してください。

`python-brace-format`
`no-python-brace-format`
: Python brace の場合も同様です。Section 15.3.4 [python-format], page 156 を参照してください。

`lisp-format`
`no-lisp-format`
: Lisp の場合も同様です。Section 15.3.5 [lisp-format], page 156 を参照してください。

`elisp-format`
`no-elisp-format`
: Emacs Lisp の場合も同様です。Section 15.3.6 [elisp-format], page 156 を参照してください。

`librep-format`
`no-librep-format`
: librep の場合も同様です。Section 15.3.7 [librep-format], page 157 を参照してください。

`scheme-format`
`no-scheme-format`
: Scheme の場合も同様です。Section 15.3.8 [scheme-format], page 157 を参照してください。

`smalltalk-format`
`no-smalltalk-format`
: Smalltalk の場合も同様です。Section 15.3.9 [smalltalk-format], page 157 を参照してください。

`java-format`
`no-java-format`
: Java の場合も同様です。Section 15.3.10 [java-format], page 157 を参照してください。

`csharp-format`
`no-csharp-format`
: C#の場合も同様です。Section 15.3.11 [csharp-format], page 157 を参照してください。

`awk-format`
`no-awk-format`
: awk の場合も同様です。Section 15.3.12 [awk-format], page 157 を参照してください。

`object-pascal-format`
`no-object-pascal-format`
: Object Pascal の場合も同様です。Section 15.3.13 [object-pascal-format], page 157 を参照してください。

`ycp-format`
`no-ycp-format`
: YCP の場合も同様です。Section 15.3.14 [ycp-format], page 157 を参照してください。

`tcl-format`
`no-tcl-format`
: Tcl の場合も同様です。Section 15.3.15 [tcl-format], page 157 を参照してください。

`perl-format`
`no-perl-format`
: Perl の場合も同様です。Section 15.3.16 [perl-format], page 158 を参照してください。

`perl-brace-format`
`no-perl-brace-format`
: Perl brace の場合も同様です。Section 15.3.16 [perl-format], page 158 を参照してください。

`php-format`
`no-php-format`
: PHP の場合も同様です。Section 15.3.17 [php-format], page 158 を参照してください。

`gcc-internal-format`
`no-gcc-internal-format`
: GCC ソースの場合も同様です。Section 15.3.18 [gcc-internal-format], page 158 を参照してください。

`gfc-internal-format`
`no-gfc-internal-format`
: GNU Fortran コンパイラーのソースの場合も同様です。Section 15.3.19 [gfc-internal-format], page 158 を参照してください。

`qt-format`
`no-qt-format`
: Qt の場合も同様です。Section 15.3.20 [qt-format], page 158 を参照してください。

`qt-plural-format`
`no-qt-plural-format`
: Qt plural 形式の場合も同様です。Section 15.3.21 [qt-plural-format], page 158 を参照してください。

`kde-format`
`no-kde-format`
: KDE の場合も同様です。Section 15.3.22 [kde-format], page 158 を参照してください。

`boost-format`
`no-boost-format`
: Boost の場合も同様です。Section 15.3.23 [boost-format], page 158 を参照してください。

`lua-format`
`no-lua-format`
: Lua の場合も同様です。Section 15.3.24 [lua-format], page 159 を参照してください。

`javascript-format`
`no-javascript-format`
: JavaScript の場合も同様です。Section 15.3.25 [javascript-format], page 159 を参照してください。

以下のように、context specifier(コンテキスト指定子) をともなうエントリーも使用することができます:

```
white-space
#  translator-comments
#. extracted-comments
#: reference...
#, flag...
#| msgctxt previous-context
#| msgid previous-untranslated-string
msgctxt context
msgid untranslated-string
msgstr translated-string
```

コンテキスト (context) は、同じ *untranslated-string*のあいまいさをなくすために提供されます。これにより PO ファイルの中で、異なる *context*で、同じ *untranslated-string*を複数もつことが可能になります。*context*に空の文字列を指定するのと、`msgctxt`の行を指定しないのは、同じではないことに注意してください。

他にも、複数形式 (plural form) を含む翻訳のために使用されるエントリーがあります。

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

以下はエントリーの例です:

```
#: src/msgcmp.c:338 src/po-lex.c:699
#, c-format
msgid "found %d fatal error"
msgid_plural "found %d fatal errors"
msgstr[0] "s'ha trobat %d error fatal"
msgstr[1] "s'han trobat %d errors fatals"
```

`msgid`の前に、前述した`msgctxt`コンテキストを指定することもできます。

ここで追加のフラグを使用できます:

`range:` このフラグは正の数値範囲をともない、`range:` *minimum-value*`..`*maximum-value* という書式で使用します。この範囲には、メッセージが受けとることができる数値を指定します。たとえばある言語では、事前に値が 0 から 10 だとわかっていれば、よりよい翻訳を生成できます。

*previous-untranslated-string*は、`msgmerge`がメッセージを fuzzy としてマークするとき同時にオプションとして挿入されます。これは開発者が、*untranslated-string*にたいしてどのような変更を行ったかを、翻訳者が知る助けになります。

これは、PO ファイルの最後のエントリーに続けて、何らかの行 (通常は空白文字やコメント) があるときに発生します。これらの行は、どのエントリーの一部でもなく、PO ファイルがツールにより処理されるときに捨てられるか、PO ファイルエディターの動作を妨げるときもあります。

このチャプターの残りの部分は、PO ファイルの正確な書式にたいしてよいアイデアを持つ人は興味があるかもしれませんが、PO ファイルエディターを使用する場合はスキップして構いません。しかし、PO ファイルを手で変更したい場合は、注意して読む必要があります。

空の *untranslated-string*は、メタ情報が含まれたヘッダーエントリー (Section 6.2 [Header Entry], page 43 を参照してください) のために予約されています。このヘッダーエントリーはファイルの最初のエントリーにすべきです。空の *untranslated-string*は、この目的のために予約されているので、他の場所で使用することはできません。

*untranslated-string*と *translated-string*は C の文法に従い、それには文字列の括り方やバックスラッシュによるエスケープシーケンスも含まれます。複数行のメッセージを記述するときは、エスケープされた改行文字を使用せずに、改行する行末の最後の文字で引用符を閉じて、PO ファイルの次の行で再び引用を開始します:

```
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"
```

この例の最初の行には、‘for’の fという文字の上に ‘Here’の Hを揃えるために、空の文字列が使用されています。また、キーワード msgidの後ろには 3 つの文字列があり、それらの文字列は連結して使用されます。空の文字列と連結することにより文字列全体は変更されませんが、msgidの行に連結される文字列を、複数行の表示を維持しつつ左揃え表示して、配置を明確にさせるための方法です。空の文字列は省略できますがその場合、msgidの後ろに記述する最初の行は ‘Here’で開始しなければなりません [1]。それぞれの文字列の括りの終端を改行 (‘\n’) の直後にしている理由は、そうしても支障がないからというだけで、任意の文字の後で括りを終端して構いません。

文字列の括りの内側にある、行末を示す ‘\n’は文字列の一部で、文字列の括りの外側の改行は PO ファイル自身の行末を示し、文字列に影響を与えない点に注意してください。

文字列の外側では、空白文字とコメントを自由に使うことができます。行頭の ‘#’から、その行の行末までがコメントとなります。翻訳者が記入するコメントは ‘#’の後ろに空白文字をいくつか記述する必要があります。‘#’の後ろに空白文字がない場合、それは特定の GNU ツールで生成・管理されるコメントとみなされ、PO ファイルが msgmergeで処理されるとき、予期せず削除される可能性があります。

[1] これは GNU gettextの制限ではなく、Solaris の msgfmtとの互換性による制限です。

4 プログラムソースの準備

プログラマーのために、C のソースコードの変更を 3 つにカテゴリーに分けます。1 番目は、ローカリゼーション関数にメッセージ翻訳を必要とするすべてのモジュールを教えることです。2 番目は、プログラムの初期化 (通常は main 関数の内部) で、GNU `gettext`の操作を的確にトリガーすることです。3 番目に、翻訳が必要なプログラム内のすべての文字列定数を識別・調整・マークする必要があります。

4.1 `gettext`宣言のインポート

GNU `gettext`が必要とするすべてのファイルが利用可能で、`Makefile`ファイルも調整済み (Chapter 13 [Maintainers], page 135 を参照してください) だとすると、翻訳対象の文字列を含む C モジュールには以下の行を含める必要があります:

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

翻訳可能な C の書式指定文字列 (他の C モジュールで文字列が定義されている場合も含まれます) を引数として呼び出される、`printf()`/`fprintf()`/... を含む C モジュールにも以下の行を含める必要があります:

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

4.2 `gettext`処理のトリガー

すべてのプログラムで、以下で示すような locale データの初期化の類が必要となります:

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

*PACKAGE*と *LOCALEDIR*は、`config.h`か Makefile で提供される必要があります。詳細については、`gettext`や GNU `hello`のソースを眺めてみるとよいでしょう。

この場合、`LC_ALL`の使用は適切ではないでしょう。LC_ALL`LC_ALL`にはすべての locale カテゴリー、特に`LC_CTYPE`が含まれます。この後者のカテゴリーは、プログラムのために`ctype.h`で定義されている、`isalnum`関数などで処理する文字列クラスを決定することに責任をもっており、入力される文字列の言語によってはうまく動作しません。たとえばソースコードで ç(c-cedilla 文字) が使用されている場合、このプログラムは France では問題ありませんが U.S. では動作しません。

`LC_ALL`という locale カテゴリーに他の locale が使用された場合、`scanf`関数による数字の解析に問題が生じるシステムもあります。標準では、このような場合は`"C"`という locale として知られる追加の書式が認識されるでしょう。しかし`"C"`という locale の書式では、数値を受け付けないシステムもあるようです。状況によっては数値の表示が`"C"`という locale なのか、それとも local のフォーマットかにより認識できないこともあります。これは千単位の桁区切り文字を使用するときに発生します。いくつかの locale 定義では national convention に従い、桁区切り文字として`'.'`を使用しますが、この文字は`"C"`という locale では小数点として使用されます。

これらの理由により、上記のコードの LC_ALLの行は、setlocaleによる行に分けることが必要な場合もあります。

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

POSIX 互換のすべてのシステムでは、LC_CTYPE、LC_MESSAGES、LC_COLLATE、LC_MONETARY、LC_NUMERICおよびLC_TIMEが利用できます。ISO C 準拠のみのシステムも存在し、それらのシステムにはLC_MESSAGESがありませんが、これらの不足にたいする代替は GNU gettext の<libintl.h>と、GNU gnulib の<locale.h>で定義されています。

LC_CTYPEを変更すると、<ctype.h>という標準ヘッダーファイルで定義されている関数、および<string.h>と<stdlib.h>という標準ヘッダーファイルで定義されているいくつかの関数が影響をうけることに注意してください。これが望ましくない場合 (例えばコンパイラーのパーサー)、GNU gnulib のソースディストリビューションにある‘c-ctype’、‘c-strcase’、‘c-strcasestr’、‘c-strtod’、‘c-strtold’モジュールの、C locale でハードコーディングされている代替の関数を使用することができます。

環境に依存した locale と C locale を切り替えて使用することもできますが、この方法はsetlocaleの呼び出しが高価であること、広大なプログラムのソース中で locale を切り替える場所を決定するのが退屈であること、locale の切り替えがスレッドセーフではないこと等の理由により、通常は行われません。

4.3 翻訳可能な文字列の準備

文字列が翻訳可能とマークされる前に、調整が必要なこともあります。翻訳可能な文字列の準備は、その文字列をマーク (次のセクションで説明) する前に行います。文字列を準備する際に留意すべきは以下の点です。

- English スタイルとして正常であること。
- センテンス全体が含まれていること。
- パラグラフで分割されていること。
- 文字列を連結するのではなく、書式文字列を使用すること。
- 特殊なマークアップや制御文字を使用しないこと。

上記のガイドラインにたいする例を、いくつか見てみましょう。

翻訳可能な文字列は、正しい English スタイルである必要があります。言語特有のスラングや省略語が使用されている場合、翻訳者がメッセージを理解できずに不適切な翻訳を作成してしまうことがあります。

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

このメッセージはほとんど翻訳不可能です。表示されるアイテムは *a* parameter(任意の parameter) なのでしょうか、*the* parameter(特定の parameter) なのでしょうか?

```
"No match"
```

メッセージに含まれるあいまいさにより、メッセージが理解できなくなっています。プログラムはファイルに何かをセットしようとしているのでしょうか? "The given object does not match the template(与えられたオブジェクトがテンプレートにマッチしない)"なのでしょうか、それとも "The

template does not fit for any of the objects(そのテンプレートは任意のオブジェクトに適合しない)"なのでしょうか?

どちらのケースも、メッセージに単語を追加することにより、翻訳者と English を話すユーザーの両方を助けることができます。

翻訳可能な文字列は、センテンス全体を含む必要があります。単独の動詞や形容詞を、すべてのメッセージに代替できるように翻訳するのは不可能な場合があります。

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

ほとんどの翻訳者はソースを見ないので、"File %s is %s protected"という、それだけでは理解できない文字列しか目にしません。これを以下のように変更します。

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

この方法なら翻訳者はメッセージを理解するだけでなく、適切な文法の組み立て方を見つけることが出きます。たとえば French の翻訳者なら"write protected"を"protected against writing"のように翻訳するでしょう。

多くの言語では、センテンスの他の場所にある性別や数 (単数形/複数形) によって、あるセンテンスの単語が変わることがあるという理由からも、センテンス全体を含めることが重要になります。English より強い単語間の相互関係を持つ言語もあります。たとえ English ではうまく動作しても、多くの言語では半分に分割した 2 つのセンテンスを翻訳者に翻訳してもらってから、2 つの翻訳を機械的に結合しても、満足な翻訳とはなりません。これが翻訳者がセンテンス全体を処理する必要がある理由です。

センテンスが 1 つの行に対応しない場合もあります。これは以下のように、printfステートメントを使って 2 つの出力により、1 つのセンテンスを出力しているような場合です。

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

翻訳者は 2 つのセンテンスを翻訳する必要があるでしょうが、POT ファイル内には 2 つのセンテンスが分割された 1 つのセンテンスだと、彼女に教える情報はありません。2 つの printfステートメントを 1 つにする必要があります。そうすれば翻訳者がセンテンスを一括して処理できるので、翻訳をどの位置で改行すべきか決められるようなります。

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

では以下のような隣接した 2 つのセンテンスの場合はどうなるでしょうか:

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

上記の 2 つのセンテンスは、1 つにまとめる必要があるでしょうか? このような場合、2 つのセンテンスが互いに関連していて、一緒にしたほうが翻訳者が理解・翻訳しやすくなるようなら、マージすることをお勧めします。一方、2 つのメッセージのうち 1 つが定型的なもので、他の場所でも使用されるようなメッセージの場合は、マージしないほうが翻訳者にとって有益です (同じメッセージが複数の箇所に出現する場合、xgettext がそれらをまとめるので、翻訳者は 1 度だけそのメッセージを翻訳すればよくなります)。

翻訳可能な文字列は、単一のパラグラフ (段落) に制限すべきです。1 つのメッセージの長さは、10 行以内にしましょう。その理由は、翻訳可能な文字列が変更されたとき、翻訳者は翻訳済み文字列全体を更新しなければならないからです。たとえ 1 つの単語を変更しただけでも、(現在の翻訳ツールでは) 翻訳者にはそれがわからないので、彼女はメッセージ全体を校正しなければならなくなってしまいます。

多くの GNU プログラムは、‘--help’オプションにより複数画面にまたがる出力を生成します。そのようなメッセージを、1 つが 5 行から 10 行のメッセージに分割するのは、翻訳者にたいする礼儀です。ドキュメント化するオプションを、入力オプションと出力オプション、情報を出力するオプションのようにグループ分けしてもよいでしょう。グループ分けすることにより、オプションを探すすべてのユーザーを助けることができます。

ハードコーディングされた文字列の結合により、English 文字列を生成することがあります:

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

翻訳者にセンテンス全体を表示するためという理由だけではなく、object1と object2の順番が入れ替わるような言語もあるので、これは以下のような書式文字列を使用するように変更する必要があります:

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

似たようなケースとして、コンパイル時の文字列結合があります。ISO C 99 のインクルードファイルである<inttypes.h>には、printfで整数型 ‘int64_t’を出力するための PRId64マクロが含まれています。このマクロは通常、プラットフォームに応じて"d"、"ld"、"lld" のような文字列定数に展開されます。以下のようなコードがあるとします。

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

gettextツールとライブラリーには、これら<inttypes.h>のマクロにたいする特別なサポートがあるので、上記のような場合は単に以下のように書くことができます。

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

この場合、PO ファイルには"The amount is %0<PRId64>\n"という文字列が含まれます。翻訳者は同様に"%0<PRId64>"と翻訳すれば、実行時に gettext関数が、"d"、"ld"、"lld" などから適切な文字列定数に変換します。

これは事前に定義された<inttypes.h>マクロにたいしてのみ機能します。あなたが ‘MYPRId64’のような似たようなマクロを定義した場合、xgettextはそれを知ることができないので、コードを以下のように変更してください:

```
char buf1[100];
sprintf (buf1, "%0" MYPRId64, number);
printf (gettext ("The amount is %s\n"), buf1);
```

これでプラットフォームに依存するコードと、インターナショナリゼーションのコードは、別のステートメントに分けられました。バッファーの長さは 100 バイト以内でよいことに注意してください。なぜなら利用可能なすべてのハードウェアーの整数型は 128 ビットに制限されており、128 ビット整数を出力するには、10 進、8 進、16 進に関わらず最大で 54 バイトあればよいからです。

これは他のプログラム言語には適用できます。Java と C#では文字列結合は、それらのコンパイラーのビルトイン操作なのでとても頻繁に使用されます。以下のような C や Java のコードがあるとします

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

これを以下のような書式師弟文字列を含むステートメントに変更します:

```
System.out.println(
    MessageFormat.format("Replace {0} with {1}?",
```

```
                                    new Object[] { object1, object2 }));
```

C#の場合は以下のように変更します

```
Console.WriteLine("Replace "+object1+" with "+object2+"?");
```

これを以下のような書式師弟文字列を含むステートメントに変更します:

```
Console.WriteLine(
    String.Format("Replace {0} with {1}?", object1, object2));
```

通常使用しないようなマークアップや制御文字は、翻訳可能な文字列の中に含めるべきではありません。翻訳者はマークアップや制御文字がもつ特別な意味は理解しません。

たとえば‘|’の右側と左側とで何らかの GUI 要素を分ける規則があるような場合、翻訳者は特別なコメントなしではその規則を理解することはできません。このような場合は、翻訳者が右側と左側の文字列を個別に翻訳できるようにするのがよいでしょう。

他の例としては、‘`argp`’で制御文字‘`\v`’(vertical tab) を使用する場合の規則です。これは 1 つの文字列を 2 つのセクションに分ける場合に使用されます。このような文字列をそのまま翻訳可能文字列とするには問題があります。翻訳者によっては、これを単純に改行や空行に置き換えてしまうかもしれません。PO ファイルエディターの中には、制御文字の vertical tab を入力するのが困難なものもあります。上記の理由により、あなたは翻訳文字列の対応する位置に、‘`\v`’文字があることを期待できません。この問題にたいする解決策は、繰り返しになりますが、翻訳者が個別に文字列を翻訳できるようにしておいて、実行時に 2 つの翻訳された文字列を、規則が要求する‘`\v`’で結合することです。

HTML マークアップは十分に一般的なマークアップなので、翻訳可能文字列を使用しても大丈夫でしょう。しかし GNU gettext ツールは、翻訳された文字列が well-formed な HTML であるかは検証しないことに留意してください。

4.4 ソース内でマークはどのように見えるか

C ソース中で翻訳される文字列は、すべてマークする必要があります。マーキングは翻訳可能な文字列を、関数やプリプロセッサーのマクロに、単独の引数として引き渡す方法で行われます。翻訳のために利用可できる関数またはマクロは少なく、マーキングのキーワードとしてそれらの名前が使用されます。マーキングは翻訳される文字列自体に何かを行うよりは、文字列にアタッチすることによりマーキングを行なう方法が、より多く使用されます。明らかな例としては、フォーマット文字列によりエラーメッセージを生成する場合です。フォーマット文字列は翻訳する必要があり、フォーマット文字列の‘`%s`’で指定した箇所に挿入される文字列も同様だとすると、たとえば`sprintf`の結果には、‘`error_string_out()`’のようなルーチンからなる、多くの異なるインスタンスが含まれることになり、これらをすべてリストするのは非現実的です。

マーキングには 2 つの目的があります。1 つ目は実行時に翻訳を取得するトリガーとなることです。キーワードは引数となる文字列にたいして、可能なかぎり (そして望む限り)、動的に適切なトランスレーションを返すルーチンへと解決されます。ローカライズ可能な文字列は、変数にあてがわれていたり、関数の引数になっている場合がほとんどです。しかし翻訳可能な文字列が構造体の初期化時に使用される等の例外もあります。Section 4.7 [Special cases], page 27 を参照してください。

2 つ目の目的は、`xgettext`が、一連のプログラムソースをスキャンして PO ファイルのテンプレートを生成するときに、翻訳可能な文字列を適切に抽出する手助けをすることです。

翻訳可能な文字列をマークするための標準的なキーワードは‘`gettext`’で、これは GNU `gettext` パッケージの名前の由来にもなっています。パッケージで少量の‘`gettext`’キーワードやマクロ、関数をそのまま使うのは簡単です。しかし`gettext`インターフェースを多用するパッケージの場合、主要なキーワードには目立つ名前ではなく、より簡潔な名前を使用する方が便利です。キーワードはパッケージ内のすべての文字列の箇所に記述されますが、プログラマーは通常、彼らのプログラムのソー

スがインターナショナライズされるものだといつも強制的に思い出したいとは望みませんし、その必要もありません。また長いキーワードはより多くの文字数を使用するので、ソースの 1 行を 79 から 80 文字にインデントするための労力が余分にかかるという欠点もあります。

多くのパッケージはキーワードとして '_'(単なるアンダーライン) を使用して、'gettext ("Translatable string")'を、'_("Translatable string")'のように記述しています。また GNU 標準のコーディング規約は実際、この特定の用途のためにキーワードと開き括弧の間に、余白としてスペースを要求しています。これにより翻訳可能な文字列のためにかかる文字的なオーバーヘッドは、アンダーラインと 2 つの括弧というたった 3 文字に短縮されます。たとえ GNU gettext がこの方式を内部的に使用していたとしても、これは公式な提案ではありません。正式なキーワードはあくまでも本物の 'gettext'です。しかし 'gettext'のかわりに '_'を使用したい人は、以下のように定義すると簡単になります。

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

単に '#include <libintl.h>'とするのではなく、上記のようにすれば簡単に使用できます。

マーキングのためのキーワードである 'gettext'と '_' は、翻訳可能文字列を単一の引数とします。他の位置に引数をするマーキング用の関数を定義することもできます。関数が呼び出されたときの引数の合計数にもとづいた位置のマーク用引数を作ることもできます (通常は C++の場合)。これは xgettextの '--keyword'により実現されます。xgettextにこのような引数を渡すには gettextize が使用されます、その方法については Section 13.4.3 [po/Makevars], page 140 と Section 13.5.6 [AM_XGETTEXT_OPTION], page 149 で説明します。

長い文字列は複数行に分けられることに注意してください。コンパイル時には ISO C および ISO C++にもとづく文字列の自動連結が行われますが、xgettextもこの構文をサポートしています。

後でメンテナンスするのも簡単になります。もしあなたがプログラマーで、文字列を追加、変更した場合、その文字列が翻訳される必要があると考えた場合は、'_()'でマークすればよいのです。たとえば '"%s"'は、翻訳しない文字列だとします。しかし '"%s: %d"'は翻訳するような場合です (French の場合は通常、English とは異なり、コロンの前にスペースを挿入する翻訳が必要になります)。

4.5 翻訳可能文字列のマーク

PO モードには、翻訳者向けというよりはプログラマー向けの一連の機能があります。それらの機能により彼は、プログラムのソース中の文字列が、翻訳可能かどうか、対話的にマークすることができます。彼が選んだ他のエディターでも、それらの文字列を探してマークするのは、プログラマーにとって簡単な作業かもしれませんが、PO モードはこれらの作業をより快適にしてくれます。また PO モードは、プログラマーの素養を持つ翻訳者、または翻訳者の素養を持つプログラマーにたいして、プログラムのソース中の翻訳可能な文字列をマークするツールを与えてくれると同時に、インターナショナライズされるパッケージにたいする翻訳を生成するツールを与えてくれるのです。

以下で説明する PO モードのコマンドが対象とするプログラムのソースは、PO ファイルのコマンドを使う前に、プロジェクト用の Emacs tags テーブルを生成する必要があります。これは簡単です。任意のシェルウィンドウでプロジェクトのルートディレクトリに移動して、以下のようなコマンドを実行してください:

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

ここでは src/、および lib/ディレクトリーにあるすべての.hと.cファイルを処理したいとします。このコマンドは指定されたすべてのファイルを検索して、プロジェクトのルートディレクトリーに TAGS という、Emacs が解釈できる要約された形式のファイルを作成します。

GNU コーディング規約に従うパッケージには、すべてのディレクトリーとソースコードを含んだすべてのファイルにたいして、tags、または TAGSファイルを作成するという目標があります。

1 度 TAGSを準備すれば、以下のコマンドが彼のソース中の翻訳可能な文字列をマークする手助けをしてくれます。これらのコマンドは PO ファイルのウィンドウから実行される必要がありますが、PO ファイルはまだ作成されていません。しかし新しいウィンドウで空の PO ファイルを新規に作成して、そこからコマンドを実行すれば問題ありません。この空の PO ファイルの内容は、プログラムのソース中の文字列を翻訳可能にマークするにつれて、徐々に増えていきます。

`,` 翻訳候補となりそうな文字列をプログラムのソースから検索します (po-tags-search)。

M-`,` 検索された最後の文字列を '_()'でマークします (po-mark-translatable)。

M-`.` 検索された最後の文字列を、利用可能なキーワードによりマークします。プレフィックスと一緒にこのコマンドを使うことにより、キーワードを管理することができます (po-select-mark-and-mark)。

`,` (po-tags-search) コマンドは、翻訳候補と思われるような次の文字列を検索して、プログラムのソースを Emacs の他のウィンドウで表示します (その文字列がウィンドウの上部にくるように表示されます)。文字列が長くてウィンドウに収まらないような場合は、文字列の最後の部分が表示されます。カーソルは常に PO ファイルのウィンドウにあります。その文字列が他の言語に翻訳されたほうがよいと判断したら、*M-*`,`、または *M-*`.`により文字列をマークします。翻訳する必要がないと判断した場合は、単に`,`コマンドで次の文字列を検索してください。

3 つ以上の文字の並びは、翻訳候補となります。1 行の文字の並びが最大で 2 つでも、文字の数が非文字より多い場合は、翻訳候補と判断します。文字を含まない文字列、または孤立した文字だけの文字列は無視されます。コメント文字列、および PO モードが把握しているキーワード (以下を参照してください) ですでにマークされている文字列も無視されます。

Emacs にたいして TAGSを指定していない場合、最初にこのコマンドを使うときにミニバッファー (minibuffer) に入力を求められます。TAGSファイルは、Emacs の標準コマンドである *M-x visit-tags-table* を入力して、正しい TAGSファイルを入力することにより、後から変更することができます。Section "Tag Tables" in *The Emacs Editor* を参照してください。

`,`コマンドは毎回、前回に検索した箇所から検索を再開し、TAGSに従ってすべてのプログラムソースを処理するまで検索します。コマンド (*C-u* `,`) にプレフィクス引数 (prefix argument) を与えることにより、プログラムのソースの先頭から検索を再開させることができます。この場合、前回マークした翻訳可能な文字列は自動的にスキップされます。

`,`コマンドを使用することにより、Emacs の標準コマンドが使用できなくなることはありません。たとえば、標準の tags-search、および tags-query-replaceコマンドは、`,`のサーチ順序とは独立して、中断されることなく使用できます。しかし、最初の`,`コマンド (またはコマンド引数をともなう`,`コマンド) は、Emacs の標準的な tags 検索を最初の tags にリセットしてしまうよう実装されているので、この再初期化は除きます。

M-`,` (po-mark-translatable) コマンドは、前回検索された文字列を、キーワード '_'でマークします。*M-*`.` (po-select-mark-and-mark) コマンドは、ミニバッファーでキーワードの入力を求めて、文字列をマークするのにそのキーワードを使用します。どちらのコマンドも、マークした文字列に対応する新しい未翻訳のエントリーを PO ファイルに作成して、そのエントリーをカレントのエントリーとします (そのエントリーをすぐに翻訳するのが簡単になります)。*M-*`,`や *M-*`.`によるプログラムソースの変更により、ソース 1 行の文字数が 80 文字を超えてしまうような場合もありますが、これにたいする再インデントなどは別の作業になります。プログラムソースのウィンドウから、Emacs の別のウィンドウに移ったりするために、PO モードから *O*コマンドを使う場合もあるでしょう。`,`コ

マンドに次の文字列を告げるような場合、PO ファイルのウィンドウにカーソルを戻すには、なんらかの Emacs の標準コマンドを使う必要があります。

*M-.*には、キーワードをいちいち全部入力しなくてもよいような、スピードアップのための機能がいくつかあります。1 つ目は、プロンプトで単に RETを押すだけで、好ましいキーワードが表示されるというスピードアップ機能です。2 つ目は、入力したいキーワードにたいして、そのキーワードの先頭部分を一意に特定できる分だけ入力すれば、コマンドが残りの部分を補完してくれるスピードアップ機能です。これは PO モードが利用可能なキーワードを知っていて、ミスタイプによる誤ったキーワードは受け付けられないことを意味します。

キーワードの入力を求められたときに?を入力すると、コマンドは既知のキーワードのリストを表示し、そこから選択して入力することができます。(*C-u M-.*) によりコマンドが引数が指定された場合、単純なキーワード管理以外による、プログラムのソースと PO ファイルのバッファーの更新が禁じられます。この場合、コマンドはキーワードの完全な入力を求め、そのキーワードは以降の *M-.*コマンドで使用されます。さらに、この新しいキーワードは自動的に、以降のコマンド用のお好みのキーワードに追加されます。*C-u M-.* にたいして既知のキーワードを答えた場合、単にお勧めのキーワードが 1 つ変更されるだけで、他には何もしません。

*M-.*により認識されるすべてのキーワードは、,コマンドによる文字列検索時に再編成されます。この時、これらのキーワードですでにマークされている文字列は自動的にスキップされます。同時に複数の PO ファイルを開いている場合、それぞれが個別に既知のキーワードを保有します。現在のところ PO モードにキーワードを削除するための機能はないので、(*q*を使用するなどして) ファイルを一旦閉じてから、再度開く必要があります。Emacs のウィンドウに PO ファイルを新規に開いたときは、‘gettext’と ‘_’だけがキーワードで、*M-.*コマンドのお好みのキーワードは‘gettext’になっています。実際のところ、‘_’はビルトインの *M-,*コマンドに割り当てられているので、お勧めにするには便利ではありません。

4.6 キーワードの前の特別なコメント

C プログラム中の文字列は、しばしば printfファミリーと呼ばれる関数呼び出しで使用されます。これらで使用される書式指定文字列に関して特筆すべきは、%で始まる書式指定子が含まれていることです。以下のようなコードがあるとしましょう。

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

上記の文字列にたいして、以下のような German の翻訳が考えられます:

```
"%d Zeichen lang ist die Zeichenkette ‘%s’"
```

German を話せない C プログラマーでも、まずい点があることに気がつくでしょう。文字列中の書式指定子の順序が変更されていますが、printfの引数の順序は変更されません。一番問題なのは、文字列のアドレスが期待されている箇所に、文字列の長さが渡していることです。

翻訳に起因する実行時のエラーを防ぐために、msgfmtは翻訳前の文字列と、翻訳後の文字列に含まれる引数のタイプと数を、静的にチェックすることができます。このチェックを満足しないような場合、msgfmtに ‘-c’が指定されていると、msgfmtはエラーを発生させて MO ファイルを生成しません。‘msgfmt -c’を使用することにより、エラーを事前に検出して、実行時の問題を防ぐことができます。

German の翻訳で上述の単語順が正しい場合は、以下のように記述する必要があります

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

msgfmtルーチンは、この特別な表記法を認識できます。

プログラムのすべての文字列が書式文字列というわけではないので、.poファイルの中のすべての文字列を msgfmtがテストする必要はありません。また文字列の中に書式指定子と似た文字列が含まれるが、その文字列が printfで使われる文字列ではないような場合は、問題が発生します。

そのため xgettextは、それらの書式文字列と思われるメッセージに特別なタグを付与します。このタグ付けは絶対的なルールではなく、発見的なルールです。.poファイルの中のそれらのエントリーには、#,によるコメント行で、c-formatというフラグによりマークされます (Chapter 3 [PO Files], page 13 を参照してください)。

注意深い読者は、まだ問題があると気づくでしょう。発見されたものが間違っている場合です。これは真実であり、そのために xgettextは、プログラマーが意志決定すべき特別な種類のコメントを認識することができるのです。gettextキーワードと同じ行、またはそれに続く行に xgettext:c-formatという単語を含むコメントを発見すると、xgettextはどのような場合であれ、文字列を c-formatフラグでマークします。この種のコメントは、xgettextが文字列を書式文字列と認識しない場合 (テストしてみて実際に認識されない場合) に使う必要があります。gettextキーワードと同じ行にコメントがある場合、翻訳される前にコメントを挿入しなければならないことに注意してください。

このような状況は頻繁に発生します。printf関数にわたされる文字列に書式指定子が含まれない場合もあります。そのようなケースでは通常、fputsを使用するのでしょうが、それでもこのような状況はあり得ます。このような場合、xgettextはそれを書式文字列として認識しませんが、翻訳に書式指定子として認識されるような文字列が含まれていると何が起こるでしょうか? printf関数はパラメーターにアクセスしようとしますが、翻訳前の文字列には何も引数がわたされないため、パラメーターは存在しません。

もちろん他の原因により、xgettextが間違って書式文字列ではない文字列を、書式文字列と認識することがあります。このような場合、msgfmtは多くの警告を出力し、.poファイルへの変換は失敗します。このように間違って書式文字列と認識されるのを防ぐには、上記と同様に xgettext:no-c-formatという文字列を含むコメントを使用します。

文字列が c-formatと間違ってマークされている場合、ユーザーは何が原因なのか調べることができます。--debugオプション使用して、どのように問題を解決するかについては、Section 5.1 [xgettext Invocation], page 33 を参照してください。

4.7 翻訳可能文字列の特別なケース

注意深い読者なら、常に翻訳可能な文字列を gettext(または同様のもの) でマークすることはできないことに気づくでしょう。たとえば以下のようなケースです:

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

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

文字列"a default message"にたいするマーク付けは問題ありませんが、配列messagesを初期化する文字列はマークできません。どうすればよいのでしょうか? このような場合は 2 つのタスクを達成する必要があります。最初に xgettextプログラムが文字列を見つけ出せるように文字列をマークします (Section 5.1 [xgettext Invocation], page 33 を参照してください)。次に実行時に文字列を出力する前に、文字列を翻訳するのです。

最初のタスクは、no-op という新しいキーワードを作ることにより達成できます。2 番目のタスクは、配列の文字列にたいするすべてのアクセスポイントをマークします。考えられる解決策の 1 つは、以下のようなものです:

```
#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);
  ...
}
```

どのようなケースでも、fputsに記述された文字列は翻訳されると思ってください。どのようにして xgettextが、追加のキーワード gettext_noopを認識するかについては、Section 5.1 [xgettext Invocation], page 33 を参照してください。

もちろん、これが唯一の解決策という訳ではありません。以下の方法のうちのいずれかを使用することもできます:

```
#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));
  ...
}
```

しかしこの方法には欠点があります。プログラマーは文字列"a default message"にも gettext_noopを使うよう留意する必要があります。gettextを使用することにより、予期しない結果となる場合もあります。

利点の 1 つは、どのようなケースでも出力が翻訳されるようにするために、制御フローを分析する必要がないことです。この分析は一般的に難しいものではありませんが、これにあてはまらないような状況では、2 番目の方法を使用することもできます。

4.8 翻訳バグの報告をユーザーに奨励する

コードにはバグが付き物ですが、翻訳も同様です。ユーザーがそれらのバグを報告できるようにする必要があります。メンテナーが同時に翻訳者であるような場合を除き、メンテナーが翻訳を変更することはないため、プログラマーやパッケージのメンテナーに翻訳のバグを報告するのは得策ではありません。したがって翻訳のバグは翻訳者に報告されなければなりません。

ここで紹介する方法で組織化することにより、メンテナーが翻訳のバグ報告をどこかへ転送したり、翻訳者や翻訳チームのアドレスのリストを維持する必要もなくなります。

すべてのプログラムには、バグを報告するためのアドレスを示す場所があります。GNU プログラムの場合、“–help” オプションにより表示される、“usage”(使用方法) とよばれる機能が該当する場所です。この場所に翻訳のバグ報告のためのアドレスを追加するよう、翻訳者に示すのです。たとえば以下のようなコードがあるとします

```
printf (_("Report bugs to <%s>.\n"), PACKAGE_BUGREPORT);
```

以下のように、翻訳者への指示を追加することができます：

```
/* TRANSLATORS: The placeholder indicates the bug-reporting address
   for this package.  Please add _another line_ saying
   "Report translation bugs to <...>\n" with the address for translation
   bugs (typically your translation team's web or email address).  */
printf (_("Report bugs to <%s>.\n"), PACKAGE_BUGREPORT);
```

これらは‘xgettext’により抽出され、以下のようなエントリーを含む.pot ファイルとなります:

```
#. TRANSLATORS: The placeholder indicates the bug-reporting address
#. for this package.  Please add _another line_ saying
#. "Report translation bugs to <...>\n" with the address for translation
#. bugs (typically your translation team's web or email address).
#: src/hello.c:178
#, c-format
msgid "Report bugs to <%s>.\n"
msgstr ""
```

4.9 翻訳にたいして正確な名前をマークする

人や都市、場所の名前などは翻訳用にマークする必要があるでしょうか? Latin 文字で記述する言語 (English、Spanish、French、German 等) しか知らない人は、“no” と言いたいでしょう。なぜなら通常は、それらの言語間で名前は変更されないからです。しかし一般的には、ある文字体系から他の文字体系に変換するときには、音声表記や音訳により名前も変換されます。たとえば Russian や Greek の名前は、English に変換されるときに Latin のアルファベットに変換され、English や French が Japanese に変換されるときは Katakana 文字に変換されます。対象となる言語を話す人たちは、一般的には翻訳前の文字で記述された元の名前を読めないので、これらの変換が必要になります。

それゆえプログラマーとしては、名前を翻訳用にマークするとともに、翻訳者にたいしてそれが元の正確な名前であることと、どのように取り扱うかについての特別なコメントを付与する必要があります。以下は簡単な例です:

```
printf (_("Written by %s.\n"),
        /* TRANSLATORS: This is a proper name.  See the gettext
           manual, section Names.  Note this is actually a non-ASCII
           name: The first name is (with Unicode escapes)
           "Fran\u00e7ois" or (with HTML entities) "Fran&ccedil;ois".
           Pronunciation is like "fraa-swa pee-nar".  */
        _("Francois Pinard"));
```

GNU gnulib は、オリジナル名にカッコ内に翻訳された名前を自動的に追加する 'propername' (http://www.gnu.org/software/gnulib/MODULES.html#module=propername) というモジュールを提供しています。これによりスクリプトを変更しなくてもよいような場合には、翻訳者がASCII で記述できないような名前にたいして、適切な非 ASCII 文字を入力するというタスクから開放されます。この、より快適な形式は以下のようなものです:

```
printf (_("Written by %s and %s.\n"),
        proper_name ("Ulrich Drepper"),
        /* TRANSLATORS: This is a proper name.  See the gettext
           manual, section Names.  Note this is actually a non-ASCII
           name: The first name is (with Unicode escapes)
           "Fran\u00e7ois" or (with HTML entities) "Fran&ccedil;ois".
           Pronunciation is like "fraa-swa pee-nar".  */
        proper_name_utf8 ("Francois Pinard", "Fran\303\247ois Pinard"));
```

元の名前を、(Unicode エスケープや HTML エンティティーとしてではなく) 直接 Unicode で記述して、IPA(International Phonetic Alphabet: 国際音標文字。http://www.wikipedia.org/wiki/International_Phonetic_Alphabet) を参照してください) により発音を示すこともできます。

翻訳者としては、名前を翻訳するときは注意深く行う必要があります。なぜなら名前がバラバラに翻訳されたり、間違って翻訳されることは、人を不快にさせるからです。

あなたの言語が Latin 文字を使用している場合、必要なのはその言語で普段使用している文字で名前を再構築することだけです。これは c-cedilla 文字を含む翻訳を提供するような場合です。あなたの言語が Latin 文字とは異なる文字を使用していて、人がそれを通常 Latin 文字として読まれるようには話していない場合、それは翻訳を意味しています。プログラマーが簡単な方法を使用している場合でも、Latin 文字を読む人のために、括弧付きで元の名前を記述する必要があります。プログラマーが上述の 'propername'モジュールを使用している場合は、元の名前を括弧付きで記述するのはプログラムが行うので、あなたが記述する必要はありません。以下は対象言語が Greek の場合の例です:

```
#. This is a proper name.  See the gettext
#. manual, section Names.  Note this is actually a non-ASCII
#. name: The first name is (with Unicode escapes)
#. "Fran\u00e7ois" or (with HTML entities) "Fran&ccedil;ois".
#. Pronunciation is like "fraa-swa pee-nar".
msgid "Francois Pinard"
msgstr "\phi\rho\alpha\sigma\omicron\alpha \pi\iota\nu\alpha\rho"
       " (Francois Pinard)"
```

このように名前の翻訳は微妙な領域に属する話題なので、翻訳を提出する前にテストすることをお勧めします。

4.10 ライブラリーソースの準備

あなたがプログラムではなくライブラリーを作成する場合、gettextの使用方法にはわずかな違いしかありません。ここでは前提として、ライブラリーが自分自身の翻訳ドメインと POT ファイルを持つとします (ライブラリーがメインプログラムの翻訳ドメインと POT ファイルを使用する場合は、前のセクションを変更なしに適用できます)。

1. ライブラリーのコードでは、`setlocale (LC_ALL, "")`を呼び出しません。locale のセットはメインプログラムの責任です。ライブラリーのドキュメントにはこの事実を明記して、ライブラリーを使用するプログラム開発者が認識できるようにする必要があります。
2. ライブラリーのコードでは、`textdomain (PACKAGE)`を呼び出しません。text domain のセットはメインプログラムの責任です。
3. プログラムのための初期化は以下のようなコードでした

   ```
   setlocale (LC_ALL, "");
   bindtextdomain (PACKAGE, LOCALEDIR);
   textdomain (PACKAGE);
   ```

   ライブラリーの場合は以下のコードだけになります

   ```
   bindtextdomain (PACKAGE, LOCALEDIR);
   ```

   ライブラリーの API にまだ初期化の関数が無いなら、bindtextdomain呼び出しを含む初期化関数を作成する必要があります。しかし通常、この初期化関数をエクスポートしたりドキュメント化する必要はありません。初期化関数がまだ呼び出されていない場合は、ライブラリーのすべてのエントリーポイントとなる関数から初期化関数を呼び出すだけで十分です。これを満足するような典型的な例は、初期化関数が呼び出し済みかどうかを保持するブール値の静的な変数を使用する方法です:

```
static bool libfoo_initialized;

static void
libfoo_initialize (void)
{
  bindtextdomain (PACKAGE, LOCALEDIR);
  libfoo_initialized = true;
}

/* This function is part of the exported API.  */
struct foo *
create_foo (...)
{
  /* Must ensure the initialization is performed.  */
  if (!libfoo_initialized)
    libfoo_initialize ();
  ...
}

/* This function is part of the exported API.  The argument must be
   non-NULL and have been created through create_foo().  */
int
foo_refcount (struct foo *argument)
{
  /* No need to invoke the initialization function here, because
     create_foo() must already have been called before.  */
  ...
}
```

4. プログラムでは通常、各ソースファイル中で、以下のように‘_’マクロを定義しました

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

自身の翻訳ドメインを持つライブラリーの場合は、以下のようになります:

```
#include <libintl.h>
#define _(String) dgettext (PACKAGE, String)
```

別の言い方をすると、gettextのかわりに dgettextを使用するということです。同様に、ngettextが使用される箇所には、dngettextを使用する必要があります。

5 PO テンプレートファイルのマーク

ソースの準備ができたら、プログラマーはPO テンプレートファイルを作成します。このセクションでは、その目的のために xgettextをどのように使用するかについて説明します。

xgettextは、*domainname*.poという名前のファイルを作成します。あなたはそれを*domainname*.potという名前にリネームする必要があります (xgettextは、どうして直接*domainname*.potを作成しないのでしょうか? これは歴史的な理由からです。xgettextが設計されたときはPO ファイルと PO テンプレートファイルの区別があいまいで、拡張子の '.pot'も使用されていなかったからです)。

5.1 xgettextプログラムの呼び出し

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

xgettextプログラムは、与えられた入力ファイルから、翻訳可能な文字列を抽出します。

5.1.1 入力ファイルの位置

'*inputfile* ...'
: 入力ファイルを指定します。

'-f *file*'
'--files-from=*file*'
: 入力ファイルの名前を、コマンドラインからではなく、*file*から読み込みます。

'-D *directory*'
'--directory=*directory*'
: ディレクトリーのリストに *directory*を追加します。このディレクトリーのリストからソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

*inputfile*に '-'が指定された場合は、標準入力から読み込みます。

5.1.2 出力ファイルの位置

'-d *name*'
'--default-domain=*name*'
: 出力ファイルとして、(messages.poのかわりに)*name*.poを使用します。

'-o *file*'
'--output=*file*'
: (*name*.poや messages.poではなく) 指定されたファイルに出力を書き込みます。

'-p *dir*'
'--output-dir=*dir*'
: ファイルは *dir*に出力されます。

出力の *file*に '-'または '/dev/stdout'が指定された場合、出力は標準出力に書き込まれます。

5.1.3 入力ファイルの言語の選択

‘-L *name*’
‘--language=*name*’
: 入力ファイルの言語を指定します。サポートされている言語は、C、C++、ObjectiveC、PO、Shell、Python、Lisp、EmacsLisp、librep、Scheme、Smalltalk、Java、JavaProperties、C#、awk、YCP、Tcl、Perl、PHP、GCC-source、NXStringTable、RST、Glade、Lua、JavaScript、Valaです。

‘-C’
‘--c++’ --language=C++の省略指定です。

デフォルトでは、入力ファイルの言語は拡張子により推測されます。

5.1.4 入力ファイルの解釈

‘--from-code=*name*’
: 入力ファイルのエンコーディングを指定します。このオプションはメッセージ文字列や、それらのコメントに非 ASCII 文字が含まれている場合のみ必要です。Tcl と Glade の入力ファイルは、このオプションの指定に関わらず、UTF-8 が想定されることに注意してください。

デフォルトでは、入力ファイルのエンコーディングは ASCII であると仮定されます。

5.1.5 オペレーションモード

‘-j’
‘--join-existing’
: 既存のファイルのメッセージを結合します。

‘-x *file*’
‘--exclude-file=*file*’
: *file*のエントリーは抽出されません。*file*には、PO ファイルか POT ファイルを指定します。

‘-c[*tag*]’
‘--add-comments[=*tag*]’
: *tag*で始まるコメントブロックを、出力ファイル中のキーワード行の前に配置します。このオプションで *tag*を指定しない場合には、出力ファイル中のすべてのキーワード行の前にコメントブロックが配置されます。

5.1.6 言語特有のオプション

‘-a’
‘--extract-all’
: すべての文字列を抽出します。

  このオプションはほとんどの言語、すなわち、C、C++、Objective-C、Shell、Python、Lisp、EmacsLisp、librep、Java、C#、awk、Tcl、Perl、PHP、GCC-source、Glade、Lua、JavaScript、Vala に影響を与えます。

‘-k[*keywordspec*]’
‘--keyword[=*keywordspec*]’
: 検索する追加のキーワードを *keywordspec*に指定します。*keywordspec*を指定しない場合には、デフォルトのキーワードを使用しないことを意味します。

*keywordspec*として指定された *id*が C のものだった場合、`xgettext`は関数 (またはマクロ)*id*の各呼び出しの最初の引数から文字列を検索します。*keywordspec*が '`id:argnum`'という形式で指定された場合、`xgettext`は呼び出しの *argnum*番目の引数を探します。*keywordspec*が '`id:argnum1,argnum2`'の形式で指定された場合、`xgettext`は呼び出しの *argnum1*番目と *argnum2*番目の引数から文字列を探して、複数形として処理すべきメッセージの singular(単数形) と plural(複数形) として扱います。同様に、*keywordspec*が '`id:contextargnumc,argnum`' や '`id:argnum,contextargnumc`'という形式で指定された場合、`xgettext`は *contextargnum*番目の引数の文字列をコンテキスト指定子 (context specifier) として扱います。そして GNOME のための特別なサポートとして、keywordspec が '`id:argnumg`'という形式で指定された場合、xgettext は argnum 番目の引数が context を伴う文字列と認識して、GNOME glib の '`"msgctxt|msgid"`'という構文を使用します。そして GNOME のための特別なサポートとして、*keywordspec*が '`id:argnumg`'という形式で指定された場合、`xgettext`は *argnum*番目の引数が context を伴う文字列と認識して、GNOME `glib`の '`"msgctxt|msgid"`'という構文を使用します。
また *keywordspec*が '`id:...,totalnumargst`'という形式で指定された場合、`xgettext`は実際の引数の数が *totalnumargs*と等しい場合のみ、この引数指定を処理します。これは C++でのオーバーロードされた関数の呼び出しなどで便利です。
最後に、もし *keywordspec*が '`id:argnum...,"xcomment"`'という形式で指定された場合、`xgettext`は指定された引数から文字列を抽出するときに、追加のコメントとして *xcomment*をメッセージに追加します。通常のシェルのコマンドラインから使用する場合は、*xcomment*を括るダブルクォーテーションはエスケープする必要があることに注意してください。
このオプションはほとんどの言語、すなわち、C、C++、Objective-C、Shell、Python、Lisp、EmacsLisp、librep、Java、C#、awk、Tcl、Perl、PHP、GCC-source、Glade、Lua、JavaScript、Vala に影響を与えます。
明示的に無効化されていない限り、常に検索されるデフォルトキーワードの指定は、言語に依存します:

- C、C++、GCC-source の場合: `gettext`、`dgettext:2`、`dcgettext:2`、`ngettext:1,2`、`dngettext:2,3`、`dcngettext:2,3`、`gettext_noop`、そして `pgettext:1c,2`、`dpgettext:2c,3`、`dcpgettext:2c,3`、`npgettext:1c,2,3`、`dnpgettext:2c,3,4`、`dcnpgettext:2c,3,4`。
- Objective C: C と同様です。`NSLocalizedString`、`_`、`NSLocalizedStaticString`、`__`も該当します。
- shell スクリプトの場合: `gettext`、`ngettext:1,2`、`eval_gettext`、`eval_ngettext:1,2`。
- Python の場合: `gettext`、`ugettext`、`dgettext:2`、`ngettext:1,2`、`ungettext:1,2`、`dngettext:2,3`、`_`。
- Lisp の場合: `gettext`、`ngettext:1,2`、`gettext-noop`。
- EmacsLisp の場合: `_`。
- librep の場合: `_`。
- Scheme の場合: `gettext`、`ngettext:1,2`、`gettext-noop`。
- Java の場合: `GettextResource.gettext:2`、`GettextResource.ngettext:2,3`、`GettextResource.pgettext:2c,3`、`GettextResource.npgettext:2c,3,4`、

gettext、ngettext:1,2、pgettext:1c,2、npgettext:1c,2,3、getString。

- C#の場合: GetString、GetPluralString:1,2、GetParticularString:1c,2、GetParticularPluralString:1c,2,3。
- awk の場合: dcgettext、dcngettext:1,2。
- Tcl の場合: ::msgcat::mc。
- Perl の場合: gettext、%gettext、$gettext、dgettext:2、dcgettext:2、ngettext:1,2、dngettext:2,3、dcngettext:2,3、gettext_noop。
- PHP の場合: _、gettext、dgettext:2、dcgettext:2、ngettext:1,2、dngettext:2,3、dcngettext:2,3。
- Glade 1 の場合: label、title、text、format、copyright、comments、preview_text、tooltip。
- Lua の場合: _、gettext.gettext、gettext.dgettext:2、gettext.dcgettext:2、gettext.ngettext:1,2、gettext.dngettext:2,3、gettext.dcngettext:2,3。
- JavaScript の場合: _、gettext、dgettext:2、dcgettext:2、ngettext:1,2、dngettext:2,3、pgettext:1c,2、dpgettext:2c,3。
- Vala の場合: _、Q_、N_、NC_、dgettext:2、dcgettext:2、ngettext:1,2、dngettext:2,3、dpgettext:2c,3、dpgettext2:2c,3。

デフォルトキーワードの指定は、'-k'オプション、'--keyword'を指定するか、*keyword-spec*を指定せずに '--keyword='として無効にすることができます。

'--flag=*word*:*arg*:*flag*'

関数 *word*の、*arg*番目の引数の一部となるような文字列のための、追加のフラグを指定します。'c-format'や、それの反対の 'no-c-format'のような、利用可能な書式文字列を示すフラグを利用でき、'pass-'を前置して指定することもできます。

--flag=*function*:*arg*:*lang*-formatは、言語 *lang*の関数 *function*の *arg*番目の引数を書式文字列とみなすという意味です (GCC 関数の属性に慣れている人は、--flag=*function*:*arg*:c-formatが、C ソース中の関数 *function*に付記される '__attribute__ ((__format__ (__printf__, *arg*, ...)))'宣言と同様だと思えばよいでしょう)。たとえば GNU libc から、関数 'error'を使用する場合、それの振る舞いについて--flag=error:3:c-formatのように指定することができます。この指定により xgettextは、すべての gettext呼び出しの *function*の *arg*番目の引数に出現する文字列を、書式指定文字列としてマークします。これは書式指定子が含まれていないような文字列にたいして 'msgfmt -c'によりチェックを行う場合に便利です。これにより翻訳者が実行時のクラッシュを引き起こすような書式指定子を意図せずに使ってしまうことを防ぐことができます。

--flag=*function*:*arg*:pass-*lang*-formatは、言語 *lang*において、書式文字列が出現しなければいけない位置に *function*呼び出しがある場合、その関数の *arg*番目の引数には、同じタイプの書式文字列となければならないという意味です。(GCC 関数の属性を知っている人は、--flag=*function*:*arg*:pass-c-formatが、C ソース中の関数 *function*に付記される '__attribute__ ((__format_arg__ (*arg*)))'宣言と同様だと思えばよいでしょう)。たとえば gettext関数の略記である '_'を使用している場合は、--flag=_:1:pass-c-formatを使う必要があります。この指定により xgettextは、_("string")呼び出しの最初の引数"string"には書式指定文字列が必

要だと伝えるために、その文字列を書式指定文字列としてマークします。これは書式指定子が含まれていないような文字列にたいして‘`msgfmt -c`’によりチェックを行う場合に便利です。これにより翻訳者が実行時のクラッシュを引き起こすような書式指定子を意図せずに使ってしまうことを防ぐことができます。
このオプションは、C、C++、ObjectiveC、Shell、Python、Lisp、EmacsLisp、librep、Scheme、Java、C#、awk、YCP、Tcl、Perl、PHP、GCC-source、Lua、JavaScript、Vala(つまり、ほとんどの言語) に影響を与えます。

‘`-T`’
‘`--trigraphs`’
入力における ANSI C の三連表記 (trigraph) を理解します。
このオプションは言語が C、C++、ObjectiveC の場合のみ効果があります。

‘`--qt`’ Qt の書式指定文字列を認識します。
このオプションは言語が C++の場合のみ効果があります。

‘`--kde`’ KDE 4 の書式指定文字列を認識します。
このオプションは言語が C++の場合のみ効果があります。

‘`--boost`’ Boost の書式指定文字列を認識します。
このオプションは言語が C++の場合のみ効果があります。

‘`--debug`’ メッセージ中の書式指定文字列を、`c-format`や`possible-c-format`フラグでマークすることにより、誰がマークしたかを表示します。後者の形式は、`xgettext`が決定したときに使用され、前者はプログラマーが決定したときに使用されます。

デフォルトでは`c-format`形式だけが使用されます。翻訳者はそれらの詳細について気にする必要はありません。

この`xgettext`の実装は、プリプロセッサーのマクロの中の文字列や、ANSI による隣接した文字列の結合、エスケープ文字による行の継続等の厄介なケースを処理することができます。

5.1.7 出力の詳細

‘`--color`’
‘`--color=`*`when`*’
色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘`--style=`*`style_file`*’
`--color`にたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘`--force-po`’
何もメッセージが定義されていない場合でも、常に出力ファイルに書き込みます。

‘`-i`’
‘`--indent`’
インデントされた形式で.po ファイルを書き込みます。

‘`--no-location`’
‘`#: `*`filename`*`:`*`line`*’のような行を書き込みません。このオプションを使用することにより、熟練した翻訳者が、どのようなコンテキストでメッセージが使用されるのかを理解するのが困難になることに注意してください。

‘`-n`’
‘`--add-location`’
: ‘`#: filename:line`’という行を生成します (デフォルト)。

‘`--strict`’
: Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘`--properties-output`’
: Java の`.properties`の書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、廃止されたメッセージを暗黙で除去することに注意してください。

‘`--stringtable-output`’
: `.strings`の書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘`-w number`’
‘`--width=number`’
: 出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘`--no-wrap`’
: 長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘`-s`’
‘`--sort-output`’
: ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

‘`-F`’
‘`--sort-by-file`’
: ファイルの場所により出力をソートします。

‘`--omit-header`’
: ‘`msgid ""`’というエントリーにたいして、ヘッダーを書き込みません。

  これはソースファイルの変更をテストする等の目的で、`.gmo`ファイルを生成するときに便利です。`--omit-header`を使用すると、同じファイルにたいして、同じオプションで`xgettext`を実行すれば、実行した時が異なっていても同じ結果を得ることができます。

  このオプションを ASCII 以外の文字が含まれたファイルにたいして使用した場合、エラーとなることに注意してください。

‘`--copyright-holder=string`’
: 出力に著作権所有者 (copyright holder) をセットします。*string*にはパッケージの著作権所有者を指定する必要があります (パッケージのソースから抽出された msgstr 文字列の著作権は、パッケージの著作権所有者に帰属することに注意してください)。翻訳者は、翻訳物の著作権を譲渡、もしくは放棄することが望まれます。これによりパッケージの

メンテナーは法的なリスクなしでそれらを配布できるのです。*string*が空の場合、出力ファイルはパブリックドメインに属するとマークされます。この場合も翻訳者は著作権を譲渡、もしくは放棄することが望まれます。繰り返しになりますが、そうすることによりパッケージのメンテナーは法的なリスクなしでそれらを配布できるのです。

*string*のデフォルト値は Free Software Foundation, Inc. です。これは単に `xgettext` が最初に使用されたのが GNU プロジェクトであることが理由です。

‘`--foreign-user`’
出力から FSF の著作権を省略します。これは‘`--copyright-holder=''`’とするのと同じです。これは GNU プロジェクト以外で、翻訳物をパブリックドメインにしたいときに便利です。

‘`--package-name=`*package*’
出力のヘッダーに、パッケージ名をセットします。

‘`--package-version=`*version*’
出力のヘッダーにパッケージのバージョンをセットします。このオプションは、同時に‘`--package-name`’を指定したときだけ効果があります。

‘`--msgid-bugs-address=`*email@address*’
msgid に関するバグの報告先アドレスをセットします。このアドレスは、翻訳者が未翻訳文字列のバグを報告するための電子メールのアドレス、または URL です。

- センテンス全体となっていないような文字列。メンテナーのためのガイドライン Section 4.3 [Preparing Strings], page 20 を参照してください。
- 不明な用語を使用したり、追加のコンテキストを理解することが必要とする文字列。
- 日付・時刻・通貨の表記で、無効な仮定をしている文字列。
- plural 化に関する問題。
- 間違った English のスペル。
- 間違った書式。

このアドレスは、あなたのメールアドレスでも構いませんし、翻訳者が登録しなくても投稿できるメーリングリストのアドレスや、翻訳者があなたに連絡をとることができるウェブページのアドレスにすることもできます。

デフォルトは空文字列が設定されており、これは翻訳者にはこれらの情報が分からないことを意味します! このオプションを指定するのを忘れないでください。

‘`-m[`*string*`]`’
‘`--msgstr-prefix[=`*string*`]`’
msgstr の値に前置する文字列として *string*(指定されていない場合は`""`) を使用します。

‘`-M[`*string*`]`’
‘`--msgstr-suffix[=`*string*`]`’
msgstr の値に後置する文字列として *string*(指定されていない場合は`""`) を使用します。

5.1.8 情報的な出力

‘`-h`’
‘`--help`’ このヘルプを表示して終了します。

‘`-V`’
‘`--version`’
バージョン情報を表示して終了します。

6 新しい PO ファイルの作成

新しい翻訳を開始する場合、翻訳者は *package*.potの初期コメント (ファイルの先頭にあります) とヘッダーのエントリー (最初のエントリーで、これもファイルの先頭付近にあります) に変更を加えたものをコピーして、*LANG*.poを作成します。

これを行う一番簡単な方法は、'msginit'を使うことです:

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

かわりにコピーしてから手で変更する方法もあります。この場合、翻訳者は *package*.potを *LANG*.poというファイル名でコピーしてから、ファイル内の初期コメントとヘッダーエントリーを修正します。

6.1 msginitプログラムの呼び出し

```
msginit [option]
```

msginitプログラムは、新しい PO ファイルを作成して、メタ情報をユーザーの環境にもとづいて初期化します。

6.1.1 入力ファイルの位置

'-i *inputfile*'
'--input=*inputfile*'
: 入力となる POT ファイルです。

*inputfile*が指定されなかった場合、カレントディレクトリから POT ファイルを検索します。'-'を指定すると、標準入力から読み込みます。

6.1.2 出力ファイルの位置

'-o *file*'
'--output-file=*file*'
: 指定された PO ファイルに出力を書き込みます。

出力ファイルが指定されなかった場合は、ユーザーのロケール設定の '--locale'オプションに依存します。'-'を指定すると、出力は標準出力に書き込まれます。

6.1.3 入力ファイルの構文

'-P'
'--properties-input'
: 入力ファイルが PO ファイルの構文ではなく、Java の.propertiesの構文にのっとった Java ResourceBundle ファイルだとみなします。

'--stringtable-input'
: 入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の.stringsの構文にのっとったファイルだとみなします。

6.1.4 出力の詳細

‘`-l` *ll_CC*’
‘`--locale=`*ll_CC*’
対象の locale を設定します。*ll*には language code を、*CC*には country code を設定する必要があります。インストールされているすべての locale のリストを出力するには、‘`locale -a`’コマンドを使用できます。デフォルトはユーザーの locale 設定が使用されます。

‘`--no-translator`’
PO ファイルが翻訳者の手で作成されたものではなく、自動的に生成されたものであることを宣言します。

‘`--color`’
‘`--color=`*when*’
色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘`--style=`*style_file*’
`--color`にたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘`-p`’
‘`--properties-output`’
Java の`.properties`の書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、廃止されたメッセージを暗黙で除去することに注意してください。

‘`--stringtable-output`’
`.strings`の書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘`-w` *number*’
‘`--width=`*number*’
出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘`--no-wrap`’
長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

6.1.5 情報的な出力

‘`-h`’
‘`--help`’ このヘルプを表示して終了します。

‘`-V`’
‘`--version`’
バージョン情報を表示して終了します。

6.2 ヘッダーエントリーを入力する

新規作成したときに初期値として入力されている、"SOME DESCRIPTIVE TITLE"、"YEAR"、および"FIRST AUTHOR <EMAIL@ADDRESS>、YEAR"などのコメントは、意味のある情報に書き換えるべきです。これはテキストエディターにより行うこともできますが、Emacs を使っていれば (拡張子を識別して) 自動的に PO モードに切り替わります。これは *M-x fundamental-mode*と入力して無効にすることができます。

ヘッダーのエントリーの変更も、PO モードで行うことができます。Emacs で、*M-x po-mode RET* と入力して、さらに *RET*を押すと、エントリーの編集が開始できるので、以下のフィールドに入力してください。

Project-Id-Version
: パッケージの名前とバージョンです。`xgettext`により入力されていない場合は入力してください。

Report-Msgid-Bugs-To
: これは `xgettext`によってすでに入力されています。未翻訳の文字列に関するバグを報告するための、電子メールアドレスか URL が含まれています:

  - センテンス全体となっていないような文字列。メンテナーのためのガイドライン Section 4.3 [Preparing Strings], page 20 を参照してください。
  - 不明な用語を使用したり、追加のコンテキストを理解することが必要とする文字列。
  - 日付・時刻・通貨の表記で、無効な仮定をしている文字列。
  - plural 化に関する問題。
  - 間違った English のスペル。
  - 間違った書式。

POT-Creation-Date
: これは `xgettext`によってすでに入力されています。

PO-Revision-Date
: これは PO ファイルのためのエディターが、ファイルを保存するときに入力される項目なので、あなたは入力する必要はありません。

Last-Translator
: 名前と電子メールアドレス（ダブルクォーテーションなし）を入力してください。

Language-Team
: 言語の英語名と、あなたが所属する language team の電子メールアドレスか、ホームページの URL を入力してください。

  重複して作業するの防ぐためだけではなく、言語に関する難しい問題を調整するためにも、翻訳を開始する前に translation team に連絡することをお勧めします。

  フリーな翻訳プロジェクトでは、それぞれの翻訳チームが、チーム自身のメーリングリストを持っています。チームの最新のメーリングリスト一覧は、Free Translation Project のホームページ (`http://translationproject.org/`) の"Teams"という場所にあります。

Language
: あなたの言語の言語コードを入力してください。以下の 3 つの形式のいずれかになります:

  - '*ll*': ISO 639 の 2 文字 (小文字) の言語コードです。コードの一覧は Appendix A [Language Codes], page 198 を参照してください。

- '*ll_CC*': '*ll*'は ISO 639 の 2 文字 (小文字) の language code、'*CC*'は ISO 3166 の 2 文字 (大文字) の country code です。いくつかの language は、異なる country で使用される方言を持っていますが、country code の仕様に冗長性はありません。たとえば'de_AT'は Austria で使用され、'pt_BR'は Brazil で使用されます。country code は方言を区別するために提供されています。コードの一覧については Appendix A [Language Codes], page 198 と Appendix B [Country Codes], page 206 を参照してください。
- '*ll_CC*@*variant*': '*ll*'は ISO 639 の 2 文字 (小文字) の language code、'*CC*'は ISO 3166 の 2 文字 (大文字) の country code、'*variant*'は variant designator です。variant designator(小文字) には、'latin'や'cyrillic'のような script designator を指定することもできます。

'*ll_CC*'の命名規則は、システムが GNU libc にもとづいて locale 名を決定する方法でもありますが、重要な違いが 3 つあります。

- PO ファイルのこの項目は locale 名とは異なり、'*ll_CC*'という組み合わせは、language の主たる方言であることを示す'*ll*'という略記であらわされます。たとえば、このコンテキストでは'de'は'de_DE'(Germany で話される German) と等しく、'pt'は'pt_PT'(Portugal で話される Portuguese) と同じです。
- PO ファイルのこの項目では、'*.encoding*'のような接尾辞は使用しません。
- PO ファイルのこの項目では、メッセージの翻訳とは関係のない、'@euro'のような variant designator は使用しません。

そのため、あなたの locale 名が'de_DE.UTF-8'の場合、PO ファイルの language specification は'de'だけになります。

Content-Type

'CHARSET'を、あなたの locale の language で使用する character encoding か UTF-8 で置き換えてください。この項目は、msgmergeと msgfmtの正しい動作のために必要です。同様に locale の character encoding が、あなたのものとは異なるユーザーにとっても必要です (Section 11.2.4 [Charset conversion], page 112 を参照してください)。

locale の character encoding は、シェルのコマンド'locale charmap'を実行して得ることができます。このコマンドの結果が'C'や'ANSI_X3.4-1968'の場合の character encoding は'ASCII'(='US-ASCII') となり、これはあなたの locale が正しく設定されていないことを意味します。そのような場合は、あなたの属する translation team に、どの charset を使用すればよいのか尋ねてください。'ASCII'は、Latin 以外の language には適用できません。

PO ファイルは、オペレーティングシステムの高度なインターナショナリゼーションの利便性に依存せずに可搬性を持たなければならないため、使用できる character encodings は GNU libcと GNU libiconvでサポートされるものに限定されています。使用できる character encoding は 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-8859-14、ISO-8859-15、KOI8-R、KOI8-U、KOI8-T、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、GEORGIAN-PS、UTF-8です。

GNU システムでは、対応する言語にたいして以下のエンコーディングが頻繁に使用されます。

- `ISO-8859-1`: Afrikaans、Albanian、Basque、Breton、Catalan、Cornish、Danish、Dutch、English、Estonian、Faroese、Finnish、French、Galician、German、Greenlandic、Icelandic、Indonesian、Irish、Italian、Malay、Manx、Norwegian、Occitan、Portuguese、Spanish、Swedish、Tagalog、Uzbek、Walloon
- `ISO-8859-2`: Bosnian、Croatian、Czech、Hungarian、Polish、Romanian、Serbian、Slovak、Slovenian
- `ISO-8859-3`: Maltese
- `ISO-8859-5`: Macedonian、Serbian
- `ISO-8859-6`: Arabic
- `ISO-8859-7`: Greek
- `ISO-8859-8`: Hebrew
- `ISO-8859-9`: Turkish
- `ISO-8859-13`: Latvian、Lithuanian、Maori
- `ISO-8859-14`: Welsh
- `ISO-8859-15`: Basque、Catalan、Dutch、English、Finnish、French、Galician、German、Irish、Italian、Portuguese、Spanish、Swedish、Walloon
- `KOI8-R`: Russian
- `KOI8-U`: Ukrainian
- `KOI8-T`: Tajik
- `CP1251`: Bulgarian、Belarusian
- `GB2312`、`GBK`、`GB18030`: Chinese の簡略表記
- `BIG5`、`BIG5-HKSCS`: Chinese の伝統的表記
- `EUC-JP`: Japanese
- `EUC-KR`: Korean
- `TIS-620`: Thai
- `GEORGIAN-PS`: Georgian
- `UTF-8`: 上記の言語を含む任意の言語

あなたの言語の翻訳に、その言語の 1 重引用符か 2 重引用符が使用されており、その locale の encoding が ISO-8859-* のいずれかの場合は、PO ファイルは locale の encoding ではなく UTF-8 encoding で作成するのが最善です。これは UTF-8 では、ISO-8859-*が持っていない実際の引用文字 (1 重引用符は U+2018 と U+2019、2 重引用符は U+201C と U+201D) が表現可能だからです。UTF-8 の locale のユーザーは実際の引用符文字列を見ることができますが、ISO-8859-*の locale では垂直方向のアポストロフィーと垂直方向のダブルクォーテーションが (文字セットの変換により) 代用で表示されます。

X11 でこれらの引用文字を入力するために、xmodmap プログラムでキーボードのマッピングを使用することができます。この場合、X11 での引用文字の名前は"leftsinglequotemark"、"rightsinglequotemark"、"leftdoublequotemark"、"rightdoublequotemark"、"singlelowquotemark"、"doublelowquotemark"になります。

UTF-8 encoding は、新しいバージョンの GNU Emacs でだけサポートされていることに注意してください。たとえば Emacs 20 with Mule-UCS や Emacs 21 では UTF-8

encoding がサポートされていますが、2001 年 1 月時点の XEmacs ではサポートされていません。

文字のエンコーディング名は、大文字または小文字で記述することができますが、通常は大文字が好まれます。

Content-Transfer-Encoding
: `8bit`にセットしてください。

Plural-Forms
: このフィールドはオプションで、PO ファイルに plural form があるときだけ必要です。これは‘`msgid_plural`’というキーワードを検索すればわかります。plural form のフィールドの書式についてはSection 11.2.6 [Plural forms], page 115 と Section 12.6 [Translating plural forms], page 131 を参照してください。

7 既存の PO ファイルの更新

7.1 msgmergeプログラムの呼び出し

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

msgmergeプログラムは、Uniforum スタイルの 2 つの.po ファイルをマージして 1 つにします。*def*.po ファイルは既存の PO ファイルで、メッセージが一致していれば既存の翻訳は新しいファイルに引き継がれます。その際、コメントは残されますが、抽出されたコメントやファイル内の位置などは破棄されます。*ref*.pot は、最新のソースより作られた PO ファイルですが、古い翻訳や、(通常はxgettextにより作成された)PO Template ファイルを参照するため、ドットコメント (訳注:プログラマーから翻訳者へのコメント#. のこと) やファイル内の位置情報は保存されますが、ファイル内のいくつかの翻訳やコメントは、破棄されるでしょう。完全に一致するメッセージが見つからない場合、より良い結果を生成するために fuzzy 一致が使用されます。

7.1.1 入力ファイルの位置

‘*def*.po’ 古いソースを参照する翻訳です。

‘*ref*.pot’ 新しいソースへの参照です。

‘-D *directory*’
‘--directory=*directory*’
ディレクトリーのリストに *directory*を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

‘-C *file*’
‘--compendium=*file*’
メッセージを翻訳するための追加のライブラリーを指定します。Section 8.4 [Compendium], page 65 を参照してください。このオプションは複数指定することができます。

7.1.2 オペレーションモード

‘-U’
‘--update’
def.po ファイルを更新します。すでに *def*.po が最新の場合は何もしません。

7.1.3 出力ファイルの位置

‘-o *file*’
‘--output-file=*file*’
指定されたファイルに出力を書き込みます。

出力ファイルが指定されていない、または ‘-’が指定された場合、結果は標準出力に出力されます。

7.1.4 更新モードでの出力ファイルの位置

処理結果は *def*.po ファイルに書き戻されます。

‘--backup=*control*’
def.po のバックアップを作成します。

‘--suffix=*suffix*’
通常使用されるバックアップの接尾辞を上書きします。

--backupオプション、もしくは環境変数 VERSION_CONTROLを通じてバージョン管理の方式を選択します。以下の値が指定できます:

‘none’
‘off’ (--backupオプションが指定されていたとしても) バックアップを作成しません。

‘numbered’
‘t’ 番号付きのバックアップを作成します。

‘existing’
‘nil’ このファイルの番号付きのバックアップがすでに存在する場合、番号付きバックアップを作成し、そうでなければ単純なバックアップを作成します。

‘simple’
‘never’ 常に単純なバックアップを作成します。

--suffixまたは環境変数 SIMPLE_BACKUP_SUFFIXが設定されていない場合は、バックアップの接尾辞として ‘~’を使用します。

7.1.5 オペレーションの修飾

‘-m’
‘--multi-domain’
def.po 内の各ドメインにたいして、*ref*.pot を適用します。

‘-N’
‘--no-fuzzy-matching’
完全に一致するものが見つからない場合、fuzzy マッチングを行いません。これにより処理のスピードが大幅に改善されます。

‘--previous’
翻訳されたメッセージをもつ古い msgid にたいして fuzzy マーカーを追加するときに、‘#|’マークをつけて古いメッセージを保持します。

7.1.6 入力ファイルの構文

‘-P’
‘--properties-input’
入力ファイルが PO ファイルの構文ではなく、Java の.propertiesの構文にのっとった Java ResourceBundle ファイルだとみなします。

‘--stringtable-input’
入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の.stringsの構文にのっとったファイルだとみなします。

7.1.7 出力の詳細

‘--lang=*catalogname*’
ヘッダーのエントリーで使用される ‘Language’フィールドを指定します。このフィールドの意味については Section 6.2 [Header Entry], page 43 を参照してください。‘Language-Team’と ‘Plural-Forms’のフィールドは変更されないことに注意してくだ

さい。このオプションを指定しない場合、‘Language-Team’フィールドから最適なものを推測して、‘Language’フィールドに入力します。

‘--color’
‘--color=*when*’
色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘--style=*style_file*’
--colorにたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘--force-po’
メッセージが何も含まれていない場合でも、常に出力ファイルに書き込みます。

‘-i’
‘--indent’
インデントされた形式で.po ファイルを書き込みます。

‘--no-location’
‘#: *filename*:*line*’という行を書き込みません。

‘--add-location’
‘#: *filename*:*line*’という行を生成します (デフォルト)。

‘--strict’
Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘-p’
‘--properties-output’
Java の.propertiesの書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、廃止されたメッセージを暗黙で除去することに注意してください。

‘--stringtable-output’
.stringsの書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘-w *number*’
‘--width=*number*’
出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘--no-wrap’
長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘-s’
‘--sort-output’
ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

‘-F’
‘--sort-by-file’
ファイルの場所により出力をソートします。

7.1.8 情報的な出力

‘-h’
‘--help’ このヘルプを表示して終了します。

‘-V’
‘--version’
バージョン情報を表示して終了します。

‘-v’
‘--verbose’
診断レベルを上げます。

‘-q’
‘--quiet’
‘--silent’
プログレスインジケーターを表示しません。

8 PO ファイルの編集

8.1 KDE の PO ファイルエディター

8.2 GNOME の PO ファイルエディター

8.3 Emacs の PO ファイルエディター

幸運にもあなたが Emacs のユーザーならば、PO ファイルの編集・変更のための快適な環境を提供するために特別に作成された PO モードがあります。PO ファイルを編集するとき PO モードを使えば、追加の PO ファイルや compendium PO ファイルを閲覧したり、PO ファイルの元となる C プログラムのソースへの参照を追跡するのが簡単になります。またプログラム中の文字列にたいして対話的に翻訳可能のマークをつけたり、PO ファイルを検証してエラーのある行を再配置するための特別な機能があります。

PO モードを使うにはまず、主要な PO モードのコマンド (Section 8.3.2 [Main PO Commands], page 52 を参照してください) 以外に、エントリー間の移動 (Section 8.3.3 [Entry Positioning], page 53 を参照してください) や、翻訳されていないエントリーの処理方法 (Section 8.3.7 [Untranslated Entries], page 57 を参照してください) を理解する必要があります。

8.3.1 GNU gettextのインストールを完了する

1 度 GNU gettextディストリビューションを入手して解凍し、configure、コンパイルしてしまえば、‘make install’コマンドで xgettext、msgfmt、gettext、msgmergeなどのプログラムや、それらが利用できるメッセージのカタログを所定の場所に配置することができます。快適なインストールの締めくくりとして、Emacs のユーザーのために PO モードを利用できるようにしましょう。

PO モードをインストールしているうちに、あなたは.emacsファイルを修正して、以下のような行を追加したいと思うことでしょう:

```
(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)
```

こうしておけば以後、.poのようなファイルや、ファイル名に‘.po.’という文字列が含まれるファイルを編集するとき、Emacs が必要に応じて po-mode.elc(または po-mode.el) をロードして、割り当てられたバッファーにたいする PO モードのコマンドが自動的に利用可能になります。PO モードがアクティブな任意のバッファーのモードラインには、*PO* という文字が表示されます。単一の Emacs セッションで、1 度に複数の PO ファイルをアクティブにすることができます。

Emacs のバージョン 20 以上を使用していて、システムに適切なインターナショナルフォントがインストールされているなら、様々な PO ファイルにたいして自動的に coding system を決定する方法を Emacs に指定することもできます。これは Emacs のスクリーンに翻訳を表示する時にしばしば、必要なフォントがロードされ使用されるということです (常にではありませんが)。これを実現するためには、あなたの.emacsファイルに以下の行を追加します:

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

これでもまだ international な character のかわりに四角が表示されるようなら、(Shift キーを押しながらマウスボタン 1 をクリックして) 違うフォントセットを試してみてください。

8.3.2 主要な PO モードのコマンド

Section 8.3.1 [Installation], page 51 で説明されているような行を追加して Emacs を設定した後は、PO ファイルを検知すると Emacs がそのウィンドウにたいして PO モードを有効にします。これによりそのウィンドウは読み取り専用となり、po-mode-map が設定されます。これはテキストモードから継承されたのではなく、純粋な Emacs のモードです。もし `po-mode-hook`に指定された関数があれば、実行されます。

あるウィンドウにたいして PO モードが有効になると、‘`PO`’という文字が、そのウィンドウのモードラインに表示されます。モードラインには PO ファイルに含まれている各種エントリーがいくつあるかも表示されます。たとえば‘`132t+3f+10u+2o`’という文字列が表示されている場合、PO モードには 132 個の翻訳済みのエントリー (Section 8.3.5 [Translated Entries], page 56 を参照してください) と、3 個の fuzzy エントリー (Section 8.3.6 [Fuzzy Entries], page 56 を参照してください)、それに 10 個の未翻訳のエントリー (Section 8.3.7 [Untranslated Entries], page 57 を参照してください) と、2 個の廃止されたエントリー (Section 8.3.8 [Obsolete Entries], page 58 を参照してください) が含まれていることを翻訳者に示しています。この際、エントリーが 0 個のものは表示されません。この例にならうと、fuzzy エントリーが解消され、未翻訳のエントリーが翻訳され、廃止されたエントリーが削除されれば、モードラインには‘`145t`’だけが表示されることになります。

主要な PO コマンドの中には、以下のセクションのカテゴリー分けに適合しないものもあります。それらのコマンドとは、PO モードや PO モードが管理するウィンドウを、特別な方法で終了する方法などです。

`_` PO ファイルにたいする最後の変更を取り消します (`po-undo`)。

`Q` 処理を終了して PO ファイルを保存します (`po-quit`)。

`q` 問い合わせの後に処理を終了します (`po-confirm-and-quit`)。

`0` 一時的に PO ファイルのウィンドウを離れます (`po-other-window`)。

`?`
`h` PO モードのヘルプを表示します (`po-help`)。

`=` PO ファイルに関する統計情報を取得します (`po-statistics`)。

`V` PO ファイル全体のフォーマットを検証します (`po-validate`)。

`_`コマンド (`po-undo`) は、Emacs の *undo* 機能と連携します。Section “Undoing Changes” in *The Emacs Editor* を参照してください。`_`を入力する度に、翻訳者が PO ファイルにたいして行った変更が少しずつ取り消されていきます。取り消し機能を実現するために、PO モードのコマンドはアトミックになっています。これは特に `RET`コマンドにたいして当てはまります。このコマンドを使用して行った 1 度の変更は、編集がいくつかの操作により行われたものだったとしても、1 度の取り消しで元に戻ります。しかし編集中のウィンドウでは、作業をより小さい単位で取り消すことができます。

`Q`コマンド (`po-quit`) と、`q`コマンド (`po-confirm-and-quit`) は、翻訳者が PO ファイルにたいする作業を終了するときに使用します。後者のコマンドは前者のコマンドに比べると冗長なコマンドです。ファイルが変更されていた場合、まずディスクにファイルが保存されます。ファイルが変更されていない場合でも、コマンドはまず PO ファイルに未翻訳のメッセージが残されていないかをチェックして、もしそのようなメッセージが残っていた場合、翻訳者は本当にこの PO ファイルにたいする作業を終了したいのか尋ねられます。これは Emacs の PO ファイルにたいするバッファーを離れるときに望ましい方法です。単にバッファーを kill する通常の *C-x k* コマンド (`kill-buffer`)) は、好ましい方法ではありません。

*O*コマンド (po-other-window) は、PO モードを一時的に離れるときに使用する、よりソフトな方法です。このコマンドはカーソルを Emacs の他のウィンドウに移動して、他のウィンドウを表示します。たとえば翻訳者が、メッセージのソース文脈中での箇所を探して、ソースのバグを修正するためだけに PO モードを開いている場合などに使用します。このコマンドにより翻訳者たる彼女は、プログラマーたる彼へと性転換を遂げ、修正したいプログラムを表示しているウィンドウにカーソルを移すことができます。後で PO ファイルのウィンドウにカーソルを戻すか、このファイルをもう一度編集するかを Emacs に指定した時に、PO モードが復元されます。

*h*コマンド (po-help) は、PO モードで利用可能なすべてのコマンドの要約が表示されます。翻訳者が任意の文字を入力することにより、通常の PO モードの操作に戻ることができます。*?*コマンドでも、*h*コマンドと同じ結果を得ることができます。

*=*コマンド (po-statistics) は、PO ファイルのすべてのエントリーを集計し、現在のエントリーが先頭から数えて何番目かと、未翻訳のエントリー数、廃止されたエントリー数等のすべての数を表示します。

*V*コマンド (po-validate) は、msgfmtの verbose mode により、編集中の PO ファイルをチェックします。このコマンドは最初に編集中の PO ファイルをディスクに保存します。GNU gettextの msgfmtは、PO ファイルの出力として MO ファイルを生成するツールで、PO モードが PO ファイル全体の書式や個々のエントリーの書式をチェックするのに、このプログラムの機能が使用されています。

msgfmtプログラムは Emacs と非同期で実行されるので、PO ファイルの評価が終わっていなくても、制御はすぐに翻訳者に戻されます。標準エラー出力への出力は Emacs により収集されて、他のウィンドウの‘*compilation*’バッファーに表示されます。Emacs の通常コマンドの *C-x‘*(next-error) や、その他の同様のコンパイル時のコマンドにより、翻訳者は素早く PO ファイル中の提示された位置に移動することができます。カーソルがエラーのある行に移動すると、翻訳者がエラーを修正するための PO モードのコマンドを選択することができます。

8.3.3 エントリーの決定

PO ファイルのウィンドウの中のカーソルは、ほとんど常にエントリ一部となります。唯一の例外は、カーソルがファイルの最後のエントリーの後ろにあったり、PO ファイルが空だったりという、特別なケースのときだけです。カーソルのある位置のエントリーのことを、カレントエントリーと呼びます。PO モードのコマンドの多くは、カレントエントリーにたいして操作を行うので、翻訳者にとってカーソルを動かすことは PO ファイルを閲覧できるだけでなく、エントリーに作用するコマンドの対象エントリーを選択することでもあるのです。

PO モードのコマンドには、特別な方法によりカーソルの位置を変更するものがあります。それらの特別な目的に対応する位置へカーソルを動かす方法については、ここで説明します。他の方法については、以降のセクションで説明します (*C-h m*で完全な一覧を得ることもできます)。

. カレントエントリーを再表示します (po-current-entry)。

n カレントエントリーの次のエントリーを選択します (po-next-entry)。

p カレントエントリーの前のエントリーを選択します (po-previous-entry)。

< PO ファイルの最初のエントリーを選択します (po-first-entry)。

> PO ファイルの最後のエントリーを選択します (po-last-entry)。

m 後で利用できるように、現在のエントリーの場所を記録します (po-push-location)。

r 以前に記録したエントリーの場所に戻ります (po-pop-location)。

`x` 現在のエントリーの場所と、以前に記録したエントリーの場所を交換します (`po-exchange-location`)。

Emacs のカーソル位置を変更するための、文字、行、paragraph、画面単位での移動や検索などのコマンドは、PO モードでカレントエントリを選択するのに使用できます。しかし PO モードには、通常の Emacs でカーソルを移動するコマンドには無いような、カレントエントリーを表示するための標準的な方法があります。`.`コマンド (`po-current-entry`) は、Emacs の画面が変更されたときなど PO モード以外の方法やでカレントエントリーが変更された時に、カレントエントリーを適切に再表示するという単一の目的のためのコマンドです。

翻訳者が作業をしているときに、PO モードによりウィンドウ配置を厳格に強制されることが、彼女を助けるものなのか、それともイライラさせるものなのかは未だ不明です。私たちは当初、ウィンドウがどのように振る舞うべきかについて明確なアイデアを持っていました。しかしその一方で、Emacs を使うとき自分で完全にコントロールできるほうを好む人もいます。固定されたウィンドウ配置は、翻訳者が有効・無効を選択できるように PO モードのオプションとして、実験的な機能として提供されるべきでしょう。もしこの機能を使う必要性や、記述する衝動をもつ人がだれもいないなら、私たちはこのアイデアを放棄するべきなのでしょう。これを行うには、プログラマーよりも翻訳者からの動機が必要です。私にとって、経験を積んだ翻訳者の意見は、他者がどうやって翻訳するか想像するしかないプログラマーの意見にくらべて、より価値があるからです。

`n`コマンド (`po-next-entry`) と `p`コマンド (`po-previous-entry`) は、カーソルをカレントエントリーの前または後のエントリーに移動します。PO ファイルの最後のエントリーにカーソルがあるときに `n`を押したり、最初のエントリーにカーソルがあるときに `p`を押しても、移動は行われません。

`<`コマンド (`po-first-entry`) と`>`コマンド (`po-last-entry`) は、PO ファイルの最初のエントリー、または最後のエントリーにカーソルを移動します。PO モードのほとんどのコマンドは、PO ファイルの最後のエントリー以降にカーソルがあるときは、‘`After last entry`’のようなエラーを戻します。`<`コマンドと`>`コマンドは、カーソルが PO ファイルのエントリーにない場合でも動作する特性があるので、このような状況をうまく解決するのに使う人もいます。しかしこれらのコマンドも、PO ファイルが空の場合は失敗します。ソースから対話的に空の PO ファイルにエントリーを追加していくように PO モードを開発するプランもあります。Section 4.5 [Marking], page 24 を参照してください。

翻訳者が特定のエントリーを翻訳する前には、そのエントリーに関連する用語や言い回しを探すために、PO ファイルの残りの部分を参照する必要があるかもしれません。もちろん彼女は Emacs の標準的な慣例にしたがって、カレントカーソルの位置をレジスターなどに保存して、後でその場所に戻るのにそのレジスターを使ったり、場所を記憶するためのリングバッファーを使うこともできます。

これらの方法にたいして、PO モードは特別なスタックにカーソルの場所を保存するという、別の方法を提供します。`m`コマンド (`po-push-location`) は、スタック上に既に保存してあるカーソル位置の情報の上に、カレントエントリーを push します。`r`コマンド (`po-pop-location`) は、スタックの最上部の要素を pop して、カーソルをその要素に関連付けられた位置へと移動します。これにより pop された要素の位置情報は失われ、次の `r`コマンドでは、その要素の 1 つ前に保存された位置にカーソルが移動します。これはスタックに保存された位置の情報がなくなるまで同じように動作します。

翻訳者がスタックの最上位の要素に関連付けられているエントリーの位置を確認してから他の場所に移動して、後で元の場所に戻る等の理由で、エントリーの場所をスタックに保存したいとき、彼女は `r`の直後に `m`を使うべきです。

`x`コマンド (`po-exchange-location`) は、カーソルをスタックの最上位の要素に関連付けられた位置に再配置すると同時に、移動する前のカレントエントリーの位置を最上位の要素に保存します。つまり、`x`コマンドを繰り返し使うと、それら 2 つのエントリーを行き来することができます。これを

行うにはまず、最初のエントリーにカーソルを移動してから *m*コマンドを使用し、その後 2 番目のエントリーで *x*コマンドを使えば、2 つのエントリー間を行き来することができます。

8.3.4 エントリー内の文字列の正規化

特定の文字列を PO ファイルのエントリーにエンコードする場合、複数行を分割したり括ったりする方法、さらには特殊な文字をバックスラッシュでエスケープする方法までもが異なっている等、とても多くの方法があります。PO モードには、特定のエンコードの文字列を `msgid`フィールドに挿入するために、既存の PO ファイルをスキャンする機能があります。PO モードにはこれらを簡単に認識するためのビルトイン機能が内部的に存在しますが、これを高速に行うのは技術的に困難です。この効率に関する問題の解決を容易にするために、わたしたちは文字列の正規表現を採択しました。

PO ファイル内の文字列の標準的な表現方法については現在も議論されていますが、PO モードでは正規表現を実験的に採用しています。`xgettext`と PO モードで、同じ文字列を統一された方法で表示するのは、PO モードで必要となる内部的な正規化が、GNU `gettext`からの `xgettext`の使用をも自動的に満たすので便利なのです。明示的な PO モードの正規化は、PO ファイルが他の場所からインポートされたときや、慣例そのものが変更されたときに必要です。

正規表現が必要な PO ファイルの文字列を正規化するために、以下の PO モードのコマンドが利用可能です:

M-x po-normalize
: エントリーをより標準化することにより、PO ファイル全体を整理します。

特別なコマンドである *M-x po-normalize*コマンド (キーは関連付けられていません) は、未翻訳のエントリーおよび翻訳済みのエントリー両方を、PO ファイル内部の標準的な引用符で括って、すべてのエントリーを修正します。このコマンドは最後のエントリーより後ろにあるゴミも削除します。このコマンドは、他の場所からインポートした PO ファイルを新たにインポートするときや、わたしたち自身がこの正規化された引用書式を改善していけるならば、有用となるでしょう。この正規化された書式は PO ファイルを整理するだけでなく、ほかの PO モードのコマンドが `msgid`から文字列を検索する処理のスピードを大幅に改善します。

*M-x po-normalize*は、エントリーにたいして 3 パスの処理を行います。最初のパスで、複数行の `msgid`と `msgstr`に、K&R C スタイルの C 文字列書式を使用している GNU `gettext` 0.6 以前の PO ファイルを発見して変換します。この発見的な処理は、廃止されたエントリーに関連付けられておらず、バックスラッシュで終端されたコメントでは失敗します。これは後続のパスで、廃止されたコメントに続くコメントを完成させる処理に依存します。この最初のパスは、すべての古い PO ファイルの調整後は行われません。2 番目と 3 番目のパスでは、すべての `msgid`と `msgstr`の文字列を、それぞれ正規化していきます。これらのパスでは XView の `msgfmt`の継続行のためのバックスラッシュも除去します。

このように明示的に正規化を指定するコマンドは、他のソースから PO ファイルをインポートするときだけではなく、現在使われている慣用句や美的観点による改善を容易にします。正規化コマンドで提案された調整を後で行うのは簡単で、最終的には他の GNU `gettext`ツールも、この適合を自動化する必要があります。Emacs を持っていないが、それでも PO ファイルを上手に手作りしたい人のために、以下では正規化された文字列の書式を説明します。

PO モードの文字列は単一行か複数行になります。文字列内に埋め込まれた改行が存在するとき、すなわち '`[^\n]\n+[^\n]`'というパターンにマッチする文字列は複数行になります。例えば以下のような文字列があったとします:

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

この文字列の空白を改行に置き換えると、以下のような文字列になります:

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

ここでは問題点を明確にするために、カリカチュアーされた例を使用して議論していきます。通常、複数行の体裁は悪いものではありません。これを処理するための実装は多分、次のような提言にしたがったものになるでしょう。すべての改行、および空行を表す改行を空文字列の中にまとめます ($n > 1$ から n-1 番目の改行は文字列を区切る改行です)。これにより文字列は以下のようになります:

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

文字列の初期化に関しては、まだ未解決の点もあります。これらの問題については、解決されたものからこのドキュメントに記載されるでしょう。

8.3.5 翻訳済みのエントリー

PO ファイル中のエントリーの `msgstr`が翻訳されて、fuzzy(Section 8.3.6 [Fuzzy Entries], page 56 を参照してください) もマークされていない場合、そのエントリーを翻訳済みのエントリーと呼びます。以後の処理では、翻訳済みのエントリーだけが GNU `msgfmt`でコンパイルされて、プログラムで利用できるようになります。他の種類のエントリーは除外され、それらにたいする翻訳は出力されません。

翻訳済みのエントリーを処理するためのコマンドがいくつかあります。

t 次の翻訳済みエントリーを検索します (`po-next-translated-entry`)。

T 前の翻訳済みのエントリーを検索します (`po-previous-translated-entry`)。

*t*コマンド (`po-next-translated-entry`) と *T*コマンド (`po-previous-translated-entry`) は、翻訳済みのエントリーを見つけて、前方または後方に移動するためのコマンドです。翻訳済みのエントリーが見つからなかった場合、PO ファイルのバッファーの先頭または終端に戻って検索します。

翻訳済みのエントリーは通常、翻訳者が翻訳を編集した結果です。Section 8.3.9 [Modifying Translations], page 59 を参照してください。ただし変数 `po-auto-fuzzy-on-edit`が `nil`でない場合、新しく翻訳されたエントリーは、公式な翻訳となる前に、最初は fuzzy エントリーになります。この場合、後でこの fuzzy エントリーの fuzzy を解消して、正式な翻訳済みのエントリーにする必要があります。Section 8.3.6 [Fuzzy Entries], page 56 を参照してください。

8.3.6 fuzzy エントリー

PO ファイルのエントリーは、一連の属性を持っています。それらは名前から得られるような性質をもち、翻訳に関するシステムコメントを明示するために使用されます。その属性 1 つが `fuzzy`で、この属性をもつエントリーが fuzzy(あいまいな) な翻訳であることを示します。この属性がつけられたエントリーのことを、fuzzy エントリーと呼びます。

通常 fuzzy エントリーは、おおよそ目的にあった翻訳であるような翻訳済みエントリーにたいして、翻訳者が見直しのために使用するものです。これらの fuzzy エントリーは、古い翻訳済みの PO ファイルを新しい PO テンプレートファイルに対応して更新するために、`msgmerge`プログラムを適

用することにより生成されることもあり、それはこのツールが、新しい msgidが、古いものをわずかに修正したものであって、新しい修正されたエントリーに古い翻訳を選択できると推測したときです。元の文字列 (msgid文字列) にたいするわずかな変更は、翻訳にも影響を与える場合があり、これは翻訳者による判断が必要です。あるエントリーにたいして msgmergeが fuzzy のマークを付与するのには、このような理由があるのです。

翻訳者が後で再検討する必要があるエントリーを覚えておくために、彼女自身の都合でエントリーを fuzzy とすることもあります。したがって特に fuzzy エントリーを処理するためのコマンドが、いくつかあります。

f 次の fuzzy エントリーを検索します (po-next-fuzzy-entry)。

F 前の fuzzy エントリーを検索します (po-previous-fuzzy-entry)。

TAB カレントエントリーの fuzzy 属性を取り除きます (po-unfuzzy)。

*f*コマンド (po-next-fuzzy-entry) と *F*コマンド (po-previous-fuzzy-entry) は、前方もしくは後方の fuzzy エントリーに移動します。fuzzy エントリーが見つからなかった場合、PO ファイルのバッファーの先頭または終端に戻って検索します。

TABコマンド (po-unfuzzy) は、エントリーに付与されている fuzzy 属性を取り除いて、通常は翻訳済みのエントリーとします。さらに、変数 po-auto-select-on-unfuzzyが nilでない場合には、TABコマンドにより自動的に他の対象となるエントリーに移動します。po-auto-select-on-unfuzzyの初期値は nil です。

po-auto-fuzzy-on-editの初期値は nilです。しかし変数 po-auto-fuzzy-on-editに tをセットすると、RETコマンドで編集したエントリーは、後から再チェックなどができるように fuzzy とマークされます。この場合、通常の使用法では、翻訳者が変更したエントリーは、(すでに fuzzy だった場合をのぞき)fuzzy エントリーに変更されることになります。彼女が翻訳に満足した場合、TABを使えば fuzzy 属性をクリアーするとともに、他のエントリーへと移動することができます。まだ翻訳が不十分だと思ったときは、SPCを使えば fuzzy 属性を保持したまま他のエントリーに移動することができます。

翻訳者が作業中のエントリーを後で見直したいようなときに見つけやすいように、翻訳済みのエントリーを fuzzy とマークする場合は、DELコマンド (po-fade-out-entry) を使うこともできます。

翻訳者が作業を終えて PO ファイルのバッファーを *q*コマンドで閉じるとき、まだ fuzzy エントリーが残っている場合は、終了してもよいか確認を求められます。

8.3.7 未翻訳エントリー

xgettextで元となる PO ファイルを作成する場合には、msgidは未翻訳の文字列で初期化され、msgstrには空文字列がセットされます。このように翻訳に空文字列がセットされているエントリーのことを、未翻訳 *(untranslated)* のエントリーと呼びます。プログラマーがプログラム内の文字列に変更を加えた場合、変更された文字列にたいする新しい未翻訳のエントリーとして PO ファイル中に現れることになります。

未翻訳のエントリーにたいしても、有効なエントリー間の移動に通常使用するコマンドと同様のレベルで考えることができます。未翻訳のエントリーは、最後に 'msgstr ""' があるので、容易に識別できます。

翻訳者の作業は (非常に簡単に表現するならば)、未翻訳のエントリーを探して編集・翻訳して、未翻訳のエントリーがなくなるまでそれを繰り返していくことではないでしょうか。特に未翻訳のエントリーを処理するためのコマンドが、いくつかあります。

u 次の未翻訳のエントリーを検索します (po-next-untranslated-entry)。

U 前の未翻訳のエントリーを検索します (po-previous-untransted-entry)。

k カレントエントリーを未翻訳にします (po-kill-msgstr)。

*u*コマンド (po-next-untranslated-entry) と *U*コマンド (po-previous-untransted-entry) は、前方もしくは後方の未翻訳のエントリーに移動します。未翻訳のエントリーが見つからなかった場合、PO ファイルのバッファーの先頭または終端に戻って検索します。

*k*コマンド (po-kill-msgstr) は、単に翻訳された文字列を空文字列にすることによって、エントリーを未翻訳のエントリーにするコマンドです。Section 8.3.9 [Modifying Translations], page 59 を参照してください。

翻訳者が作業を終えて PO ファイルのバッファーを *q*コマンドで閉じるとき、まだ未翻訳のエントリーが残っている場合は、終了してもよいか確認を求められます。

8.3.8 陳腐化したエントリー

PO ファイルの陳腐化したエントリーとは、msgmergeによりローカライズされるパッケージ内で、その翻訳がもはや必要ないのでコメントアウトされているエントリーのことです。

陳腐化したエントリーにたいしても、有効なエントリー間の移動に通常使用するコマンドと同様のレベルで考えることができます。行に msgidや msgstrが含まれているか否かに関係なく、行が#で開始されているという事実により、陳腐化したエントリーを識別できます。

再初期化するために翻訳を空文字列に置き換えて、元の未翻訳の空文字列にするコマンドがあります。これらのコマンドは Emacs の kill リングと互換性があるので、以前に kill リングに保存された文字列を翻訳として挿入することもできます。またユーザーは翻訳を対話的に編集することができます。これらすべてのコマンドは廃止されたエントリーの編集にも適用できますが、エントリーは廃止された状態のままになります。

陳腐化したエントリーに特化したコマンドがいくつかあります。

o 次の陳腐化エントリーを検索します (po-next-obsolete-entry)。

O 前の陳腐化したエントリーを検索します (po-previous-obsolete-entry)。

DEL 有効なエントリーにたいしては、それを陳腐化したエントリーにします。陳腐化したエントリーの場合は、エントリーを削除します (po-fade-out-entry)。

*o*コマンド (po-next-obsolete-entry) と *O*コマンド (po-previous-obsolete-entry) は、前方もしくは後方の陳腐化したエントリーに移動します。陳腐化したエントリーが見つからなかった場合、PO ファイルのバッファーの先頭または終端に戻って検索します。

PO モードには、陳腐化したエントリーにたいして、そのエントリーを非コメント化することにより有効なエントリーにする方法は用意されていません。用意されていない理由は、元となる未翻訳の文字列と、プログラム中の文字列の対応をとることができなくなるからで、これは msgid 駆動の哲学と反するからです。

とはいえ有効なエントリーをコメントアウトして、陳腐化したエントリーとすることは可能です。後で GNU gettextユーティリティーが処理するとき、翻訳が見つからなければ未翻訳の文字列が使用されます。DELコマンド (po-fade-out-entry) は、カレントエントリーを消滅の方向へと押しやるコマンドです。有効なエントリー (翻訳されたエントリー) の場合には、そのエントリーを fuzzy エントリーにします。すでに fuzzy エントリーの場合には、確認後にそのエントリーをコメントアウトします。すでに廃止されたエントリーの場合には、そのエントリーを PO ファイルから削除します。削除した翻訳を、他の PO ファイルの、(通常は) 未翻訳のエントリに再使用するのは簡単です。Section 8.3.9 [Modifying Translations], page 59 を参照してください。

今後 PO モードを開発していく上で、あなたを寝不足とさせるような、解決すべき興味深い問題が存在します。PO モードをよりよくするこのアイデアとは、新しく出現した文字列にたいする翻訳として、すべての陳腐化したエントリーの中から最適な候補を推測することです。これはアルゴリズム的に解決するには困難な問題であり、文字列の相似をより効果的に計測するための開発を行う必要があると私は考えています。現在ではこれらの作業は翻訳者がすべて決定しなければなりませんが、いつの日か陳腐化したエントリーから翻訳を検索することができる便利なツールを提供できるように努力しています。

8.3.9 翻訳の修正

PO モードは、通常 Emacs のバッファーを変更するような方法で PO ファイルを直接編集することを防ぎます。そうすることで、直接編集してファイル全体のフォーマットや文字列の引用符を誤って編集してしまう等の、容易に発生し得るエラーを防ぎます。他の種類のエラーもありますが、それらのエラーは翻訳者が *V*コマンドを使っていつでも、バッチ検証プロセスにより発見・診断することができます。その他のエラーについては、翻訳者自身の判断と、彼女が翻訳したパッケージにたいする同じ母国語ユーザーによる、言語的な判定に頼る必要があります。

翻訳を作成し、機械的な診断およびユーザーによる報告を経た後、翻訳者は以下のコマンドを使って翻訳を変更します。

`RET` 翻訳を対話的に編集します (`po-edit-msgstr`)。

`LFD`
C-j 翻訳を元の未翻訳の文字列で再初期化します (`po-msgid-to-msgstr`)。

k 翻訳を kill リングに保存してから、削除します (`po-kill-msgstr`)。

w 翻訳を kill リングに保存するだけで、削除はしません (`po-kill-ring-save-msgstr`)。

y 翻訳を kill リングのもので置き換えます (`po-yank-msgstr`)。

`RET`コマンド (`po-edit-msgstr`) は、新しい翻訳を編集したり既存の翻訳を変更するための、新しい Emacs のウィンドウをオープンします。新しいウィンドウには PO ファイルのカレントエントリーの、翻訳のコピーが含まれています。翻訳のコピーは、すぐに編集できるように引用符を除かれていて、Emacs による編集コマンドのすべてが使用できます。翻訳者が文字列の変更を終えたら、*C-c C-c* により、自動的に引用符を付加した形式で結果を保存し、編集用のサブウィンドウを閉じることができます。変更を保存せずに取り消す場合には、*C-c C-k* を使用してください。詳細は、Section 8.3.11 [Subedit], page 62 を参照してください。

`LFD`コマンド (`po-msgid-to-msgstr`) は、翻訳を元の文字列で初期化します。このコマンドは通常、翻訳者が以前の作業を破棄して、元の文字列にたいして新しく翻訳をやり直したいときに使用します。

未翻訳のエントリーを編集するときに、常に `LFD`コマンドを自動的に実行させることもできます。`po-auto-edit-with-msgid`に `t`をセットすれば、翻訳に何も文字列が設定されていない場合には、元の文字列により翻訳が初期化されます。デフォルトでは `po-auto-edit-with-msgid`は `nil`です。

実際のところ、空の文字列から翻訳を開始するのか、それとも元の文字列のコピーから翻訳を開始するのかは好みの問題です。元の言語と、翻訳する言語があまりに異なっている場合には、単に空の文字列から開始するのがよいでしょう。その反対に元の言語と翻訳する言語が似ている場合には、元の文字列の数字や文字を再入力する手間を省きたいときもあるでしょう。未翻訳の余分な元文字列を取り除く手間がかかるとしても、彼女は元の文字列を見ながら未翻訳の文字列を翻訳で上書きしていく方法を好むかもしれません。

これにより、空文字列になる前の内容は、kill リングと呼ばれる特別な場所に置かれます。w コマンド (po-kill-ring-save-msgstr) も、翻訳を kill リングにコピーする効果に違いはありませんが、エントリーをそのままにする点が異なります。この場合、エントリーから翻訳は削除されません。どちらのコマンドも、Emacs 愛好家にはよく知られている共有バッファーである、Emacs の kill リングを使用します。

翻訳者は作業する過程で、`k`や `w`を多く使うことでしょう。それにともない kill リングには翻訳が保存されていきます。kill リングに保存された文字列は、後で Emacs の他のバッファーに挿入することができます。kill リングは、単一の PO ファイル内の異なるエントリー間だけではなく、翻訳者が PO ファイルを複数開いている場合は、異なる PO ファイル間で翻訳文字列を移動するのに使用されます。

PO モードではないバッファーと文字列をやりとりするのを容易にするために、`k`コマンドで kill リングに置かれた翻訳文字列は、引用符が取り除かれて保存されます。すなわち、文字列を囲うための引用符は取り除かれ、複数行の文字列は結合され、バックスラッシュでエスケープされた文字は対応する実際の文字に変換されます。陳腐化したエントリーの場合、保存される前に翻訳は非コメント化されます。

`y`コマンド (`po-yank-msgstr`) は、カレントエントリーの翻訳を kill リングの文字列で完全に置き換えます。Emacs の用語にしたがうと、置き換えた文字列は、PO ファイルのバッファーへ *yank(yanked)* されたといいます。Section “Yanking” in *The Emacs Editor* を参照してください。最初に `y`を使用したときは、kill リングに最後に追加された値が翻訳として戻されます。他のキーを押さずに、もう一度 `y`をタイプすると、kill リングの最後から 2 番目に追加された文字列が、翻訳として挿入されます。`y`を何度も繰り返すことにより、望む文字列が見つかるまで、kill リングに保存された文字列を巡回することができます。

文字列が PO ファイルのエントリーに yank されるときには、自動的に PO ファイルの書式にしたがった形式の引用符が付与されます。さらに陳腐化したエントリーの場合には、文字列は適切にコメント化されます。プログラムが使用できるように、翻訳された個々の文字列に引用符を付与するために、翻訳者が患わされることはありません。

`k`と `w`だけが、文字列を kill リングに保存するコマンドではないことに注意してください。PO モードの多くのコマンドは、翻訳された文字列 (または翻訳者のコメント) を置き換えて、自動的にリングに保存します。この一般的なルールに当てはまらないコマンドは、yank コマンド自身です。

文字列の kill と yank については、一般的な状況の実例で説明したほうがよいでしょう。プログラマーが文字列にちょっとした変更を加えたとしましょう。その後、彼が行った変更は、変更した文字列にたいする新しい未翻訳のエントリーとして PO ファイルに出現し、元の変更されていない文字列にたいする翻訳は、陳腐化したエントリーとなります。多くの場合、翻訳者は未翻訳エントリーの `msgstr`に、陳腐化したエントリーの変更前の翻訳を流用することで作業を節約できるでしょう。その後、陳腐化したエントリーが必要ないなら、安全に削除することができます。

翻訳者が未翻訳のエントリーを見つけて、それが既存の翻訳と少ししか違わないのではないか、と思ったとしましょう。そのような場合は、すぐにカレントエントリーの場所を `m`でマークしてから、陳腐化したエントリーを検索して、変更される前の文字列にたいする翻訳を探すために `o`を使用します。見つかったら、DEL コマンドで廃止されたエントリーを削除します。なぜなら彼女は DEL コマンドが翻訳を kill することを知っており、それはつまり翻訳が kill リングに保存されることを知っているからです。その後 `r`コマンドで最初の未翻訳エントリーに戻り、保存した翻訳を `y`コマンドで `msgstr`に *yank* します。これで翻訳者は、`RET`を使って自由に翻訳内容を調整することができます。そしてその後は再び `u`と `m`で次の未翻訳の文字列を探していくのかもしれません。

翻訳者が同じキーシーケンスを何度も使用する必要があるときには、要求したときにそのキーシーケンスを再生させる Emacs の機能について学習するほうがよいかもしれません。Section “Keyboard Macros” in *The Emacs Editor* を参照してください。

8.3.10 コメントの修正

翻訳とは、言語的な難しさをともなう作業です。翻訳においてどのような決定をしたのか、その選択に関してドキュメントを残す必要があるでしょう。これらのドキュメントは、翻訳者のコメントとして PO ファイルに保存されます。これは、翻訳者が自由に作成・削除、または変更ができるコメントで、彼女が後で PO ファイルを見直すときなどに便利です。

最初の‘#’の後に空白がないコメント、たとえば‘#.’や‘#:’ではじまるコメントは、翻訳者のコメントではありません。これらは、gettextツールにより作成されたコメントです。それらのシステムが追加したコメントは、翻訳者が変更するべきではないコメントなので、以下で説明するコマンドの対象外です。Chapter 3 [PO Files], page 13 を参照してください。

以下のコマンドは翻訳を変更するコマンドと似ているので、一般的な原則は同様に適用できます。Section 8.3.9 [Modifying Translations], page 59 を参照してください。

`#` 翻訳者のコメントを対話的に編集します (`po-edit-comment`)。

`K` 翻訳者のコメントを kill リングに保存してから、削除します (`po-kill-comment`)。

`W` 翻訳者のコメントを kill リングに保存するだけで、削除はしません (`po-kill-ring-save-comment`)。

`Y` 翻訳者のコメントを、kill リングのもので置き換えます (`po-yank-comment`)。

これらの、翻訳文字列を変更するための PO モードの類似コマンドは、翻訳文字列の代わりに翻訳者のコメントを処理する以外は、同じように動作します。詳細はすでに説明済みなので、以下ではこれらのコマンドを簡単に説明します。Section 8.3.9 [Modifying Translations], page 59 を参照してください。

`#`コマンド (`po-edit-comment`) は、PO ファイルのカレントエントリーにたいする翻訳者コメントのコピーを含む、新しい Emacs ウィンドウをオープンします。エントリーにそのようなコメントがない場合、PO モードは翻訳者がエントリーにコメントを追加したいと解釈し、空のスクリーンが表示されます。編集前にコメントマーク (`#`) とそれに続くスペースは自動的に削除され、編集後に自動的に再付加されます。陳腐化したエントリーにたいする翻訳者コメントは、非コメント化とコメント化の操作が 2 度行われます。編集ウィンドウで *C-c C-c* キーを押すと、コメントの編集を終了します。詳細については、Section 8.3.11 [Subedit], page 62 を参照してください。

`po-subedit-mode-hook`に関数が登録されている場合には、編集バッファーに文字列が挿入されたときに実行されます。

`K`コマンド (`po-kill-comment`) は、翻訳者コメントを kill リングに保存してから削除します。`W`コマンド (`po-kill-ring-save-comment`) は、翻訳者コメントを kill リングにコピーするだけで、カレントエントリーのコメントは変更しません。`Y`コマンド (`po-yank-comment`) は、翻訳者コメントを kill リングの文字列で置き換えます。このコマンドを繰り返し入力すると、挿入されたコメントは kill リングに保存された他の文字列で順に置き換えられます。

kill リングの文字列は、すべて同じ性質をもちます。翻訳された文字列と翻訳者のコメントに違いはありません。たとえば翻訳者が翻訳を終了したとき、以前の翻訳の何が悪かったのかをドキュメント化して覚えておこうと、コメントを付与したい場合を考えます。彼女は翻訳者コメントで、以前の翻訳を引用したいと思うのではないでしょうか。それを行うには、まず翻訳者コメントを、kill リングに残っている以前の翻訳で初期化するでしょう。すでに kill リングに保存されている以前の翻訳を

使って編集するには、#の前に *M-w*とタイプすれば、以前の翻訳が kill リングに保存されるので、それに説明文などを追加すればよいでしょう。

すでに何らかの翻訳者コメントがあり、そのコメント全体を置き換えるのではなく翻訳者がコメントを追加したい場合を考えてみましょう。その場合には#でコメントを編集する必要があります。編集ウィンドウが開いたら、Emacs の標準コマンドの *C-y*(yank) と *M-y*(yank-pop) で、以前の翻訳を取得できます。

8.3.11 サブエディションの詳細

PO subedit マイナーモードは、ここで詳細な説明をする価値のある特殊なモードです。これにより Emacs の通常の編集コマンド以外に、以下で説明するコマンドがインストールされます。

C-c C-c 編集を完了します (po-subedit-exit)。

C-c C-k 編集を中止します (po-subedit-abort)。

C-c C-a 追加 (auxiliary) の PO ファイルを参照します (po-subedit-cycle-auxiliary)。

ウィンドウにはメッセージにたいする翻訳、もしくは翻訳者コメントが表示されます。翻訳者は自分の思うように、このウィンドウ内のコンテンツを変更します。作業が終わったら、*C-c C-c* コマンド (po-subedit-exit) を使えば、バッファーが切り替えられていたり、表示されていなくても、編集した翻訳で元の翻訳を置き換えて PO ファイルに反映することができます。

kill 翻訳者が自分の翻訳 (または翻訳者コメント) に満足できなくて、RETコマンド (または#コマンド) を押す前の状態に戻したい場合には、*C-c C-k* コマンド (po-subedit-abort) を使えば、編集したものを破棄して、元の翻訳 (または翻訳者コメント) に戻すことができます。他にも、普通に *C-c C-c* で編集を終了してから、U(訳注：Undo をするコマンドが U コマンドと記述してあるが、Emacs の Undo コマンドである Ctrl+_コマンドの間違いではないか) で元のバージョンに戻す方法があります。

C-c C-a コマンド (po-subedit-cycle-auxiliary) は、カレントエントリーの翻訳を編集しているとき、すでに他の言語へ翻訳されたメッセージに目を通したいときに使用します。このコマンドは翻訳者が複数の言語に通じているときなどに便利でしょう (もちろん利用可能な追加の PO ファイルがある場合ですが (Section 8.3.13 [Auxiliary], page 64 を参照してください)。

po-subedit-mode-hookに関数が登録されている場合には、編集バッファーに文字列が挿入されたときに実行されます。

編集中には、翻訳文字列の最後で意図せず RET(改行) キーを入力したり、必要な改行を誤って削除していまわないよう注意を払う必要があります。そのような文字が編集バッファーで非表示になっていると、容易に間違いを犯してしまいます。そのような間違いが起きないように、RETコマンドでは、編集している文字列の最後に自動的に<が付加されます。この<は実際のメッセージ文字列ではありません。*C-c C-c* で編集ウィンドウを閉じると、PO モードは自動的にそのような<文字を削除して、適切な空白文字に置き換えます。翻訳者が末尾の<の後ろに文字を追加すると、<は区切り文字としての性質を失って、翻訳文字列の一部となります。<を削除した場合には、編集文字列はそのまま評価され、たとえ非表示であったとしても、末尾に改行があればそれもそのまま評価されます。翻訳した文字列が本物の<で終わる場合には、区切り文字の<も削除されずに表示されるので、編集ウィンドウの文字列の終端には 2 つの<が表示されます。

翻訳 (またはコメント) を編集するとき、翻訳者はカーソルを PO ファイルのバッファーに戻してから、エントリーを表示するために自由に他のエントリーに移動を行えます。編集を保留して、PO ファイルバッファーの他の箇所に移動したり、他のエントリーの編集をはじめることもできます。それぞれのエントリーは、それら自身のサブエディットバッファーで編集されます。1 つのエントリーにたいする特定の翻訳やコメントを同時に編集したり、異なる PO ファイルのエントリーを同時に編

集することも可能です。すでに編集中のエントリーにたいしてRETをタイプすると、単にそのエントリーの編集を再開します。Emacs の複数のウィンドウの扱いに慣れれば、翻訳者はより快適になるでしょう。

保留したサブエディットの完了または中止は、編集を開始した順番に関わらず任意の順番で行うことができます。複数のサブエディットを保留している状態で、(*q*コマンドで)PO ファイルを閉じようとすると、サブエディットが 1 つずつ順番に再開されるので、翻訳者それら個々について決定していくことができます。

8.3.12 C ソースのコンテキスト

PO モードは、GNU gettextユーティリティーで作成された PO ファイルの場合、それらのユーティリティーが生成した PO ファイルに特別なコメントを挿入するので、特に威力を発揮します。それらの特別なコメントの中には、PO ファイルのエントリーの未翻訳の文字列が、プログラムのソース中で出現する位置を示すものがあります。

翻訳者が未翻訳の文字列を翻訳するとき、その元文字列があまりに簡潔すぎたり、不可解あったり、曖昧である等、通常のように有効でない場合があります。そのような文字列をどのように翻訳するか決める前に、その文字列が本当は何を意味するのか、そしてそれにぴったりな翻訳は何なのかを理解する必要があります。このような問題を判断するために残された唯一の方法は、プログラムのソースからその文字列の場所を探し、その周辺に残されたプログラマーのコメントや、他に助けになりそうな何かを探すことに時間を割くことです。

翻訳者が有能なプログラマーである場合、プログラムのソースを見ることにより多くの助けを得ることができるでしょう。しかしプログラミングに精通していなくて、C のコードを見ると不安な気持ちになったとしても、恥ずかしがらずにたまにはソースを見てみましょう。そうすれば彼女が必要とする何らかのヒントを得られるようになれるでしょう。プログラマーのコメント、そして (彼が適切な名前をつけていれば) 変数名や関数名、プログラムコード自体の全体的な構成などに注意を払って学習することにより、すぐにプログラムのコードを見ても違和感を感じないようになるでしょう。

以下は、翻訳者が PO ファイルのエントリーから、プログラムのソースコンテキストを参照するのに助けとなるコマンドです。

s プログラムのソースコンテキストを表示、またはソースコンテキストのサイクル表示を再開します (po-cycle-source-reference)。

M-s メニューで選択されたプログラムソースのコンテキストを表示します (po-select-source-reference)。

S ソースファイルの検索パスにディレクトリを追加します (po-consider-source-path)。

M-S ソースファイルの検索パスからディレクトリを削除します (po-ignore-source-path)。

*s*コマンド (po-cycle-source-reference) と *M-s*コマンド (po-select-source-reference) は、どちらも他のウィンドウを開いてプログラムのソースファイルの、翻訳しようとしている文字列が使用されている場所を表示します。このように、これらのコマンドは文字列にたいするソースプログラムのコンテキストを与えます。しかしエントリーがコンテキストへの参照を保有していなかったり、検索パスにあるプログラムソースでは参照が解決されない場合、コマンドはその旨をエラーとして表示します。

s(または *M-s*) も新しいウィンドウをオープンしますが、カーソルは PO ファイルのウィンドウに留まったままです。翻訳者がプログラムソースのウィンドウに移動したい場合には、明示的に O コマンドを使用する必要があります。

はじめて *s*を使用するときや、PO ファイルのエントリーのソースコンテキストが直前に取得したものと異なるときには、コマンドはこのエントリーにたいして利用可能な、最初のコンテキストを返

します。すでにその PO ファイルのカレントエントリーにたいする、何かしらのコンテキストを表示していて、さらに他のものを探したいときには最後に表示したコンテキストのウィンドウで *s*を入力することにより、検索を再開できます。このコマンドにより、翻訳者がソースファイルのコンテキストからカーソルを移動していた場合には、カーソルがコンテキストの場所に戻されます。他のコマンドを入力しないで *s*コマンドを連続して入力すると、PO モードはこのエントリーにたいして利用可能なコンテキストを順々に表示していき、最後のコンテキストを表示すると、また最初のコンテキストに戻って表示します。

*M-s*コマンドは異なる動作をします。このコマンドは参照を循環して表示せずに、いくつか存在する参照のうちから 1 つを翻訳者に選択させます。翻訳者が *M-s*で表示される質問にたいして、すぐに TAB を押すと、翻訳者が適切なものを選べるように利用可能なすべての参照メニューが表示されます。このコマンドは翻訳する 1 つの文字列にたいして、多数の利用可能なコンテキストが存在するときに有用です。

プログラムのソースファイルは通常、PO ファイルの場所から相対的に見つけることができます。この検索が失敗したときには、特別なケースとして PO ファイルの 1 つ上のディレクトリーからの相対パスのファイルも検索対象になります。これらの 2 つのケースを考えておけば、大抵の PO ファイルを処理することができます。しかし PO ファイルが移動されていたり、通常あるべき場所とは異なる場所で編集されているときには検索が失敗します。このような場合には、翻訳者が PO モードにたいして、PO ファイルが本来どのディレクトリーにあるのかを、伝える必要があります。そのように指定したディレクトリーのことをまとめて、プログラムソースの検索パスと呼びます。*S*コマンド (po-consider-source-path) は、検索パスに新しいディレクトリーを対話的に入力するために使用され、*M-S*コマンド (po-ignore-source-path) は、検索パスから削除したいディレクトリーを選択して削除するのに使用されます。

8.3.13 追加 PO ファイルを調べる

PO モードには、複数の言語に通じている翻訳者が、彼女の知っている言語への既存の翻訳を利用するための機能があります。この機能は、他の言語への翻訳を追加のコンテキストとして、彼女の作業に提供することができます。また一度に複数の言語への翻訳を作成したいような場合にも、翻訳者にたいして作業を容易にするための機能をもっています。

追加 *(auxiliary)* の PO ファイルとは、翻訳者が作業するパッケージの、他の言語用の既存の PO ファイルのことです。追加の PO ファイルを定義・処理したり、作業中のエントリーのコンテキストを表示するためのコマンドが存在します。

以下は、PO モードで利用可能な、追加の PO ファイルのコマンドです。

`a` 追加の PO ファイルから、同じエントリーにたいする他の翻訳を探します (po-cycle-auxiliary)。

C-c C-a 追加の PO ファイルを指定して、それに切り替えます (po-select-auxiliary)。

A 表示している PO ファイルを、追加の PO ファイルとして定義します (po-consider-as-auxiliary)。

M-A 表示している PO ファイルを、追加の PO ファイルのリストから削除します (po-ignore-as-auxiliary)。

*A*コマンド (po-consider-as-auxiliary) は、現在の PO ファイルを、追加の PO ファイルのリストに追加し、*M-A*コマンド (po-ignore-as-auxiliaryは、リストから削除します。

aコマンド (po-cycle-auxiliary) は、すべての追加 PO ファイルを一つずつ走査して、カレントエントリーと同じ msgidにたいする、他の言語に翻訳されたエントリーを検索するコマンドです。

PO ファイルが見つかったら、その PO ファイルが現在のウィンドウに表示されます (そのウィンドウがもっとも前面に表示されます)。追加の PO ファイルに作業中の PO ファイルが含まれていない場合は、これらの処理を行う前に追加しておくとよいでしょう。このようにしておけば、aコマンドで検索された他言語の PO ファイルがウィンドウに表示されても、aコマンドを繰り返し入力して、元の PO ファイルに戻ることができるからです。

*C-c C-a*コマンド (`po-select-auxiliary`) は、翻訳者にたいして追加の PO ファイルを補完付き入力で選択させて、その PO ファイルに切り替えるコマンドです。選択した PO ファイルにカレントエントリーと同じ `msgid`があった場合は、そのエントリーをカレントエントリーとします。同じエントリー存在しない場合には、カーソルは元の位置から変更されません。

この機能が完全に動作するためには、`msgid`が、同じ方法で正確に、正規化されて記述されている必要があります。たとえ文字列を記述する方法は異なっていても `msgid`に同じ文字列が設定されていれば問題はありませんが、違う文字列が記述されていると、PO モードの追加 PO ファイル関連のコマンドの動作が損なわれてしまいます。しかしほとんどの PO ファイルの `msgid`は、同じ GNU `gettext`ツールで書き込まれたものなので、実際には問題になることはないでしょう。

しかしソースファイルの文字列をマークしながら、PO モードで一から作成した PO ファイルは、異なる形式で正規化されています。そのために '`M-x normalize`'コマンドを PO ファイルに適用するのです。PO モードと他の GNU `gettext`ツール間の矛盾を解決するまでは、翻訳者は正規化の問題に留意してください。

8.4 翻訳 compendia の使用

*compendium(*要約*)* とは、多くのパッケージで繰り返し使用される翻訳を含んだ特別な PO ファイルのことです。翻訳者は gettext ツールを使って、新しい compendium を構築して、compendium に含まれた翻訳から、エントリーを compendium に追加したり、未翻訳エントリーの初期化、既存の翻訳済みエントリーの更新できます。

8.4.1 compendia の作成

基本的に、すべての PO ファイルに含まれる翻訳済みエントリーだけを、有効な compendium として定義できます。翻訳者が特別な compendia を所有したい場合があります。連結 *PO* ファイル *(concatenating PO files)* と *PO* ファイルからメッセージを抽出したサブセット *(extracting a message subset from a PO file)* という、2 つのケースを考えてみましょう。

8.4.1.1 PO ファイルの連結

複数の有効な PO ファイルを、1 つの compendium ファイルに連結するためには、'`msgcomm`'か、'`msgcat`'(推奨) を使用することができます:

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

デフォルトでは '`msgcat`'は、同じ文字列にたいして異なる翻訳がある場合には、それらの翻訳を蓄積します。これらの複数の翻訳には `fuzzy`マークが付与されるとともに、目立つように装飾されます。たとえば以下のような 2 つのファイルがあるとします。`file1.po`は以下のような内容です:

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

そして `file2.po`です:

```
#: src/bye.c:100
```

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

これらにたいして msgcatを呼び出すと、以下のような結果になります:

```
#: 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"
```

“競合” は翻訳者が手動で解決する必要があります。彼女は最初のバージョンが適しているのか、それとも 2 番目のバージョンなのか (それとも新しい翻訳を提供する必要があるのか) を決定して、“マーカー行” を削除し、fuzzyマークをはずす必要があります。

最初に検索される翻訳済みのメッセージが常に最善の翻訳であることを翻訳者が知っている場合は、‘--use-first’スイッチを使用できます:

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

よい compendium ファイルを作るには、fuzzyや未翻訳エントリーを含めてはいけません。入力ファイルがそのようなエントリーで “汚染” されている場合は、‘msgattrib --translated --no-fuzzy’を使って入力ファイルを前処理するか、結果ファイルを後処理しなければなりません。

8.4.1.2 PO ファイルからのメッセージサブセットの抽出

同じメッセージを何度も翻訳したいと思う人はいないでしょう。たとえば、あなたが getopt.cのメッセージを含んだ compendium ファイルが欲しいと思うかもしれません。

既存の PO ファイルから 1 つの compendium にメッセージのサブセット (例: getopt.cのすべてのメッセージ) を抽出する場合は、‘msggrep’を使用できます。

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

8.4.2 compendia の使用

compendium ファイルを使用して、スクラッチから翻訳を初期化したり、既存の翻訳を更新できます。

8.4.2.1 新しい翻訳ファイルの初期化

まだ翻訳された PO ファイルが存在しないときは、“古い” 翻訳済みファイルとして/dev/nullを使用できます。

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

8.4.2.2 既存の翻訳ファイルの更新

compendium ファイルと既存の PO ファイルを結合した後、それをマージして POT ファイルを作成し、陳腐化したエントリーを削除します (これは任意です。ここでは ‘msgattrib’が使用されています)。

```
msgcat --use-first -o update.po compendium1.po compendium2.po file.po
msgmerge update.po file.pot | msgattrib --no-obsolete > file.po
```

9 PO ファイルの操作

PO ファイルを手で扱うよりは、自動的な方法で取り扱うほうがよいときがあります。GNU gettextには、この目的のための完全なツールが含まれています。

2 つのパッケージを 1 つのパッケージにマージするときには、元の 2 つのパッケージの POT ファイルが結合されたものが、マージされたパッケージの POT ファイルになります。したがってメンテナーは、翻訳された各言語ごとに、既存の 2 つの翻訳済みパッケージを 1 つの翻訳カタログにマージしなければなりません。これを行うには 'msgcat'を使うのが最善です。マージにより発生し得る競合を解決するのは、翻訳者の役目となります。

ある翻訳者が他の翻訳者から作業を引き継ぐときに、彼女がその locale の異なるエンコーディングを使っている場合には、カタログの文字のエンコーディングを変換することになるでしょう。これを行うには 'msgconv'プログラムを使うのが最善です。

メンテナーが他のパッケージからタグ付けされたメッセージを取得するとき、彼はこのソースファイルの既存の翻訳も取り込む必要があります (翻訳者が同じ作業をしなくても済むように)。これを行うには 'msggrep'を使う方法と、そのソースファイルから POT ファイルを作成して 'msgmerge'を使う方法があります。

翻訳者がある翻訳カタログを特定の方言や正書法に適応させたいとき – たとえば Switzerland で記述された German を、Germany で記述された German に適応させる場合など – 彼女はカタログの中のすべてのメッセージに適用できるテキストプロセッサーが必要にるでしょう。これを行うためのツールが、'msgfilter'です。

msgfilterの他の使い方としては、PO ファイルが作成される元となった POT ファイルに近いものを生成することです。これは、'msgfilter sed -e d | sed -e '/^# /d''のようなフィルターコマンドにより行うことができます。オリジナルの POT ファイルには異なるコメントがあったり、plural message の数も異なります。この理由により、利用可能ならオリジナルの POT ファイルを使うほうがよいことに注意してください。

翻訳者が翻訳をチェックしたいとき、たとえば正書法のルールや非対話型のスペルチェッカーにしたがってチェックをしたいときは、'msgexec'を使うことができます。

サードパーティー製のツールにより PO、または POT ファイルを作成するとき、重複が無視されるときがあります。しかし GNU gettextツールは、同じファイル中に同じドメインで重複した msgid がある場合にはエラーとなります。重複をマージするためには、'msguniq'を使うことができます。

複数のファイル間での重複を維持 (または破棄) するための、より一般的なツールとしては 'msgcomm'があります。

翻訳カタログが完全に翻訳されているかをチェックするには、'msgcmp'を使うことができます。

翻訳カタログから fuzzy や未翻訳のメッセージだけを選択・抽出するためには、'msgattrib'を使うことができます。

English の翻訳カタログを準備するための最初のステップとしては、'msgen'が便利です。これは、各メッセージの msgid を msgstr にコピーします。

そして最後に、これらの様々なアプリケーションでも十分でない場合には、PO ファイルを取り扱う特殊なプログラムを記述するために使用できる、'libgettextpo'ライブラリーが提供されています。

9.1 msgcatプログラムの呼び出し

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

msgcatプログラムは、指定されたPOファイルを結合・マージするプログラムです。プログラムは、指定された複数のPOファイルの中から、2つ以上のファイルで使用されている共通のメッセージを見つけます。--more-thanオプションを使うと、指定したファイル数より多くのファイルで共通のメッセージを出力するか指定できます。反対に--less-thanオプションでは、指定したファイル数より少ないファイルで共通のメッセージを出力するか指定できます (例 '--less-than=2' と指定すると一意なメッセージだけが出力されます)。翻訳やコメントは累積されますが、--use-firstを指定した場合は、指定されたPOファイルのうちで最初のものを採用します。すべてのPOファイルの位置情報も累積されます。

9.1.1 入力ファイルの位置

'*inputfile* ...'
: 入力ファイルです。

'-f *file*'
'--files-from=*file*'
: 入力ファイルの名前を、コマンドラインからではなく、*file*から読み込みます。

'-D *directory*'
'--directory=*directory*'
: ディレクトリーのリストに *directory*を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

*inputfile*に '-'が指定された場合は、標準入力から読み込みます。

9.1.2 出力ファイルの位置

'-o *file*'
'--output-file=*file*'
: 指定されたファイルに出力を書き込みます。

出力ファイルが指定されていない、または '-'が指定された場合、結果は標準出力に出力されます。

9.1.3 メッセージ選択

'-< *number*'
'--less-than=*number*'
: *number*に指定した数より少ないメッセージを出力します。指定しなかった場合のデフォルトは無限大です。

'-> *number*'
'--more-than=*number*'
: *number*に指定した数より大きいメッセージを出力します。指定しなかった場合のデフォルトは 0 です。

'-u'
'--unique'
: '--less-than=2'の省略指定です。一意なメッセージだけを出力します。

9.1.4 入力ファイルの構文

‘`-P`’
‘`--properties-input`’
入力ファイルが PO ファイルの構文ではなく、Java の`.properties`の構文にのっとった Java ResourceBundle ファイルだとみなします。

‘`--stringtable-input`’
入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の`.strings`の構文にのっとったファイルだとみなします。

9.1.5 出力の詳細

‘`-t`’
‘`--to-code=`*`name`*’
出力のエンコーディングを指定します。

‘`--use-first`’
各メッセージで利用可能な最初のメッセージを使用します。複数の翻訳を 1 つにマージしません。

‘`--lang=`*`catalogname`*’
ヘッダーのエントリーで使用される、‘`Language`’フィールドを指定します。このフィールドの意味については、Section 6.2 [Header Entry], page 43 を参照してください。‘`Language-Team`’と‘`Plural-Forms`’のフィールドは変更されないことに注意してください。

‘`--color`’
‘`--color=`*`when`*’
色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘`--style=`*`style_file`*’
`--color`にたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘`--force-po`’
メッセージが何も含まれていない場合でも、常に出力ファイルに書き込みます。

‘`-i`’
‘`--indent`’
インデントされた形式で.po ファイルを書き込みます。

‘`--no-location`’
‘`#: `*`filename`*`:`*`line`*’という行を書き込みません。

‘`-n`’
‘`--add-location`’
‘`#: `*`filename`*`:`*`line`*’という行を生成します (デフォルト)。

‘`--strict`’
Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘-p’
‘--properties-output’
Java の.propertiesの書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、陳腐化したメッセージを暗黙で除去することに注意してください。

‘--stringtable-output’
.stringsの書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘-w *number*’
‘--width=*number*’
出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘--no-wrap’
長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘-s’
‘--sort-output’
ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

‘-F’
‘--sort-by-file’
ファイルの場所により出力をソートします。

9.1.6 情報的な出力

‘-h’
‘--help’ このヘルプを表示して終了します。

‘-V’
‘--version’
バージョン情報を表示して終了します。

9.2 msgconvプログラムの呼び出し

msgconv [*option*] [*inputfile*]

msgconvは、ある翻訳カタログを別の文字エンコーディングに変換するプログラムです。

9.2.1 入力ファイルの位置

‘*inputfile*’
入力となる PO ファイルです。

‘-D *directory*’
‘--directory=*directory*’
: ディレクトリーのリストに *directory* を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

*inputfile*が指定されていないか、‘-’が指定された場合は、標準入力から読み込みます。

9.2.2 出力ファイルの位置

‘-o *file*’
‘--output-file=*file*’
: 指定されたファイルに出力を書き込みます。

出力ファイルが指定されていない、または ‘-’が指定された場合、結果は標準出力に出力されます。

9.2.3 変換する対象

‘-t’
‘--to-code=*name*’
: 出力のエンコーディングを指定します。

デフォルトのエンコーディングは、現在のロケールのエンコーディングです。

9.2.4 入力ファイルの構文

‘-P’
‘--properties-input’
: 入力ファイルが PO ファイルの構文ではなく、Java の.propertiesの構文にのっとった Java ResourceBundle ファイルだとみなします。

‘--stringtable-input’
: 入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の.stringsの構文にのっとったファイルだとみなします。

9.2.5 出力の詳細

‘--color’
‘--color=*when*’
: 色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘--style=*style_file*’
: --colorにたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘--force-po’
: メッセージが何も含まれていない場合でも、常に出力ファイルに書き込みます。

‘-i’
‘--indent’
: インデントされた形式で.po ファイルを書き込みます。

‘--no-location’
: ‘#: *filename*:*line*’という行を書き込みません。

‘--add-location’
: ‘#: *filename*:*line*’という行を生成します (デフォルト)。

‘--strict’
: Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘-p’
‘--properties-output’
: Java の.propertiesの書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、陳腐化したメッセージを暗黙で除去することに注意してください。

‘--stringtable-output’
: .stringsの書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘-w *number*’
‘--width=*number*’
: 出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘--no-wrap’
: 長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘-s’
‘--sort-output’
: ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

‘-F’
‘--sort-by-file’
: ファイルの場所により出力をソートします。

9.2.6 情報的な出力

‘-h’
‘--help’ このヘルプを表示して終了します。

‘-V’
‘--version’
: バージョン情報を表示して終了します。

9.3 msggrepプログラムの呼び出し

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

msggrepは翻訳カタログから、指定したパターン、指定したソースファイルに属するすべてのメッセージを抽出するプログラムです。

9.3.1 入力ファイルの位置

‘*inputfile*’
入力となる PO ファイルです。

‘`-D` *directory*’
‘`--directory=`*directory*’
ディレクトリーのリストに *directory*を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

*inputfile*が指定されていないか、‘`-`’が指定された場合は、標準入力から読み込みます。

9.3.2 出力ファイルの位置

‘`-o` *file*’
‘`--output-file=`*file*’
指定されたファイルに出力を書き込みます。

出力ファイルが指定されていない、または‘`-`’が指定された場合、結果は標準出力に出力されます。

9.3.3 メッセージ選択

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

以下のような場合、メッセージが選択されます

- 指定したソースファイルのいずれかに属するメッセージの場合
- 指定した domain のいずれかに属するメッセージの場合
- ‘`-J`’が指定されていて、メッセージのコンテキスト (msgctxt) が *msgctxt-pattern*にマッチする場合
- ‘`-K`’が指定されていて、メッセージのキー (msgid または msgid_plural) が *msgid-pattern*にマッチする場合
- ‘`-T`’が指定されていて、翻訳 (msgstr) が *msgstr-pattern*にマッチする場合
- ‘`-C`’が指定されていて、翻訳者のコメントが *comment-pattern*にマッチする場合

1 つ以上の条件を指定した場合には、それぞれの条件に適合するメッセージのがすべて選択されます。

msgctxt-pattern、*msgid-pattern*、*msgstr-pattern*の書式です:

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

*pattern*にはデフォルトでは標準の正規表現 (POSIX Basic Regular Expressions: grep -e と同等) を指定します。拡張された正規表現 (POSIX Extended Regular Expressions: egrep, grep -E と同等) の場合は-E を、固定文字列の場合 (Fixed String search: fgrep, grep -F と同等) には-F を指定してください。

‘`-N` *sourcefile*’
‘`--location=`*sourcefile*’
*sourcefile*から抽出されたメッセージを選択します。*sourcefile*にはファイル名の文字列、またはワイルドカード文字列を指定できます。

‘`-M` *domainname*’
‘`--domain=`*domainname*’
ドメイン *domainname*に属するメッセージを選択します。

‘`-J`’
‘`--msgctxt`’
msgctxt を選択するためのパターンの開始を宣言します。

‘`-K`’
‘`--msgid`’ msgid を選択するためのパターンの開始を宣言します。

‘`-T`’
‘`--msgstr`’
msgstr を選択するためのパターンの開始を宣言します。

‘`-C`’
‘`--comment`’
翻訳者コメントを選択するためのパターンの開始を宣言します。

‘`-X`’
‘`--extracted-comment`’
抽出コメントを選択するためのパターンの開始を宣言します。

‘`-E`’
‘`--extended-regexp`’
*pattern*が、拡張された正規表現であることを指定します。

‘`-F`’
‘`--fixed-strings`’
*pattern*が、改行で区切られた一連の文字列であることを指定します。

‘`-e` *pattern*’
‘`--regexp=`*pattern*’
*pattern*を、正規表現として使用します。

‘`-f` *file*’
‘`--file=`*file*’
*pattern*を、*file*から取得します。

‘`-i`’
‘`--ignore-case`’
大文字と小文字を区別しません。

‘`-v`’
‘`--invert-match`’
条件に一致するメッセージではなく、一致しないメッセージだけを出力します。

9.3.4 入力ファイルの構文

‘`-P`’
‘`--properties-input`’
入力ファイルが PO ファイルの構文ではなく、Java の`.properties`の構文にのっとった Java ResourceBundle ファイルだとみなします。

‘`--stringtable-input`’
入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の`.strings`の構文にのっとったファイルだとみなします。

9.3.5 出力の詳細

‘`--color`’
‘`--color=`*`when`*’
色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘`--style=`*`style_file`*’
`--color`にたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘`--force-po`’
メッセージが何も含まれていない場合でも、常に出力ファイルに書き込みます。

‘`--indent`’
インデントされた形式で.po ファイルを書き込みます。

‘`--no-location`’
‘`#: `*`filename`*`:`*`line`*’という行を書き込みません。

‘`--add-location`’
‘`#: `*`filename`*`:`*`line`*’という行を生成します (デフォルト)。

‘`--strict`’
Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘`-p`’
‘`--properties-output`’
Java の`.properties`の書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、陳腐化したメッセージを暗黙で除去することに注意してください。

‘`--stringtable-output`’
`.strings`の書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘`-w `*`number`*’
‘`--width=`*`number`*’
出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘`--no-wrap`’
長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘--sort-output’
ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

‘--sort-by-file’
ファイルの場所により出力をソートします。

9.3.6 情報的な出力

‘-h’
‘--help’ このヘルプを表示して終了します。

‘-V’
‘--version’
バージョン情報を表示して終了します。

9.3.7 例

gnulib-lib/error.cと gnulib-lib/getopt.cというソースファイルからメッセージを抽出する場合:

```
msggrep -N gnulib-lib/error.c -N gnulib-lib/getopt.c input.po
```

“Please specify” という文字列が含まれるメッセージを抽出する場合:

```
msggrep --msgid -F -e 'Please specify' input.po
```

“Menu>File”、“Menu>Edit” またはそれらのサブメニューであることを指定するコンテキストをもつメッセージを抽出する場合:

```
msggrep --msgctxt -E -e '^Menu>(File|Edit)' input.po
```

翻訳文字列に wordlist.txtというファイル中の文字列を含むメッセージを抽出する場合:

```
msggrep --msgstr -F -f wordlist.txt input.po
```

9.4 msgfilterプログラムの呼び出し

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

msgfilterは、翻訳カタログ内の翻訳にフィルターを適用するためのプログラムです。

それぞれの *filter*呼び出しにおいて、環境変数の MSGFILTER_MSGIDと MSGFILTER_LOCATIONに、メッセージの msgid とメッセージを含む PO ファイルの場所がバインドされます。メッセージがコンテキストを保有する場合は、環境変数 MSGFILTER_MSGCTXTがメッセージのコンテキストにバインドされます (コンテキストを保有していない場合にはバインドされません)。

9.4.1 入力ファイルの位置

‘-i *inputfile*’
‘--input=*inputfile*’
入力となる PO ファイルです。

‘-D *directory*’
‘--directory=*directory*’
ディレクトリーのリストに *directory*を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

*inputfile*が指定されていないか、‘-’が指定された場合は、標準入力から読み込みます。

9.4.2 出力ファイルの位置

‘`-o` *`file`*’
‘`--output-file=`*`file`*’
　　指定されたファイルに出力を書き込みます。

出力ファイルが指定されていない、または‘`-`’が指定された場合、結果は標準出力に出力されます。

9.4.3 フィルター

*filter*は、標準入力から翻訳を読み込み、それに変更を加えて標準出力に書き込むプログラムです。フィルターとして頻繁に使用されるプログラムとしては‘`sed`’があります。その他にも認識できるビルトインフィルターがいくつか存在します。

注意: ビルトインではないフィルターの場合には、エンコーディングに注意する必要があります。*filter*が、入力となる翻訳カタログのエンコードに対処できるようにするのは、あなたの責任となります。*filter*が入力として特定のエンコーディングを期待する場合には、‘`msgfilter`’を呼び出す前に、最初のステップとして‘`msgconv`’で翻訳カタログをそのエンコーディングに変換できます。*filter*が入力として locale のエンコーディングを期待しているけれど、あなたは locale のエンコーディングを無視したいときには、最初に‘`msgconv`’で翻訳カタログを UTF-8 に変換してから、環境変数 `LC_ALL` に UTF-8 locale を指定して、‘`msgfilter`’を使うことができます。

注意: 翻訳カタログ内のほとんどの翻訳は改行で終端されていません。そのため、入力の最終行が改行で終端されていなくても、*filter*がそれを認識すること、そして最終行に余分な改行を付加しないことが重要になります。いくつかのプラットフォームにおいて‘`sed`’が、改行で終端されていない最終行を無視することが知られています。代用として、このような制限を持たない GNU ‘`sed`’を使うことができます。

9.4.4 *filter*が‘`sed`’のときの便利な *filter-option*

‘`-e` *`script`*’
‘`--expression=`*`script`*’
　　実行するコマンドに *script*を追加します。

‘`-f` *`scriptfile`*’
‘`--file=`*`scriptfile`*’
　　実行するコマンドに、*scriptfile*の内容を追加します。

‘`-n`’
‘`--quiet`’
‘`--silent`’
　　パターンの空白の出力を自動的に抑制します。

9.4.5 ビルトインの *filter*

‘`recode-sr-latin`’はビルトインのフィルターとして認識されます。‘`recode-sr-latin`’は、Cyrillic 文字で記述された Serbian のテキストを、Latin 文字に変換するコマンドです。‘`msgfilter recode-sr-latin`’コマンドにより、PO ファイルの翻訳にたいしてこの変換を適用できます。これを使えば `sr.po`ファイルを、`sr@latin.po`ファイルに変換できます。

ビルトインのフィルターは、現在のロケールのエンコーディングとは無関係です。またビルトインのフィルターを使う場合、‘`msgfilter`’はメッセージカタログのエンコーディングを自動的に UTF-8 に変換することができます。

9.4.6 入力ファイルの構文

‘`-P`’
‘`--properties-input`’
入力ファイルが PO ファイルの構文ではなく、Java の`.properties`の構文にのっとった Java ResourceBundle ファイルだとみなします。

‘`--stringtable-input`’
入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の`.strings`の構文にのっとったファイルだとみなします。

9.4.7 出力の詳細

‘`--color`’
‘`--color=`*`when`*’
色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘`--style=`*`style_file`*’
`--color`にたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘`--force-po`’
メッセージが何も含まれていない場合でも、常に出力ファイルに書き込みます。

‘`--indent`’
インデントされた形式で.po ファイルを書き込みます。

‘`--keep-header`’
ヘッダーのエントリーを保持します (例: ‘`msgid ""`’にフィルターを適用しないで未変更にします)。デフォルトでは、ヘッダーのエントリーにたいしても、他のメッセージと同様にフィルタリングの対象になります。

‘`--no-location`’
‘`#: `*`filename`*`:`*`line`*’という行を書き込みません。

‘`--add-location`’
‘`#: `*`filename`*`:`*`line`*’という行を生成します (デフォルト)。

‘`--strict`’
Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘`-p`’
‘`--properties-output`’
Java の`.properties`の書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、陳腐化したメッセージを暗黙で除去することに注意してください。

‘`--stringtable-output`’
`.strings`の書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘`-w` *number*’
‘`--width=`*number*’
出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘`--no-wrap`’
長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘`-s`’
‘`--sort-output`’
ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

‘`-F`’
‘`--sort-by-file`’
ファイルの場所により出力をソートします。

9.4.8 情報的な出力

‘`-h`’
‘`--help`’ このヘルプを表示して終了します。

‘`-V`’
‘`--version`’
バージョン情報を表示して終了します。

9.4.9 例

German の翻訳を、Swiss の正書法に変換する場合 (UTF-8 locale):

```
msgconv -t UTF-8 de.po | msgfilter sed -e 's/ß/ss/g'
```

Cyrillic 文字の Serbian の翻訳を、Latin 文字に変換する場合:

```
msgfilter recode-sr-latin < sr.po
```

9.5 `msguniq`プログラムの呼び出し

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

`msguniq`は、翻訳カタログ内の重複した翻訳を統一するためのプログラムです。このプログラムは、同じメッセージ ID にたいする、重複した翻訳を探します。このような重複したメッセージは、`msgfmt`、`msgmerge`、`msgcat`の入力としては無効です。デフォルトでは重複はマージされます。‘`--repeated`’オプションを指定すると、重複したメッセージだけが出力され、他のすべてのメッセージは破棄されます。コメント、および抽出されたコメントは累積されます。ただし ‘`--use-first`’が指定された場合には、最初の翻訳のものが使用されます。‘`--unique`’オプションを使用すると、重複は破棄されます。

9.5.1 入力ファイルの位置

‘*inputfile*’
入力となる PO ファイルです。

‘-D *directory*’
‘--directory=*directory*’
: ディレクトリーのリストに *directory* を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

*inputfile*が指定されていないか、‘-’が指定された場合は、標準入力から読み込みます。

9.5.2 出力ファイルの位置

‘-o *file*’
‘--output-file=*file*’
: 指定されたファイルに出力を書き込みます。

出力ファイルが指定されていない、または ‘-’が指定された場合、結果は標準出力に出力されます。

9.5.3 メッセージ選択

‘-d’
‘--repeated’
: 重複したメッセージだけを出力します。

‘-u’
‘--unique’
: 一意なメッセージだけを出力します (重複したメッセージは破棄されます)。

9.5.4 入力ファイルの構文

‘-P’
‘--properties-input’
: 入力ファイルが PO ファイルの構文ではなく、Java の.propertiesの構文にのっとった Java ResourceBundle ファイルだとみなします。

‘--stringtable-input’
: 入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の.stringsの構文にのっとったファイルだとみなします。

9.5.5 出力の詳細

‘-t’
‘--to-code=*name*’
: 出力のエンコーディングを指定します。

‘--use-first’
: 各メッセージで利用可能な最初のメッセージを使用します。複数の翻訳を 1 つにマージしません。

‘--color’
‘--color=*when*’
: 色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘--style=*style_file*’
: --colorにたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘`--force-po`’
: メッセージが何も含まれていない場合でも、常に出力ファイルに書き込みます。

‘`-i`’
‘`--indent`’
: インデントされた形式で.po ファイルを書き込みます。

‘`--no-location`’
: ‘`#: filename:line`’という行を書き込みません。

‘`-n`’
‘`--add-location`’
: ‘`#: filename:line`’という行を生成します (デフォルト)。

‘`--strict`’
: Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘`-p`’
‘`--properties-output`’
: Java の`.properties`の書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、知能化したメッセージを暗黙で除去することに注意してください。

‘`--stringtable-output`’
: `.strings`の書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘`-w number`’
‘`--width=number`’
: 出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘`--no-wrap`’
: 長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘`-s`’
‘`--sort-output`’
: ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

‘`-F`’
‘`--sort-by-file`’
: ファイルの場所により出力をソートします。

9.5.6 情報的な出力

‘`-h`’
‘`--help`’ このヘルプを表示して終了します。

‘-V’
‘--version’
バージョン情報を表示して終了します。

9.6 msgcommプログラムの呼び出し

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

msgcommは、指定された 2 つ以上のファイルから、共通のメッセージを探すプログラムです。--more-thanオプションを使用すると、指定された数より多くのファイルで共通のメッセージを出力します。反対に--less-thanオプションを使用すると、指定された数より少ないファイルで共通のメッセージを出力します (例: ‘--less-than=2’は一意なメッセージだけを出力します)。翻訳、コメント、および抽出されたコメントは蓄積されます (最初の PO ファイルのものを使用するように指定した場合を除く)。PO ファイルのファイル位置の情報も蓄積されます。

9.6.1 入力ファイルの位置

‘*inputfile* ...’
入力ファイルです。

‘-f *file*’
‘--files-from=*file*’
入力ファイルの名前を、コマンドラインからではなく、*file*から読み込みます。

‘-D *directory*’
‘--directory=*directory*’
ディレクトリーのリストに *directory*を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

*inputfile*に ‘-’が指定された場合は、標準入力から読み込みます。

9.6.2 出力ファイルの位置

‘-o *file*’
‘--output-file=*file*’
指定されたファイルに出力を書き込みます。

出力ファイルが指定されていない、または ‘-’が指定された場合、結果は標準出力に出力されます。

9.6.3 メッセージ選択

‘-< *number*’
‘--less-than=*number*’
*number*に指定した数より少ないメッセージを出力します。指定しなかった場合のデフォルトは無限大です。

‘-> *number*’
‘--more-than=*number*’
*number*に指定した数より大きいメッセージを出力します。指定しなかった場合のデフォルトは 1 です。

‘-u’
‘--unique’
‘--less-than=2’の省略指定です。一意なメッセージだけを出力します。

9.6.4 入力ファイルの構文

‘-P’
‘--properties-input’
入力ファイルが PO ファイルの構文ではなく、Java の.propertiesの構文にのっとった Java ResourceBundle ファイルだとみなします。

‘--stringtable-input’
入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の.stringsの構文にのっとったファイルだとみなします。

9.6.5 出力の詳細

‘--color’
‘--color=*when*’
色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘--style=*style_file*’
--colorにたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘--force-po’
メッセージが何も含まれていない場合でも、常に出力ファイルに書き込みます。

‘-i’
‘--indent’
インデントされた形式で.po ファイルを書き込みます。

‘--no-location’
‘#: *filename*:*line*’という行を書き込みません。

‘-n’
‘--add-location’
‘#: *filename*:*line*’という行を生成します (デフォルト)。

‘--strict’
Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘-p’
‘--properties-output’
Java の.propertiesの書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、陳腐化したメッセージを暗黙で除去することに注意してください。

‘--stringtable-output’
.stringsの書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘`-w` *number*’
‘`--width=`*number*’
出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘`--no-wrap`’
長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘`-s`’
‘`--sort-output`’
ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

‘`-F`’
‘`--sort-by-file`’
ファイルの場所により出力をソートします。

‘`--omit-header`’
Don’t write header with ‘`msgid ""`’ entry.

9.6.6 情報的な出力

‘`-h`’
‘`--help`’ このヘルプを表示して終了します。

‘`-V`’
‘`--version`’
バージョン情報を表示して終了します。

9.7 `msgcmp`プログラムの呼び出し

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

`msgcmp`プログラムは、Uniforum 形式の 2 つの.po ファイルを比較して、同じ msgid 文字列を含んでいるかチェックするプログラムです。*def*.po ファイルは、翻訳を含んだ既存の PO ファイルです。*ref*.pot は、最後に作成した PO ファイル、または PO Template ファイル (通常 `xgettext`により作成される) です。このプログラムは、プログラム内のメッセージが翻訳されているかチェックするときに便利です。完全に一致するエントリーが見つからない場合は、より良い診断メッセージを生成するために fuzzy マッチングが行われます。

9.7.1 入力ファイルの位置

‘*def*`.po`’ 翻訳です。

‘*ref*`.pot`’ ソースへの参照です。

‘`-D` *directory*’
‘`--directory=`*directory*’
ディレクトリーのリストに *directory*を追加します。このディレクトリーのリストよりソースファイルを検索します。

9.7.2 オペレーションの修飾

‘`-m`’
‘`--multi-domain`’
def.po 内の各ドメインにたいして *ref*.pot を適用します。

‘`-N`’
‘`--no-fuzzy-matching`’
完全に一致するものが見つからない場合、fuzzy マッチングを行いません。これにより処理のスピードが大幅に改善されます。

‘`--use-fuzzy`’
def.po 内の fuzzy メッセージを、翻訳されたメッセージとみなします。fuzzy メッセージは翻訳者により検証されていないので、普通はこのオプションを使うのは正しくないことに注意してください。

‘`--use-untranslated`’
def.po 内の未翻訳のメッセージを、翻訳されたメッセージとみなします。普通はこのオプションを使うのは正しくないことに注意してください。

9.7.3 入力ファイルの構文

‘`-P`’
‘`--properties-input`’
入力ファイルが PO ファイルの構文ではなく、Java の`.properties`の構文にのっとった Java ResourceBundle ファイルだとみなします。

‘`--stringtable-input`’
入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の`.strings`の構文にのっとったファイルだとみなします。

9.7.4 情報的な出力

‘`-h`’
‘`--help`’ このヘルプを表示して終了します。

‘`-V`’
‘`--version`’
バージョン情報を表示して終了します。

9.8 `msgattrib`プログラムの呼び出し

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

`msgattrib`プログラムは、翻訳カタログのメッセージの属性にしたがってフィルターを適用したり、属性を操作するためのプログラムです。

9.8.1 入力ファイルの位置

‘*inputfile*’
入力となる PO ファイルです。

‘-D *directory*’
‘--directory=*directory*’
ディレクトリーのリストに *directory* を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

*inputfile*が指定されていないか、‘-’が指定された場合は、標準入力から読み込みます。

9.8.2 出力ファイルの位置

‘-o *file*’
‘--output-file=*file*’
指定されたファイルに出力を書き込みます。

出力ファイルが指定されていない、または ‘-’が指定された場合、結果は標準出力に出力されます。

9.8.3 メッセージ選択

‘--translated’
翻訳されたメッセージを残して、未翻訳のメッセージは削除します。

‘--untranslated’
未翻訳のメッセージを残して、翻訳済みのメッセージは削除します。

‘--no-fuzzy’
削除: ‘fuzzy’ とマークされたメッセージを削除します。

‘--only-fuzzy’
保持: ‘fuzzy’ とマークされたメッセージを残して、他のすべてのメッセージは削除します。

‘--no-obsolete’
#~のついた廃止されたメッセージを削除します。

‘--only-obsolete’
#~のついた廃止されたメッセージを残して、他のすべてのメッセージは削除します。

9.8.4 属性の操作

メッセージの選択・削除が実行された後に、属性は変更されます。‘--only-file’か‘--ignore-file’オプションを指定すると、*only-file*に記載されたメッセージ、もしくは *ignore-file* に記載されていないメッセージにだけ、変更が適用されます。

‘--set-fuzzy’
すべてのメッセージに、‘fuzzy’ をセットします。

‘--clear-fuzzy’
すべてのメッセージに、非 ‘fuzzy’ をセットします。

‘--set-obsolete’
すべてのメッセージを、陳腐化したメッセージにします。

‘--clear-obsolete’
陳腐化したメッセージを、陳腐化していないメッセージにセットします。

‘--previous’
もし ‘fuzzy’ とマークされているとき、翻訳されたメッセージの “以前の msgid” を残します。

‘--clear-previous’
すべてのメッセージから、“以前の msgid” であることを示すコメントマーク (‘#|’) がついたものを削除します。

‘--only-file=*file*’
*file*に記載されたエントリーの属性だけを変更します。*file*には PO、または POT ファイルを指定します。

‘--ignore-file=*file*’
*file*に記載されていないエントリーの属性だけを変更します。*file*には PO、または POT ファイルを指定します。

‘--fuzzy’ ‘--only-fuzzy --clear-fuzzy’の省略指定です。fuzzy メッセージだけを残すとともに、それらのメッセージの ‘fuzzy’ マークを外します。

‘--obsolete’
‘--only-obsolete --clear-obsolete’の省略指定です。廃止されたメッセージだけを残すとともに、それらのメッセージが陳腐化していることを示すマークを外します。

9.8.5 入力ファイルの構文

‘-P’
‘--properties-input’
入力ファイルが PO ファイルの構文ではなく、Java の.propertiesの構文にのっとった Java ResourceBundle ファイルだとみなします。

‘--stringtable-input’
入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の.stringsの構文にのっとったファイルだとみなします。

9.8.6 出力の詳細

‘--color’
‘--color=*when*’
色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘--style=*style_file*’
--colorにたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘--force-po’
メッセージが何も含まれていない場合でも、常に出力ファイルに書き込みます。

‘-i’
‘--indent’
インデントされた形式で.po ファイルを書き込みます。

‘--no-location’
‘#: *filename*:*line*’という行を書き込みません。

‘`-n`’
‘`--add-location`’
: ‘`#: filename:line`’という行を生成します (デフォルト)。

‘`--strict`’
: Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘`-p`’
‘`--properties-output`’
: Java の`.properties`の書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、陳腐化したメッセージを暗黙で除去することに注意してください。

‘`--stringtable-output`’
: `.strings`の書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘`-w number`’
‘`--width=number`’
: 出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘`--no-wrap`’
: 長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘`-s`’
‘`--sort-output`’
: ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

‘`-F`’
‘`--sort-by-file`’
: ファイルの場所により出力をソートします。

9.8.7 情報的な出力

‘`-h`’
‘`--help`’ このヘルプを表示して終了します。

‘`-V`’
‘`--version`’
: バージョン情報を表示して終了します。

9.9 `msgen`プログラムの呼び出し

```
msgen [option] inputfile
```

msgenは、English の翻訳カタログを作成するプログラムです。最後に作成された English の PO ファイル、または PO Template(xgettext により生成されます) を入力とし、未翻訳エントリーの翻訳に、msgid と同じ文字列を割り当てます。

注意: 'msginit --no-translator --locale=en'でも、同じような処理を行うことができます。異なるのは、msginitはヘッダーエントリーにたいして特別な配慮を払いますが、msgenは異なるという点です。

9.9.1 入力ファイルの位置

'*inputfile*'
: 入力となる PO または POT ファイルです。

'-D *directory*'
'--directory=*directory*'
: ディレクトリーのリストに *directory*を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

*inputfile*に '-'が指定された場合は、標準入力から読み込みます。

9.9.2 出力ファイルの位置

'-o *file*'
'--output-file=*file*'
: 指定されたファイルに出力を書き込みます。

出力ファイルが指定されていない、または '-'が指定された場合、結果は標準出力に出力されます。

9.9.3 入力ファイルの構文

'-P'
'--properties-input'
: 入力ファイルが PO ファイルの構文ではなく、Java の.propertiesの構文にのっとった Java ResourceBundle ファイルだとみなします。

'--stringtable-input'
: 入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の.stringsの構文にのっとったファイルだとみなします。

9.9.4 出力の詳細

'--lang=*catalogname*'
: ヘッダーエントリーで使用される、'Language'フィールドを指定します。このフィールドの意味については、Section 6.2 [Header Entry], page 43 を参照してください。注意: このオプションでは、'Language-Team'と 'Plural-Forms'はセットされません。

'--color'
'--color=*when*'
: 色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

'--style=*style_file*'
: --colorにたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘`--force-po`’
: メッセージが何も含まれていない場合でも、常に出力ファイルに書き込みます。

‘`-i`’
‘`--indent`’
: インデントされた形式で.po ファイルを書き込みます。

‘`--no-location`’
: ‘`#: filename:line`’という行を書き込みません。

‘`--add-location`’
: ‘`#: filename:line`’という行を生成します (デフォルト)。

‘`--strict`’
: Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘`-p`’
‘`--properties-output`’
: Java の`.properties`の書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、陳腐化したメッセージを暗黙で除去することに注意してください。

‘`--stringtable-output`’
: `.strings`の書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘`-w number`’
‘`--width=number`’
: 出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘`--no-wrap`’
: 長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘`-s`’
‘`--sort-output`’
: ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

‘`-F`’
‘`--sort-by-file`’
: ファイルの場所により出力をソートします。

9.9.5 情報的な出力

‘`-h`’
‘`--help`’ このヘルプを表示して終了します。

‘-V’
‘--version’
バージョン情報を表示して終了します。

9.10 msgexecプログラムの呼び出し

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

msgexecは、翻訳カタログ内のすべての翻訳に、コマンドを適用するためのプログラムです。*command*には、標準入力から翻訳を読み込む任意のプログラムを指定できます。呼び出しは、それぞれの翻訳について 1 回行われます。コマンドの出力は、msgexec の出力となります。msgexec自体の戻り値は、すべての呼び出しにおける最大の戻り値となります。

‘0’という、特別なビルトインコマンドを呼び出すと、NULL 終端された翻訳が出力されます。‘msgexec 0’の出力は、‘xargs -0’の入力として適しています。

それぞれの *command*呼び出しにおいて、環境変数の MSGEXEC_MSGIDと MSGEXEC_LOCATION に、メッセージの msgid とメッセージを含む PO ファイルの場所がバインドされます。メッセージがコンテキストを保有する場合は、環境変数 MSGEXEC_MSGCTXTがメッセージのコンテキストにバインドされます (コンテキストを保有していない場合にはバインドされません)。

注意: *command*が翻訳カタログ内の翻訳のエンコーディングに対処できるようにするのは、あなたの責任です。*command*が特定のエンコーディングを期待する場合、‘msgexec’を呼び出す前に、‘msgconv’プログラムで、翻訳カタログをそのエンコーディングに変換するのが最初のステップとなります。*command*が locale のエンコーディングを期待しているが、あなたは locale のエンコーディングを無視したいときには、最初に ‘msgconv’で翻訳カタログを UTF-8 に変換してから、環境変数 LC_ALLを指定して、‘msgexec’が UTF-8 を処理するようにできます。

9.10.1 入力ファイルの位置

‘-i *inputfile*’
‘--input=*inputfile*’
入力となる PO ファイルです。

‘-D *directory*’
‘--directory=*directory*’
ディレクトリーのリストに *directory*を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし.poファイルが出力されるのは、カレントディレクトリーです。

*inputfile*が指定されていないか、‘-’が指定された場合は、標準入力から読み込みます。

9.10.2 入力ファイルの構文

‘-P’
‘--properties-input’
入力ファイルが PO ファイルの構文ではなく、Java の.propertiesの構文にのっとった Java ResourceBundle ファイルだとみなします。

‘--stringtable-input’
入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の.stringsの構文にのっとったファイルだとみなします。

9.10.3 情報的な出力

‘-h’
‘--help’ このヘルプを表示して終了します。

‘-V’
‘--version’
バージョン情報を表示して終了します。

9.11 PO ファイルの一部をハイライトする

翻訳者は通常、PO ファイル中の未翻訳、および fuzzy メッセージを見ることだけに関心を持っています。また、msgid が変更されたことによりメッセージに fuzzy がセットされたときに、以前のメッセージと現在のメッセージの差分を見たいと望みます (長い msgid の中の数語が変更されたときは特に)。そして最後に、PO ファイル内のセクションのメッセージ (コメント、msgid、msgstr など) の違いを強調するのは、いつでも歓迎します。

このような強調表示は、msgcatの ‘--color’と ‘--style’オプションで可能になります。

9.11.1 --colorオプション

‘--color=*when*’オプションは、どのような状況で着色された出力を生成するか指定します。*when* には、以下のうち 1 つを指定できます:

always
yes 着色された出力が生成されます。

never
no 出力は着色されません。

auto
tty 出力デバイスが tty のとき (例 テキスト画面や terminal emulator ウィンドウに直接出力する場合) は、出力に着色します。

html 着色された HTML 出力が生成されます。

‘--color’と ‘--color=yes’は同じです。デフォルトは ‘--color=auto’です。

そのため、コマンドウィンドウで ‘msgcat vi.po’のようなコマンドを実行すると、着色された出力が生成されます。‘msgcat vi.po | less -R’のように、パイプにたいして出力するときは、出力への着色は行われません。このような状況でも常に着色された出力を得るには、‘msgcat --color vi.po | less -R’のように指定します。

‘--color=html’オプションでは、ブラウザーで閲覧可能な出力が生成されます。このオプションは、例えばIndic 言語を表示したいときに有用です。なぜなら、通常はIndic 文字の表示には terminal emulator よりもブラウザーのほうが適しているからです。

--colorオプションにより生成される出力は、有効な PO ファイルではないことに注意してください。出力には terminal 特有のエスケープシーケンスや HTML タグが含まれます。このような PO ファイルをプログラムが読み込むと、文法エラーとなります。‘--color=html’オプションで HTML ファイルを生成する場合をのぞき、通常は--colorオプションで生成した結果をファイルに保存する必要はありません。

9.11.2 環境変数 TERM

環境変数 TERMには、テキストウィンドウの能力に関する識別情報が含まれています。これらの能力について詳細なリストを得るには、‘infocmp’コマンドを使用します (リファレンスは ‘man 5 terminfo’コマンドで参照することができます)。

埋め込みの色指定をもつテキストを生成するとき、msgcatは TERM 変数を参照します。現在のテキストウィンドウは、普通は少なくとも 8 色の表示をサポートします。しかしテキストウィンドウが 16 色、またはそれ以上の色数をサポートするのに、TERM変数には 8 色しかサポートしないように記述されている場合もあります。そのようなときは、TERMに異なる値を設定する価値があります。

`xterm` 多くのケースでは、xtermは 16 色をサポートするように構築されています。88 色、または 256 色をサポートするように構築することもできます (両方はできませんが)。このような場合は、TERMに xterm-16color、xterm-88color、または xterm-256color をセットすることを試みてもよいでしょう。

`rxvt` rxvtが、16 色をサポートするよう構築されている場合があります。このような場合は、TERMに rxvt-16colorをセットすることができます。

`konsole` konsoleも、16 色をサポートするよう構築されている場合があります。このような場合には、TERMに konsole-16color、または xterm-16colorをセットすることができます。

TERMを設定した後は、‘msgcat --color=test’により設定を検証するとともに、適切なカラーマップに見えるか出力を視認できます。

9.11.3 --styleオプション

‘--style=*style_file*’オプションで、着色時に使用するスタイルファイルを指定できます。この指定は--colorが有効なときだけ効果があります。

--styleオプションが指定されていないときは、環境変数 PO_STYLEが使用されます。この環境変数にはユーザーが好む PO ファイル用のスタイルファイルを指定します。

デフォルトのスタイルファイルは、$prefix/share/gettext/styles/po-default.cssです。$prefixはインストールした場所です。

いくつかのスタイルファイルが事前に定義されています:

`po-vim.css`
: このスタイルは vim 7 の表示を模倣します。

`po-emacs-x.css`
: このスタイルは、X11 ウィンドウでの GNU Emacs 21、22 の表示を模倣します。

`po-emacs-xterm.css`
`po-emacs-xterm16.css`
`po-emacs-xterm256.css`
: このスタイルは、‘xterm’(8 色)、‘xterm-16color’(16 色)、‘xterm-256color’(256 色) の端末で GNU Emacs 22 を実行したときの表示を模倣します。

これらのスタイルについてはディレクトリーを指定しなくても使うことができます。これらのスタイルファイルは$prefix/share/gettext/styles/にあります。$prefixはインストールした場所です。

あなた自身でスタイルをデザインできます。これは次のセクションで説明します。

9.11.4 PO ファイルのスタイルルール

端末出力と HTML 出力で、同じ PO ファイル用スタイルを使用できます。PO ファイル用のスタイルは CSS(Cascading Style Sheet) の書式で記述します。CSS の正式な定義については、`http://www.w3.org/TR/css2/cover.html`を参照してください。CSS についての説明を含んだ、HTML 記述のチュートリアルも数多く存在します。

HTML 出力の場合、スタイルファイルは HTML 出力中に埋め込まれます。テキスト出力の場合、スタイルファイルは `msgcat`プログラムにより逐次解釈されます。これは`@import`に関連するファイル名が指定されていて、そのファイル名が以下のような場合、特に意味をもちます:

- HTML 出力のときは関係のある結果 HTML ファイル、
- テキスト出力のときは`@import`を含む、関係のあるスタイルシート (実際にこのようなケースでは、libcroco の制限により@import はまだサポートされていません)。

CSS ルールは selector と declaration により構築されます。declaration にはグラフィカルなプロパティーを指定し、selector にはそれをいつ適用するかを指定します。

PO ファイル用に、以下の簡単な selector がサポートされています ("CSS classes"を基本とします。詳細は CSS2 spec の section 5.8.3 を参照してください)。

- 以下はメッセージ全体に適用される Selector です:

  `.header` PO ファイルのヘッダーエントリーにマッチします。

  `.translated`
  翻訳されたメッセージにマッチします。

  `.untranslated`
  未翻訳のメッセージにマッチします (例: 翻訳が空のメッセージ)。

  `.fuzzy` fuzzy メッセージにマッチします (例: 翻訳者のレビューが必要な翻訳をともなうメッセージ)。

  `.obsolete`
  陳腐化したメッセージにマッチします (例: 現在の POT ファイルでは必要とされない翻訳済みのメッセージ)。

- 以下は、メッセージの PO 文法の一部に適用される Selector です。PO 文法の一般的なメッセージ構造ごとに呼び出されます:

```
white-space
#  translator-comments
#. extracted-comments
#: reference...
#, flag...
#| msgid previous-untranslated-string
msgid untranslated-string
msgstr translated-string
```

  `.comment` すべてのコメントにマッチします (翻訳者コメント、抽出されたコメント、ソースファイルへの参照コメント、フラグコメント、以前のメッセージであることを示すコメント、同様にすべての廃止されたコメント)。

  `.translator-comment`
  翻訳者のコメントにマッチします。

`.extracted-comment`
抽出されたコメントにマッチします (例: 翻訳者への注意を換気するためにプログラマーにより記述されたコメント)。

`.reference-comment`
ソースファイルへの参照コメントにマッチします (行全体)。

`.reference`
ソースファイルへの参照コメント行中の、特定のソースファイルへの参照にマッチします。

`.flag-comment`
フラグコメントにマッチします (行全体)。

`.flag` フラグコメント行の中の、特定のフラグにマッチします。

`.fuzzy-flag`
コメント行中の 'fuzzy' フラグにマッチします。

`.previous-comment`
以前の未翻訳文字列に含まれるコメントにマッチします (行全体)。

`.previous`
区切り文字、結びつけられたキーワード (`msgid`など)、それらの文字列間の空白を含んだ、以前の未翻訳文字列にマッチします。

`.msgid` 区切り文字、結びつけられたキーワード (`msgid`など)、それらの文字列間の空白を含んだ、未翻訳文字列にマッチします。

`.msgstr` 区切り文字、結びつけられたキーワード (`msgstr`など)、それらの文字列間の空白を含んだ、翻訳済みの文字列にマッチします。

`.keyword` キーワード (`msgid`、`msgstr`など) にマッチします。

`.string` 区切り文字 (2 重引用符) を含む文字列にマッチします。

- 以下はメッセージ文字列の一部に適用される Selector です:

`.text` 文字列の内容全体にマッチします (区切り文字は除く例: 2 重引用符)。

`.escape-sequence`
(バックスラッシュで始まる) エスケープシーケンスにマッチします。

`.format-directive`
書式指定文字列にマッチします (多くの言語では '`%`'、`java-format`と`csharp-format`では '`{`'、`lisp-format`と `scheme-format`では '`~`'、`sh-format`では '`$`'で開始されます)。

`.invalid-format-directive`
無効な書式指定文字列にマッチします。

`.added` 未翻訳文字列中で、以前の未翻訳文字列には無かった文字列にマッチします (このリリースではまだ実装されていません)。

`.changed` 未翻訳文字列、または以前の未翻訳文字列中で、変更または置き換えられた文字列にマッチします (このリリースではまだ実装されていません)。

`.removed` 以前の未翻訳文字列中で、現在の未翻訳文字列には無い文字列にマッチします (このリリースではまだ実装されていません)。

これらの selector は、以下の例のように階層的な selector として組み合わせることができます。

```
.msgstr .invalid-format-directive { color: red; }
```

上記の例では、翻訳文字列中の無効な書式指定を強調表示しています。

テキストモードでは、pseudo-classes(CSS2 spec, section 5.11) と pseudo-elements(CSS2 spec, section 5.12) はサポートされません。

HTML モードでの宣言には制限はありません。ブラウザーがサポートする任意の graphical attribute がサポートされます。

テキストモードでの宣言は以下のプロパティーに制限され、他のプロパティーは暗黙に無視されます。

`color` (CSS2 spec, section 14.1)
`background-color` (CSS2 spec, section 14.2.1)
: これらのプロパティーはサポートされます。色数は terminal の能力に適合されます。ほとんどの terminal のサポートは 8 色であることに注意してください。

`font-weight` (CSS2 spec, section 15.2.3)
: このプロパティーはサポートされますが、ほとんどの terminal は `normal`と `bold`の 2 種類の weight しか描画できません。600 以上の値を指定したときは `bold`として描画されます。

`font-style` (CSS2 spec, section 15.2.3)
: このプロパティーはサポートされます。`italic`と `oblique`は、同じ方法で描画されます。

`text-decoration` (CSS2 spec, section 16.3.1)
: このプロパティーはサポートされます。値は `none`と `underline`に制限されます。

9.11.5 PO ファイルを閲覧するために `less`をカスタマイズする

'`less`'は、テキストスクリーンや terminal emulator でテキストファイルを閲覧するための一般的なプログラムです。このプログラムは、色表示や文字飾りのための埋め込みエスケープシーケンスもサポートします。

以下のようにして、PO ファイルの閲覧に `less`を使用できます (UTF-8 環境の場合):

```
msgcat --to-code=UTF-8 --color xyz.po | less -R
```

これと同じことを、次ような簡単なコマンドで行うための方法を説明します:

```
less xyz.po
```

以下の 3 つの準備が必要です:

1. 環境変数 `LESS`に '`-R`'と '`-f`'のオプションを追加します:

   ```
   $ LESS="$LESS -R -f"
   $ export LESS
   ```

2. あなたのシステムに `lessopen.sh`と `lessclose.sh`スクリプトがない場合は、マニュアル ('`man less`') に記載されているように、それらのスクリプトを作成して、環境変数 `LESSOPEN`、`LESSCLOSE`にセットします。
3. 以下のような、ファイルの拡張子から PO ファイルを認識することにより `msgcat`を呼び出し、一時ファイルを生成する断片的なスクリプトを `lessopen.sh`に追加します:

```
case "$1" in
  *.po)
    tmpfile=`mktemp "${TMPDIR-/tmp}/less.XXXXXX"`
    msgcat --to-code=UTF-8 --color "$1" > "$tmpfile"
    echo "$tmpfile"
    exit 0
    ;;
esac
```

9.12 PO ファイルを処理するプログラムを独自に記述する

'msgattrib'や'msgcat'などの組み合わせによる処理では十分でない場合のために、一連の C 関数がライブラリにより提供されています。これを使うことにより、あなたのプログラムから PO ファイルを処理できるようになります。ライブラリーを使う場合は、PO ファイルをパースするルーチンを自分で記述する必要はありません。かわりに PO ファイル内の各メッセージに対応するメモリーへのポインターを取得することができます。現時点では、PO ファイルへ書き込むための関数は提供されていません。

関数はヘッダーファイル'<gettext-po.h>'で宣言されており、'libgettextpo'というライブラリーで定義されています。

`po_file_t` [Data Type]
PO ファイルのコンテンツをメモリーに読み込んだ後に、それらを参照するためのポインター型です。

`po_message_iterator_t` [Data Type]
一連のメッセージを生成する iterator を参照するためのポインター型です。

`po_message_t` [Data Type]
PO ファイルのメッセージ (翻訳を含む) を参照するためのポインター型です。

`po_file_t po_file_read` (*const char *filename*) [Function]
関数 `po_file_read`は、引数としてファイル名を受け取り、その PO ファイルをメモリー内に読み込みます。戻り値は PO ファイル内のコンテンツへのハンドルで、そのハンドルは `po_file_free`が呼び出されるまで有効です。エラーが発生したときの戻り値は `NULL`で、`errno`がセットされます。

`void po_file_free` (*po_file_t file*) [Function]
関数 `po_file_free`は、メモリー内の PO ファイルのコンテンツを解放します。iterator を通じて暗黙にアクセス可能なすべてのメッセージも解放されます。

`const char * const * po_file_domains` (*po_file_t file*) [Function]
関数 `po_file_domains`は、メッセージを所有する PO ファイルの domain を戻します。戻り値は `NULL`で終端された配列で、この配列は *file*のハンドルが有効な間は有効です。'`domain`'指定を持たない PO ファイルの場合は、デフォルトのドメインである`"messages"`という名前のドメインだけが戻されます。

`po_message_iterator_t po_message_iterator` (*po_file_t file*, *const char *domain*) [Function]
`po_message_iterator`は、与えられた *domain*に属する *file*のメッセージを生成する iterator を戻します。*domain*が `NULL`のときは、かわりにデフォルトの domain が使用されます。関数 `po_next_message`を繰り返し呼び出すと、メッセージをリストすることができます。

`void po_message_iterator_free (`*po_message_iterator_t* ***iterator***`)` [Function]
関数 `po_message_iterator_free`は、関数 `po_message_iterator`により割り当てられた iterator を開放します。

`po_message_t po_next_message (`*po_message_iterator_t* ***iterator***`)` [Function]
関数 `po_next_message`は、*iterator*から次のメッセージを戻すとともに iterator を 1 つ進めます。メッセージリストの終端に達すると、`NULL`が戻されます。

以下は `po_message_t`のメンバーを戻す関数です。*file*ハンドルが有効な間は、呼び出しによる結果も有効です。

`const char * po_message_msgid (`*po_message_t* ***message***`)` [Function]
関数 `po_message_msgid`は、メッセージの `msgid`(未翻訳の English 文字列) を戻します。この結果は、非 `NULL`であることが保証されています。

`const char * po_message_msgid_plural (`*po_message_t* ***message***`)` [Function]
関数 `po_message_msgid_plural`は、plural をもつメッセージの `msgid_plural`(未翻訳の English plural 文字列) を戻します。メッセージが plural をもたない場合には、`NULL`が戻されます。

`const char * po_message_msgstr (`*po_message_t* ***message***`)` [Function]
関数 `po_message_msgstr`は、メッセージの `msgstr`(翻訳済み) を戻します。未翻訳のメッセージの場合は、空文字列が戻されます。

`const char * po_message_msgstr_plural (`*po_message_t* ***message***`,` [Function]
int ***index***`)`
関数 `po_message_msgstr_plural`は、plural をもつメッセージの `msgstr[`*index*`]`を戻します。*index*が範囲外のとき、またはメッセージが plural をもたない場合は `NULL`が戻されます。

以下は、これらの関数がどのように使用されるかを示す例です。

```
const char *filename = ...;
po_file_t file = po_file_read (filename);

if (file == NULL)
  error (EXIT_FAILURE, errno, "couldn't open the PO file %s", filename);
{
  const char * const *domains = po_file_domains (file);
  const char * const *domainp;

  for (domainp = domains; *domainp; domainp++)
    {
      const char *domain = *domainp;
      po_message_iterator_t iterator = po_message_iterator (file, domain);

      for (;;)
        {
          po_message_t *message = po_next_message (iterator);

          if (message == NULL)
```

```
            break;
          {
            const char *msgid = po_message_msgid (message);
            const char *msgstr = po_message_msgstr (message);

            ...
          }
        }
      po_message_iterator_free (iterator);
    }
}
po_file_free (file);
```

10 バイナリーの MO ファイルの生成

10.1 msgfmtプログラムの呼び出し

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

msgfmtは、翻訳済みのテキストのメッセージから、バイナリーのメッセージカタログを生成するプログラムです。

10.1.1 入力ファイルの位置

‘*filename*.po ...’
‘-D *directory*’
‘--directory=*directory*’
ディレクトリーのリストに *directory*を追加します。このディレクトリーのリストよりソースファイルを検索します。しかし、結果となるバイナリーファイルが出力されるのは、カレントディレクトリーです。

入力ファイルに ‘-’が指定された場合は、標準入力から読み込みます。

10.1.2 オペレーションモード

‘-j’
‘--java’ Java モード: Java の ResourceBundleクラスを生成します。

‘--java2’ –java と同様ですが Java2(JDK 1.2 以上) とみなします。

‘--csharp’
C#モード: GettextResourceSetのサブクラスを含んだ、.NET の.dll ファイルを生成します。

‘--csharp-resources’
C# resources モード: .NET の.resourcesファイルを生成します。

‘--tcl’ Tcl モード: tcl/msgcat の.msgファイルを生成します。

‘--qt’ Qt モード: Qt の.qmファイルを生成します。

10.1.3 出力ファイルの位置

‘-o *file*’
‘--output-file=*file*’
指定されたファイルに出力を書き込みます。

‘--strict’
プログラムが Uniforum/Sun 実装にしたがうように指定します。これは現時点では、出力ファイルの名前に影響を与えるだけです。オプションにファイル名を指定しなかった場合、出力ファイルは同じ domain 名になります。厳密な Uniforum モードが有効でファイル名が与えられなかった場合には、ファイル名に.moが付加されます。

わたしたちはこの Sun 実装は意味がないと考え、デフォルトではこのモードは選択されません。

出力となる *file*に ‘-’が指定されたときは、出力は標準出力に書き込まれます。

10.1.4 Java モードでの出力ファイルの位置

‘`-r` *`resource`*’
‘`--resource=`*`resource`*’
リソース名を指定します。

‘`-l` *`locale`*’
‘`--locale=`*`locale`*’
locale 名を指定します。*ll*形式による言語指定と、国と言語指定を組み合わせた *ll_CC* のどちらでも指定できます。

‘`-d` *`directory`*’
class のディレクトリー階層のベースとなるディレクトリーを指定します。

クラス名はリソース名の後ろに区切り文字のアンダースコアーと locale 名を付加して決定されます。‘`-d`’オプションは必須です。クラスは指定されたディレクトリーに出力されます。

10.1.5 C#モードでの出力ファイルの位置

‘`-r` *`resource`*’
‘`--resource=`*`resource`*’
リソース名を指定します。

‘`-l` *`locale`*’
‘`--locale=`*`locale`*’
locale 名を指定します。*ll*形式による言語指定と、国と言語指定を組み合わせた *ll_CC* のどちらでも指定できます。

‘`-d` *`directory`*’
locale に依存する`.dll`ファイルを出力するベースディレクトリーを指定します。

\‘`-l`’と ‘`-d`’が必須オプションです。`.dll`ファイルは、locale に依存した名前の指定ディレクトリーのサブディレクトリーに出力されます。

10.1.6 Tcl モードでの出力ファイルの位置

‘`-l` *`locale`*’
‘`--locale=`*`locale`*’
locale 名を指定します。*ll*形式による言語指定と、国と言語指定を組み合わせた *ll_CC* のどちらでも指定できます。

‘`-d` *`directory`*’
メッセージカタログ`.msg`のベースディレクトリーを指定します。

‘`-l`’と ‘`-d`’は必須オプションです。`.msg`は指定されたディレクトリーに出力されます。

10.1.7 入力ファイルの構文

‘`-P`’
‘`--properties-input`’
入力ファイルが PO ファイルの構文ではなく、Java の`.properties`の構文にのっとった Java ResourceBundle ファイルだとみなします。

‘`--stringtable-input`’
入力ファイルが PO ファイルの構文ではなく、NeXTstep/GNUstep の localized resource の`.strings`の構文にのっとったファイルだとみなします。

10.1.8 入力ファイルの解釈

‘-c’
‘--check’ --check-format、--check-header、--check-domainがすべて指定されたとみなしてチェックを行います。

‘--check-format’
language に依存した書式文字列をチェックします。

文字列がprintfのような関数で使用される書式文字列の場合、書式指定子‘%’と、それらに対応する型の変数の個数は一致するはずです。エントリーにたいして#,コメントでc-formatや possible-c-formatフラグが指定されている場合は、チェックが行われます。たとえば、‘%s’が期待される箇所に‘%.*s’や‘%d’が使われていたり、‘%x’が期待される箇所に‘%d’が使われている場合、チェックは診断メッセージを出力します。このチェックは位置パラメーターを処理することさえできるのです。

xgettextプログラムは通常、ある文字列が書式文字列かどうかを、自動的に判定します。しかしこのアルゴリズムも完全ではありません。そのため、printfのような関数で使用されていない文字列を書式文字列とみなしてしまい、エラーが存在しないにもかかわらず msgfmtがエラーを報告する場合があります。

プログラマーが xgettextに判定結果を指示することにより、この問題を解決することができます (Section 15.3.1 [c-format], page 155 を参照してください)。翻訳者は#,行からフラグを削除しようと考える必要はありません。なぜならこの"fix"は、次に msgmergeを呼び出したときに元に戻されてしまうからです。

‘--check-header’
ヘッダーエントリーの存在および内容をチェックします。ヘッダーエントリーのさまざまなフィールドの説明は、Section 6.2 [Header Entry], page 43 を参照してください。

‘--check-domain’
domain 指定と--output-fileオプションの競合をチェックします。

‘-C’
‘--check-compatibility’
GNU msgfmt が X/Open msgfmt のように振る舞うかをチェックします。GNU 拡張を使用しているとエラーになります。

‘--check-accelerators[=*char*]’
メニューアイテムにたいしてキーボードアクセラレーターの存在をチェックします。このチェックは、いくつかの GUI においてメニューアイテム文字列内のキーボードアクセラレーターが、‘&’のすぐ後ろに続く文字としてデザインされていることにもとづきます。キーボードアクセラレーターが"keyboard mnemonic"と呼ばれることもあります。このチェックは、未翻訳文字列に‘&’が 1 つあるとき、翻訳文字列にも 1 つの‘&’があるかをチェックします。このオプションの引数に *char*が与えらる場合、*char*には非英数文字を指定します。指定した文字は‘&’のかわりに、キーボードアクセラレーターのマークとして使用されます。

‘-f’
‘--use-fuzzy’
出力に fuzzy エントリーを使用します。これらの fuzzy メッセージは人間の翻訳者により検証されたものではないため、このオプションの使用は通常は正しくないことに注意してください。

10.1.9 出力の詳細

‘`-a` *number*’
‘`--alignment=`*number*’
文字列を *number*バイトに揃えます (デフォルトは 1)。

‘`--endianness=`*byteorder*’
32 ビットの数字を与えられたバイト順で書き出します。`big`と `little`を指定できます。デフォルトはプラットフォーム、正確に言うと CPU のインディアンに依存します。

任意のインディアンをもつ MO ファイルは、任意のプラットフォームで使用できます。MO ファイルのインディアンがプラットフォームのものでない場合、32 ビットの数値は実行時に交換されます。パフォーマンスに与える影響は無視できるものです。

このオプションは、プラットフォームに束縛されず MO を作成するために便利です。

‘`--no-hash`’
バイナリーファイルにハッシュテーブルを含めないようにします。(ハッシュテーブルを参照するかわりに、バイナリーサーチが行われるため) 実行時の検索が、より高価な処理となります。

10.1.10 情報的な出力

‘`-h`’
‘`--help`’ このヘルプを表示して終了します。

‘`-V`’
‘`--version`’
バージョン情報を表示して終了します。

‘`--statistics`’
翻訳の統計情報を表示します。`--statistics`とともに`--verbose`オプションが指定された場合は、統計行の前に入力ファイルの名前が出力されます。

‘`-v`’
‘`--verbose`’
診断レベルを上げます。

10.2 `msgunfmt`プログラムの呼び出し

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

`msgunfmt`は、バイナリーのメッセージカタログを、Uniforum 形式の.po ファイルに変換するプログラムです。

10.2.1 オペレーションモード

‘`-j`’
‘`--java`’ Java モード: Java の `ResourceBundle` class が入力となります。

‘`--csharp`’
C#モード: `GettextResourceSet`のサブクラスを含む.NET の.dll ファイルが入力となります。

‘`--csharp-resources`’
C# resources モード: .NET の`.resources`ファイルが入力となります。

‘`--tcl`’ Tcl モード: tcl/msgcat の`.msg`ファイルが入力となります。

10.2.2 入力ファイルの位置

‘*file* ...’ 入力となる.mo ファイルです。

*file*が指定されていないか、‘`-`’が指定された場合は、標準入力から読み込みます。

10.2.3 Java モードでの入力ファイルの位置

‘`-r` *resource*’
‘`--resource=`*resource*’
リソース名を指定します。

‘`-l` *locale*’
‘`--locale=`*locale*’
locale 名を指定します。*ll*形式による言語指定と、国と言語指定を組み合わせた *ll_CC* のどちらでも指定できます。

class 名は、resource 名の後ろにアンダースコアーをつけて、その後ろに locale 名を付加することにより決定されます。class は、`CLASSPATH`によって配置されます。

10.2.4 C#モードでの入力ファイルの位置

‘`-r` *resource*’
‘`--resource=`*resource*’
リソース名を指定します。

‘`-l` *locale*’
‘`--locale=`*locale*’
locale 名を指定します。*ll*形式による言語指定と、国と言語指定を組み合わせた *ll_CC* のどちらでも指定できます。

‘`-d` *directory*’
locale に依存する`.dll`ファイルを出力するベースディレクトリーを指定します。

‘`-l`’と‘`-d`’が必須オプションです。`.msg`ファイルは、locale に依存した名前の指定ディレクトリーのサブディレクトリーに配置されています。

10.2.5 Tcl モードでの入力ファイルの位置

‘`-l` *locale*’
‘`--locale=`*locale*’
locale 名を指定します。*ll*形式による言語指定と、国と言語指定を組み合わせた *ll_CC* のどちらでも指定できます。

‘`-d` *directory*’
メッセージカタログ`.msg`のベースディレクトリーを指定します。

‘`-l`’と‘`-d`’は必須オプションです。`.msg`は指定されたディレクトリーに配置されています。

10.2.6 出力ファイルの位置

‘`-o` *file*’
‘`--output-file=`*file*’
指定されたファイルに出力を書き込みます。

出力ファイルが指定されていない、または ‘`-`’が指定された場合、結果は標準出力に出力されます。

10.2.7 出力の詳細

‘`--color`’
‘`--color=`*when*’
色や色以外のテキスト属性を使うか、いつ使うかを指定します。詳細は Section 9.11.1 [The –color option], page 92 を参照してください。

‘`--style=`*style_file*’
`--color`にたいして CSS style rule ファイルを使うかを指定します。詳細は Section 9.11.3 [The –style option], page 93 を参照してください。

‘`--force-po`’
メッセージが何も含まれていない場合でも、常に出力ファイルに書き込みます。

‘`-i`’
‘`--indent`’
インデントされた形式で.po ファイルを書き込みます。

‘`--strict`’
Uniforum に厳密に準拠した PO ファイルを出力します。この Uniforum 形式は GNU の拡張をサポートしないため避けたほうがよいでしょう。

‘`-p`’
‘`--properties-output`’
Java の`.properties`の書式で、Java ResourceBundle を出力します。このファイル形式は plural form をサポートせず、陳腐化したメッセージを暗黙で除去することに注意してください。

‘`--stringtable-output`’
`.strings`の書式で、NeXTstep/GNUstep のローカライズされたリソースファイルを出力します。このファイル形式は plural form をサポートしないことに注意してください。

‘`-w` *number*’
‘`--width=`*number*’
出力ページの幅をセットします。これにより出力ファイル中の長い文字列が指定した幅 (例:スクリーンの列数) に収まるように、各行の長さが *number*以下のような複数の行に分割されます。

‘`--no-wrap`’
長いメッセージ行を分割しません。出力ページの幅を超えるようなメッセージ行も、複数行に分割されません。出力ページの幅を超えるファイル参照行だけが分割されます。

‘-s’
‘--sort-output’
ソートされた出力を生成します。このオプションを使用することにより翻訳者が、メッセージがどのようなコンテキストで使用されるかを理解するのが、困難になることに注意してください。

10.2.8 情報的な出力

‘-h’
‘--help’ このヘルプを表示して終了します。

‘-V’
‘--version’
バージョン情報を表示して終了します。

‘-v’
‘--verbose’
診断レベルを上げます。

10.3 GNU MO ファイルのフォーマット

生成された MO ファイルの書式については、以下のような図を用いて説明するのがよいでしょう。

最初の 2word には、ファイルの識別が含まれます。magic number は常に GNU MO ファイルを意味する number になります。number は生成されたマシンのバイトオーダーにしたがって格納されます。つまり実際の magic number は、0x950412deと 0xde120495のいずれかです。

2 番目の word は、ファイル書式の現在の revision を説明し、major revision number、および minor revision number から成り立ちます。revision number により、MO ファイルの読み手は、古い書式と新しい書式を識別して、(可能なかぎり) 内容を処理できます。いまのところ major revision は 0 か 1 で、minor revision も 0 と 1 ですが、将来は追加されるかもしれません。想定外の major revision number の場合、プログラムは MO ファイル全体を読み込むのを中止する必要があります。想定外の minor revision number は、ファイルは読み込めても、すべての内容は読み込めないことを意味します。プログラムが解析できるのは、より小さな minor revision number のときだけです。

異なる magic number によって書式の違いを表すのではなく、magic number とは別に version が保持されます。これは/etc/magicが滅多に更新されないことが主な理由です。

MO ファイルの冒頭部の情報が拡張されたときに、それらを読み込むプログラムをリコンパイルしなくても良いように、以降のテーブルのポインターを使ってください。そのようにしておけば、後で新しいフラグのビットを追加したり、使用されている文字コードの表示や、新しいテーブルなどを挿入されたときに便利です。

図中の offset Oと offset Tには、2 つの string descriptor を見出すことができます。この 2 つのテーブルでは、どちらも string descriptor として 32 ビットの整数が使用されており、1 つは文字列の長さを示し、もう 1 つは文字列が MO ファイルの先頭から何バイト目かという offset を示します。最初のテーブルには元の文字列の descriptor が含まれていて、これらの元文字列は辞書順にソートされて格納されています。2 番目のテーブルには翻訳された文字列の descriptor が含まれており、これらは 1 番目のテーブルに対応しています。つまり 1 番目のテーブルと同じ添字で 2 番目のテーブルにアクセスすれば、対応する翻訳を取得できます。

元の文字列をソートして格納することにより、MO ファイルにハッシュテーブルを含まれていなかったり、含まれていたとしても実際に使うことができないときにも、単純な二分探索が可能になります。これには他にも利点があります。GNU gettextは、PO ファイルの空の文字列にたいする翻

訳文字列として、MO ファイルに付加するシステム情報を割り当てます。この空文字列と翻訳のペアが、元の文字列のテーブルと、翻訳文字列のテーブルの最初に配置されることにより、システム情報を簡単に見つけることができるのです。

ハッシュテーブルのサイズ Sが 0 のときもあります。これは、ハッシュテーブル自体が MO ファイルに含まれていない場合です。事前に算出されたハッシュテーブルはディスク容量を消費し、速度も早くないという理由で、この方式を好む人もいます。ハッシュテーブルは、MO ファイル中の文字列のソートされた配列の添字を含んでいます。競合は double hashing により解決しています。使用されている正確な hashing algorithm は、GNU `gettext`のコード実装の説明になってしまうので、ここでは説明しません。

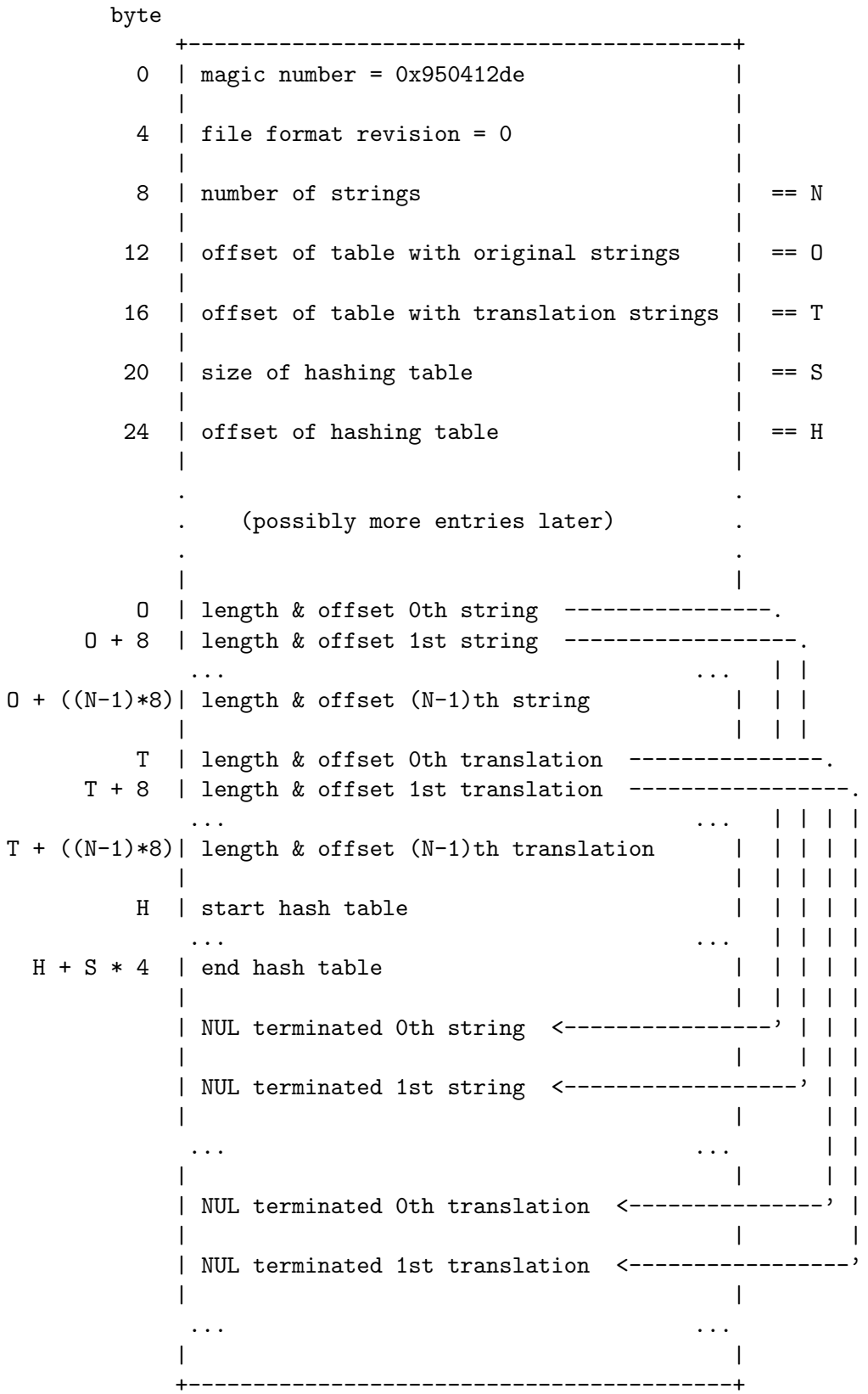
ハッシュテーブルを参照して取得する文字列自体は `NUL`終端されており、string descriptor の文字列長にその `NUL`の分は含まれません。`msgfmt`プログラムには、MO ファイル中の文字列のインデントを選択するオプションがあります。このオプションを指定すると、個々の文字列の開始位置のオフセットは指定されたインデント値の倍数分ずれます。RISC マシンには、適したインデント指定によって速度が改善するものがあります。

context については、元の文字列の代わりに、context 文字列と元の文字列を `EOT`で連結したものが、ソートされて格納されます。

plural form については、元の文字列の singular と plural が `NUL`で区切られて格納されます。string descriptor には、両方の長さが記述されます。しかし、ハッシュテーブルを参照するときは、元の文字列の singular だけが使用されます。さまざまな plural にたいする翻訳は、すべて `NUL`区切りで格納されます。この場合も string descriptor には、それらすべての長さが格納されます。

MO ファイル内の文字列に `NUL`が埋め込まれることを防ぐ方法はありません。しかし現在のプログラムのインターフェースは、文字列が `NUL`で終端されると仮定しているため、文字列の途中に `NUL`が埋め込まれている場合、何らかの不都合が起こり得ます。MO ファイルの書式は、後から他のインターフェースを適用できるほどには一般的です。一例としては、意図しない `NUL`が出現するような箇所に wide character を使用する方法などがあります (実際には MO ファイル中に wide character を保持することはしません。wide character を使用するとファイルの容量が不必要に大きくなります。また '`wchar_t`'はプラットフォームに依存するため、MO ファイルもプラットフォームに依存することになるからです)。

この技術的な問題は、GNU `gettext`の development forum で盛んに議論されており、MO ファイルの書式が将来、進化・変更されることが予想されます。その可能性としては、同時に複数の書式にたいするサポートさえもが含まれます。しかし、わたしたちに何らかの出発点が必要なことは確かで、ここで説明している MO ファイルの書式はよい出発点でした。今の書式には厳密な制約もなく、後から書式を拡張するのは簡単なので、わたしたちは現在のアプローチに満足しています。


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

11 プログラマーの視点

GNU gettextにより提供される現在のメッセージカタログの実装は、インストーラーにより選択された場合にはシステムによるメッセージカタログ処理を使用することを目標にしています。そのため、まず最初に既知の解決策を概観しておく必要があるかもしれません。POSIX コミティーの人たちは、以下で説明する準公式な標準候補のうちから 1 つを採択して管理しませんでした。実際、彼らは何も採択せず、インターフェースの例を含めるだけに決めました。Unix のメジャーなベンダーによるのインターフェースの採用は、X/Open の catgets と、Uniforum の gettext に二分されました。以下ではそれらのインターフェースについて説明するとともに、このジレンマにたいするわたしたちの解決策を説明します。

11.1 catgetsについて

catgetsの実装は、X/Open Portability Guide, Volume 3, XSI Supplementary Definitions, Chapter 5 に定義されています。しかし、この標準を作成する過程は、いくつかの Unix ベンダーからは遅すぎると見なされ、それゆえ彼らは標準を先取りしたバージョンを実装しました。プラットフォームに依存したプログラムを記述したことから、これは問題を引き起こしました (catgetsが一意なインターフェースを保証しないことさえありました)。

インターフェースの決定にコメントするのがコミティー仲間だけに制限されていて、彼らだけがインターフェースを作成できました。彼らは、このインターフェースを本当にプログラムに使わせようと考えてはいませんでした。メモリー保存の実装手法により処理は高速だったので、ユーザーはハッピーでしたが、プログラマーはそれを嫌っていました (少なくともわたしと、他の何人かも...)。

Unix(tm) への正しい移植にともなうすべてのトラブルの原因は、結局のところ彼らがこの仕様を発行した人たちと同じ、X/Open の人たちだったことです。これはわたしに、すべての Unix(という名前を名乗ることを許された) 実装にたいして、このインターフェースが Unix 標準となる未来を予想させるのです (例: Spec1170)。

11.1.1 インターフェース

catgetsの実装にたいするインターフェースには、ファイルのアクセスのための 3 つの関数: ファイルを開く catopen、メッセージテーブルにアクセスする catgets、そして処理が終わった後にファイルを閉じるための catcloseが含まれます。関数のプロトタイプと、必要となる定義は、<nl_types.h> というヘッダーファイルにあります。

catopenは、以下のように呼び出されます:

```
nl_catd catd = catopen ("catalog_name", 0);
```

関数の引数としてカタログ名を指定します。これは通常、プログラムかパッケージを参照する名前を指定します。2 番目のパラメーターは、標準仕様では規定されていません。わたしには、それがさまざまなシステムで一貫した形で実装されているかすら、わかりません。あたりさわりのないアドバイスとしては、値として 0を指定するのがよいでしょう。戻り値はメッセージカタログのハンドルで、このハンドルは openで戻されるファイルのハンドルと同じです。

このハンドルは、以下のようにして catgets関数で使うことができます:

```
char *translation = catgets (catd, set_no, msg_id, "original string");
```

最初のパラメーターは catalog descriptor です。2 番目のパラメーターには msg_idに保持されるメッセージの、セット番号を指定します。つまり catgetsは、以下のような 3 段階のアドレッシングを行います:

catalog name ⇒ set number ⇒ message ID ⇒ translation

4 番目の引数は、翻訳された文字列のアドレスを指すためには使用しません。これはアドレッシングステージが失敗したとき、デフォルト値を与えるためのものです。ここで重要なのは、catgets の戻り値の型が `char *`だとしても、結果の文字列を変更してはならないということです。本来、この戻り値の型は `const char *`のほうがよいのですが、この標準は ANSI C 標準が発行される 1 年前の、1988 年に発行されたものなのです。

最後の関数は期待されたとおりに使用され、そのとおりに振る舞います:

```
catclose (catd);
```

この関数を呼び出すと、その descriptor を `catgets`の呼び出しには使用できません。

11.1.2 `catgets`インターフェースに関する問題点?!

これを説明するのはとても簡単に見えます — わたしたちが話してきたインターフェースのどこに問題があったのでしょうか? 実際のところ、分別のある使い方をするかぎり、そのインターフェースを使うことはできます。しかしメッセージカタログを構築するのは苦痛をともないます。その理由は `catgets`の 3 番目の引数となる、一意な message ID です。これには、すべてのメッセージのペアごとに、数字を割り当てなければなりません。ソースコードの変更、たとえばメッセージを追加したり削除するときに、このリストを保守する際に発生する問題を想像してみてください。もちろん、この混沌を組織化するためのツールもたくさん開発されました。しかし、あるツールで処理できるのに、他のツールでは読み込めないといった様相でした。より簡単に管理できて問題もない、他のやり方はあるのですが、述べないでおきましょう。

11.2 `gettext`について

`gettext`のインターフェースの定義は、Uniforum の提案によるものです。これは Sun から提出されたもので、Sun は 1990 頃に、SunOS4 で `gettext`を実装済みでした。現在では、`gettext`のインターフェースは、OpenI18N 標準により規定されています。

この解決策の主要な点は、通常のファイル処理の手順 (open-use-close) を踏襲しないことと、プログラマーに負担 (特に一意なキーの取扱いにたいして) をかけないことにあります。もちろん一意なキーは必要なので、(メッセージの長短に関わらず) メッセージ自身をキーとします。比較に関する 2 つの方法の詳細については、Section 11.3 [Comparison], page 121 を参照してください。

以下のセクションでは、より詳細にインターフェースを説明します。インターフェースについて詳細を掘り下げて説明するのは、それが GNU `gettext`ライブラリーに密接に関係しているからです。ライブラリーの使い方に興味をもつプログラマーは、この説明にも興味をもつことでしょう。

11.2.1 インターフェース

インターフェースは最小限、a) 文字列が由来するドメインの選択 (すべてのプログラムが 1 つのドメインを使用するのは、構築と保守が難しいので、たとえ可能であったとしても合理的ではありません)、b) 選択されたドメインの文字列へのアクセス、の機能をもたなければなりません。

これは主に `gettext`のインターフェースについての説明です。このインターフェースは、使用するドメインを与えなかった場合に、無条件に参照されるグローバルドメインをもっています。もちろん、このドメインはユーザーが選択することができます。

```
char *textdomain (const char *domain_name);
```

これにより、`LC_MESSAGE`カテゴリーにおける現在のグローバルドメインの状態を問い合わせたり変更することができます。引数はヌル終端された文字列で、ファイル名として使用できる文字でなければなりません。引数 *domain_name*が `NULL`の場合、この関数は現在の値を戻します。値がセットさ

れていなければ、デフォルトのドメイン名 *messages* が戻されます。textdomainの戻り値型は char *となっていますが、それを変更することはできないことに注意してください。それと、ドメイン名が利用可能かのチェックは行われないことを知ることも重要です。ドメイン名が利用可能でない場合、それは翻訳が提供されていないという事実をあらわします。

textdomainでセットしたドメインは、以下の関数で使用されます

```
char *gettext (const char *msgid);
```

この関数は想像されるとおりの簡単な方法で使用されます。これにより、現在のドメインが利用可能な場合には、*msgid*にたいする翻訳文字列が戻されます。ドメインが利用可能でない場合には、引数自身が戻されます。引数に NULLが指定された場合の戻り値は未定義です。

1 つ念頭においてもらいたいのは、使用するドメインを明示的に与えられなかった場合のことです。この場合には現在のドメインが使用されます。プログラム中で同じ gettextを呼び出したとしても、実行の間にドメインが変更された場合には、異なるメッセージカタログが参照されることになります。

もっとも簡単なのは、国際化されたパッケージで普通に使うケースで、この場合は実行開始時に呼び出される textdomainにより、一意なドメイン名 (通常はパッケージ名) がセットされます。以降のコードでは、翻訳が必要な文字列はすべて gettext 関数により処理されます。これだけでパッケージがあなたの言語で話し出すのです。

11.2.2 あいまいざの解決

ほとんどのアプリケーションでは、単一のドメイン名でうまく動作するかもしれませんが、複数のドメインから翻訳を取得する必要があるアプリケーションも存在します。textdomainを呼び出すことにより異なるドメインに切り替えることもできますが、これは不便だし低速です。起こり得る状況としては、この文書を記述しているときに提出された議論 (一般的に使われる関数のすべてのエラーメッセージは、error というドメインに分離するべきである) のケースがあります。これには翻訳が 1 度で済むという意味があります。他のケースとしてはライブラリーのメッセージの場合で、これらがアプリケーションの現在のドメインからは独立している必要があります。

これらの理由により、文字列を取得するために、さらに 2 つの関数が用意されています:

```
char *dgettext (const char *domain_name, const char *msgid);
char *dcgettext (const char *domain_name, const char *msgid,
                 int category);
```

これらの関数は、どちらも 1 番目に追加の引数があります。これには textdomainと同じ引数を与えます。dcgettextの 3 番目の引数により、LC_MESSAGES以外の、他の locale category を使うことができます (実際のところ、わたしにはこれがどんなときに便利なのか、わかりませんが)。*domain_name*が NULLのとき、または *category*に未知のものが指定された場合、結果は未定義となります。Solaris の関数ファミリーの 2 番目の実装にはでは、1 つは含まれているのに、この関数は含まれていないことも触れておくべきでしょう

多重定義が発生する 2 番目の理由は、複数のドメインが同じ名前を共有するかもしれないという事実に起因します。これは必要なメッセージカタログがどこにあるか指定することで解決します。

```
char *bindtextdomain (const char *domain_name,
                      const char *dir_name);
```

この関数を呼び出すことにより、指定されたドメインとディレクトリーのファイルがバインドされます (ファイルがどのように決定されるかは以下で説明します)。特に、システムの既定の位置にあるファイルのかわりに、指定された位置のファイルを使って textdomainを呼び出したいときに使用します。*dir_name*パラメーターに NULLポインターを与えると、*domain_name*にバインドされている値が戻されます。*domain_name*自身が NULLの場合は何も行われず、NULLポインターが戻されます。他の関数と同様に、戻された値を変更することはできません!

*dir_name*に相対パスを指定することは、トラブルの原因になることを覚えておくことは重要です。プログラムが `chdir`コマンドを呼び出すことにより、カレントディレクトリーに関連づいた相対パスは、再計算されます。相対パスの使用により、常に非依存性と信頼性が無効にされます。

11.2.3 メッセージカタログファイルの配置

さまざまなパッケージごとに、多くの異なる言語を保存する必要があるという理由により、わたしたちにはこれらの情報をメッセージカタログファイルに記録するための、何らかの方法が必要です。Unix 環境でよく使われるのは、ファイル名にエンコード名をもたせる方法で、ここでも同じ方法を用います。ディレクトリーは、`bindtextdomain`の 2 番目の引数に指定するディレクトリー (または既定のディレクトリー) の後ろに、locale 名と locale category、それと domain 名を連結したものになります:

dir_name`/`*locale*`/LC_`*category*`/`*domain_name*`.mo`

*dir_name*の既定値はシステムにより定義されます。この習慣を順守する GNU のライブラリーやパッケージのために、以下のように定義されています:

`/usr/local/share/locale`

*locale*は、`LC_`*category*で指定された locale category の名前です。`gettext`と `dgettext`の場合、`LC_`*category*は常に `LC_MESSAGES`になります [1]。locale category の名前は、`setlocale (LC_`*category*`, NULL)`を通じて決定されます [2]。`dcgettext`関数では、3 番目の引数に locale category を指定できます。

11.2.4 `gettext`が使用する出力文字セットの指定方法

`gettext`は、メッセージカタログ内の翻訳を取得するだけでなく、オンザフライで翻訳出力の文字セットを変換することもできます。これは、翻訳者がメッセージカタログを作ったときとは異なる文字セットを使っているユーザーにとって便利です。これにより、文字セットだけが異なるメッセージカタログをいくつも作らなくてよくなるからです。

出力される文字セットのデフォルトは `nl_langinfo (CODESET)`です。これは現在の locale の `LC_CTYPE`の部分に依存します。文字列を locale とは関係のない文字セット (例: UTF-8) で保存するプログラムは、`gettext`や、それに関連するプログラムにたいして、そのエンコードで翻訳を戻すように要求することができます。これは、`bind_textdomain_codeset`関数により行います。

`gettext`の引数である *msgid*は、文字セットの変換の対象外であることに注意してください。`gettext`が、*msgid*に対応する翻訳を見つけられなかったときは、現在の出力の文字セットとは関係なく、元の *msgid*が変更されずに出力されます。すべての *msgid*に US-ASCII 文字列が推奨されているのは、これが理由です。

`char * bind_textdomain_codeset (`*const char *domainname*, *const char *codeset*`)` [Function]

関数`bind_textdomain_codeset`は、ドメイン *domainname*用のメッセージカタログの出力文字セットを指定するのに使用します。引数 *codeset*には、関数`iconv_open`で使用できる有効なコードセット名、または NULL ポインターでなければなりません。

---

[1] `LC_MESSAGES`のないシステムも存在します (例: mingw)。そのような場合には、任意の値として 1729(2 つの立方数の和として、2 通りの方法で表すことのできる最小の正の整数。訳注:ハーディ＝ラマヌジャン数と呼ぶそうです) を使用します。

[2] `setlocale`がサポートされないシステムでの locale の値のセットは、環境変数を参照するのと同じ方法によりシミュレートされます。

パラメーター *codeset*が NULL ポインターの場合、`bind_textdomain_codeset`は、ドメイン *domainname*で現在選択されているコードセットを戻します。コードセットが選択されていないときは、`NULL`が戻されます。

`bind_textdomain_codeset`関数を複数回呼び出すこともあるでしょう。引数 *domainname* を変更せずに複数回呼び出したときは、それ以前に呼び出したときのセッティングによりオーバーライドされます。

`bind_textdomain_codeset`関数は、選択されたコードセットを結合した文字列へのポインターを戻します。その文字列は関数内部で割り当てられ、ユーザーは変更できません。`bind_textdomain_codeset`の実行中にシステムが割り当てに失敗すると、戻り値は `NULL`となり、グローバル変数 *errno*に対応するエラーがセットされます。

11.2.5 あいまいさの解決のためにコンテキストを使用する

グラフィカルユーザーインターフェース (GUI) をもつプログラムで、`gettext`関数を普通に使うと、大きな問題がおきるかもしれない箇所があります。その問題とは、翻訳する必要がある文字列のほとんどが、短い文字列であるような場所で発生します。そらの文字列とは、プルダウンメニューの文字列のように、長さを制限する必要のある文字列です。それらの文字列は、センテンス全体を含んでいなかったり、センテンスの断片がプログラムの異なるシチュエーションで出現し、シチュエーションごとに異なる翻訳を割り当てる必要のあるものです。特に GUI プログラムで頻繁に使用される、1 単語の文字列が問題になります。

`gettext`のアプローチには問題があるので、このような問題が存在しない `catgets`を使う必要があるという人もたくさんいます。しかし、この種の問題を処理するための簡単で強力な方法が、`gettext` 関数には備わっているのです。

それは翻訳するべき文字列に、context を追加する方法です。context にもとづく翻訳参照とは、与えられた文字列にたいする翻訳を検索するときに与えられた context に検索範囲を限定することです。同じ文字列でも、異なる context に属する場合、異なる翻訳を割り当てることができます。ある文字列にたいする context ごとの翻訳は、1 つの MO ファイルに一緒に保存でき、翻訳者も 1 つの PO ファイルを編集するだけです。

`gettext.h`には、context 付きの文字列を参照するためのマクロが含まれます。これらは `<libintl.h>`由来の軽量マクロ、またはインライン関数により実装されています。

```
const char *pgettext (const char *msgctxt, const char *msgid);
```

このマクロの呼び出しでは、*msgctxt*と *msgid*を文字列リテラルにしなければなりません。マクロは、*msgctxt*に与えられた context の、*msgid*に対応する翻訳を戻します。

*msgctxt*は、PO ファイル中で翻訳者が目にすることのできる文字列です。あなたは何らかの方法により標準的なものを定めることと、それを決して変更しないことが必要です。なぜなら *msgctxt*を変更する度に、翻訳者は *msgid*にたいする翻訳をレビューする必要があるからです。

時間を経過しても変更されないような、標準的な *msgctxt*文字列を見つけるのは困難です。しかし `pgettext`の呼び出しに、ファイル名やクラス名を使うべきではありません– なぜならファイルやクラスの名前を変更するのは開発タスクでは一般的なので、それが翻訳者の作業に影響を及ぼすべきでないからです。また、*msgctxt*に完全な英語センテンスのコメント形式を使うべきでもありません– なぜなら、そのようなセンテンスに適用される正書法や文法はしばしば変更されるので、繰り返しになりますが、その変更により翻訳者がレビューを強いられるべきではありません。

‘`pgettext`’の ‘`p`’は、“particular(特定の)” から由来しています: `pgettext`は、特定の *msgid*から翻訳を取得します。

```
const char *dpgettext (const char *domain_name,
```

```
                          const char *msgctxt, const char *msgid);
const char *dcpgettext (const char *domain_name,
                        const char *msgctxt, const char *msgid,
                        int category);
```

これらの関数は pgettextを、より一般化したものです。それぞれの関数は、dgettextやdcgettextと同様に振る舞います。引数 *domain_name*には、翻訳のドメインを定義します。引数*category*を指定することにより、LC_MESSAGESとは異なる locale category を指定できます。

次のような例で考えてみましょう。メニューバーをもつ GUI プログラムがあり、メニューには以下のようなエントリーがあるとします:

```
+------------+------------+--------------------------------------+
| File       | Printer    |                                      |
+------------+------------+--------------------------------------+
| Open     | | Select   |
| New      | | Open     |
+----------+ | Connect  |
             +----------+
```

コード中の File、Printer、Open、New、Select、Connectの文字列は、gettextファミリーの関数によって翻訳される必要があります。しかし Openという文字列は、2ヶ所で使われており、それにたいして異なる翻訳を割り当てなければならないかもしれず、それゆえ上述したようなジレンマが発生します。

メニュー中の同じ 2 つの文字列を区別するのは、メニューのルートからそれらのエントリーへのパスです:

```
Menu|File
Menu|Printer
Menu|File|Open
Menu|File|New
Menu|Printer|Select
Menu|Printer|Open
Menu|Printer|Connect
```

したがって context は、メニューのパスから最後の部分を除いたものになります。そうすると呼び出しは以下のようになるでしょう:

```
pgettext ("Menu|", "File")
pgettext ("Menu|", "Printer")
pgettext ("Menu|File|", "Open")
pgettext ("Menu|File|", "New")
pgettext ("Menu|Printer|", "Select")
pgettext ("Menu|Printer|", "Open")
pgettext ("Menu|Printer|", "Connect")
```

context の最後に、区切り文字の '|'をつけるかどうかは、スタイルの問題になります。

より複雑なケースとしては、*msgctxt*や *msgid*が文字列リテラルでない場合があります。そのようなケースにたいしては、より一般的なマクロが利用できます:

```
const char *pgettext_expr (const char *msgctxt, const char *msgid);
const char *dpgettext_expr (const char *domain_name,
                            const char *msgctxt, const char *msgid);
const char *dcpgettext_expr (const char *domain_name,
                             const char *msgctxt, const char *msgid,
                             int category);
```

これらのマクロは、*msgctxt*と *msgid*に、任意の文字列変数を指定できるので、より一般的です。どちらの引数も文字列リテラルのときは、'_expr'が付加されていないマクロのほうが効率的です。

11.2.6 複数形 (plural forms) にたいする追加の関数

いままで説明してきた既存のアプローチでは完全に無視してきましたが、gettextファミリーの関数 (すべての catgets関数も同様) を実世界で使用する場合は、1 つの問題があります。それは plural form(複数形書式) の取り扱いです。

インターナショナリゼーション以前の Unix のソースコード (そして悲しいことにインターナショナリゼーション以降のソースコードさえも) を見てみると、以下のようなコードを目にすることがあります:

```
printf ("%d file%s deleted", n, n == 1 ? "" : "s");
```

最初にコードをインターナショナライズする人々から不具合が報告された後は、このような複数形についての場合分けを無視するか、"file(s)"という文字列を使うようになりました。これはどちらも不自然で解決法とはなりませんでした。最初の試みとして、以下のようにすることで解決が図られました:

```
if (n == 1)
  printf ("%d file deleted", n);
else
  printf ("%d files deleted", n);
```

しかし、これは問題の解決になっていません。この方法は、単純に名詞に 's' を追加することにより plural form を表現するだけではない言語をも対象としていましたが、結局それだけのことでした。人々は自分が使っている言語のルールが、他の言語にも適用できる普遍的なものだと信じるという罠に陥っていたのです。実際には言語ファミリー間で plural form の取り扱いかたは大きく異なっていたのです。たとえば、Rafal Maszkowski <rzm@mat.uni.torun.pl>による以下のレポートを見てください:

> わたしたちは Polish で plik(file のこと) を以下ように表現します:
>
> ```
> 1 plik
> 2,3,4 pliki
> 5-21 pliko'w
> 22-24 pliki
> 25-31 pliko'w
> ```
>
> 以下同様にカウントしていきます (o' は 8859-2 oacute を意味しており、okreska より、むしろ aogonek と似ています。

言語間 (そして言語ファミリー内部においてさえも) には、2 つの異なる事象が存在しました:

- plural form を構築する方式が異なること。これは規則的ではないルールを多くもつ言語で問題になります。極端なケースとしては German があげられます。English から German へは、同じ言語ファミリー (Germanic) ですが、名詞に ('s') を付加して複数形を表す標準的な方式を、German ではほとんど見出せません。
- plural form を適用するべき数値が異なること。これは、複数形にすべき数値が単に 2 からであるような、Romanic や Germanic しか経験したことのない人を驚かせます。

  単一の形式しかもたない言語ファミリーや、多数の形式をもつ言語ファミリーがあります。これに関する更なる情報は、このセクションの範囲を超えています。

これらの理由により、アプリケーションを記述する人は、これらの問題をコードで解決するべきではないという結論になります。これは言語環境にたいしてハードコードされた状態においてのみ有用なローカリゼーションでしょう。かわりに拡張された gettextのインターフェースを使うべきです。

これらの追加の関数は、単一のキー文字列ではなく、2 つの文字列と、数値の引数をとります。この考え方の背景には、数値の引数と最初の文字列をキー文字列として使用することにより、翻訳者が

指定した正しい plural form を実装が選択できるようにするというアイデアがあります。それから、2 つの文字列引数は、メッセージカタログが見つからなかったときの戻り値を提供するために使用されます (これは通常の gettextの振る舞いと同じです)。このケースでは Germanic 言語のルールが適用され、最初の文字列引数は singular form、2 番目の文字列引数は plural form とみなされます。

この結果、言語カタログをもたないプログラムは、それが Germanic 言語のルールにしたがって記述された場合のみ正しい文字列を表示できるということになります。これは確かに制限なのですが、GNU C library(と GNU gettextパッケージ) が GNU パッケージの一部として記述されていること、そして GNU プロジェクトのコーディング規約が English によるプログラム記述を要請しているので、制限があるにもかかわらずこの解決策により要件を満足することができるのです。

`char * ngettext (`*const char *`**`msgid1`**`, const char *`**`msgid2`**`, unsigned long int `**`n`**`)` [Function]

ngettext関数は、gettext関数と似ており、メッセージカタログを検索する方法は同じですが、2 つの追加の引数をとります。パラメーター *msgid1*には、変換する必要のある文字列の singular form を指定します。このパラメーターはカタログを検索するキーとしても使用されます。パラメーター *msgid2*には plural form を指定します。パラメーター *n*は、plural form の適用を決定するのに使用されます。メッセージカタログが見つからなかったとき、n == 1なら msgid1が戻され、それ以外のときは msgid2が戻されます。

以下はこの関数の使用例です:

```
printf (ngettext ("%d file removed", "%d files removed", n), n);
```

*n*の数値は、printf関数にも引き渡されることに注意してください。ngettextだけに渡したい場合、この例は不適切です。

English の singular case の場合、常に 1 であるような数値は、"one"に置き換えることができます:

```
printf (ngettext ("One file removed", "%d files removed", n), n);
```

‘printf’関数は、format string が与えられていない余分な引数を無視するので、この例は問題なく動作します。

ここで 2 つ以上の引数を要求するような format string を関数で使う場合、以下の例のような使い方はできません:

```
printf (ngettext ("%d file removed from directory %s",
                  "%d files removed from directory %s",
                  n),
        n, dir);
```

English で singular case のときに ‘%d’を “one” に置き換えたように、他の言語の翻訳者も特定の単語に置き換えたいと望むかもしれません。しかし C の format string では、1 番目の引数をスキップして 2 番目の引数を使用するようなことはできません。このような場合は引数の順番を変えて、‘n’が最後にくるようにする必要があります:

```
printf (ngettext ("%$2d file removed from directory %$1s",
                  "%$2d files removed from directory %$1s",
                  n),
        dir, n);
```

このように引数の配置を指定する文法についての詳細は、Section 15.3.1 [c-format], page 155 を参照してください。

nの値がとる範囲がわかっている場合には、xgettextツールのためのコメントを指定することができます。以下の例のような情報は、翻訳者が適切な翻訳を行う助けとなるでしょう:

```
if (days > 7 && days < 14)
  /* xgettext: range: 1..6 */
  printf (ngettext ("one week and one day", "one week and %d days",
                    days - 7),
          days - 7);
```

以下の例のように、文字列に数値が含まれないようなときに、この関数を使うこともできます:

```
puts (ngettext ("Delete the selected file?",
                "Delete the selected files?",
                n));
```

この例で、*n*は plural form であるかを判定するためだけに使用されています。

`char * dngettext` (*const char ***domain**, const char ***msgid1**, const char ***msgid2**, unsigned long int* **n**) [Function]

`dngettext`は、選択されたメッセージカタログにたいして、`dgettext`と同様な方法で使用します。異なるのは、正しい plural form のために、2 つの余分なパラメーターを指定できる点です。この 2 つのパラメーターは、`ngettext`のときと同様に処理されます。

`char * dcngettext` (*const char ***domain**, const char ***msgid1**, const char ***msgid2**, unsigned long int* **n**, *int* **category**) [Function]

`dcngettext`は、選択されたメッセージカタログにたいして、`dcgettext`と同様な方法で使用します。異なるのは、正しい plural form のために、2 つの余分なパラメーターを指定できる点です。この 2 つのパラメーターは、`ngettext`のときと同様に処理されます。

では、これらの関数はどのようにして plural forms の問題を解決しているのでしょうか? 言語学の情報 (そして、それは利用可能ではありません) がないかぎり、少しの差異しかない plural form のうちからどれを使用すればよいのか、そして新たにサポートされる言語ごとに数を増やせるかを決定することはできません。

したがって、plural form を選択するルールを翻訳者が指定するという解決策が実装されました。各言語ごとに方式が異なる以上、これはコード内に情報をハードコーディングする以外の、唯一可能な解決策なのです (それでもまだ新しい言語の使用を妨げない拡張可能性を満たすという要件は残されています)。

plural form 選択のための情報は、以下のように PO ファイルのヘッダーエントリー (`msgid`が空文字列のエントリーのうちの 1 つ) に保存されています:

```
Plural-Forms: nplurals=2; plural=n == 1 ? 0 : 1;
```

`nplurals`には、その言語では何種類の plural form があるのかを数字で指定します。`plural`に続く文字列には、C 言語で使用できる評価式です。負の数値は使用できません。数値には正の整数を指定します。また、変数として使用できるのは`n`だけです。式で空白を使うことはできますが、バックスラッシュは使用できません。以下の例のうち、バックスラッシュと改行が使用されている例がありますが、これは表示を見やすくする目的のためだけに使用しています。この式は`ngettext`、`dngettext`、`dcngettext`が呼び出されたときに評価されます。これらの関数に数値が渡されると、式中の変数`n`の値として、その数値が評価されます。結果は 0 以上、かつ`nplurals`に指定した値より小さくなければなりません。

複数形の使い分けについては、以下のルールが知られています。言語と言語のファミリーが記載されていますが、この情報を言語ファミリー全体に適用する必要があるという訳ではありません (見やすくするために併記しています)。[3]

[3] 追加の情報を歓迎します。`bug-gnu-gettext@gnu.org`か`bug-glibc-manual@gnu.org`までメールしてください。

1つの形式だけを使うもの:

1つの形式しか必要としない言語があります。これらの言語ではsingular formとplural formの区別はありません。この場合の適切なヘッダーエントリーは以下のようになるでしょう:

```
Plural-Forms: nplurals=1; plural=0;
```

このような特性をもつのは以下の言語です:

Asian family
: Japanese, Vietnamese, Korean

2つの形式がありsingularは1にしか適用しないもの

この形式はEnglishで使われているもので、既存のプログラムでも一番多く使用されています。この場合のヘッダーエントリーは以下のようになるでしょう:

```
Plural-Forms: nplurals=2; plural=n != 1;
```

(注意: これはCの真偽値が0か1の2値をとる機能を使用しています。)

このような特性をもつのは以下の言語です:

Germanic family
: English, German, Dutch, Swedish, Danish, Norwegian, Faroese

Romanic family
: Spanish, Portuguese, Italian, Bulgarian

Latin/Greek family
: Greek

Finno-Ugric family
: Finnish, Estonian

Semitic family
: Hebrew

Artificial Esperanto

同じヘッダーエントリーを使う他の言語:

Finno-Ugric family
: Hungarian

Turkic/Altaic family
: Turkish

Hungarianは、数字を含んだセンテンスでは複数形を使いません。たとえば “1 apple” は “ 1 alma ” で、“123 apples” も “123 alma” です。しかし数が明確でない場合には “the apple” は “az alma”、“the apples” は “az almák” のようにsingularとpluralを区別します。このようなセンテンスを`ngettext`がサポートするようになってから、Hungarianも2つの形式をもつクラスに分類されるようになりました。

Turkishも同様です: “1 apple” は “1 elma” で、“123 apples” も “123 elma” です。しかし数字が省略された場合には、“the apple” は “elma”、“the apples” は “elmalar” のようにsingularとpluralを区別します。

2つの形式がありsingularを0と1に適用するもの

言語ファミリーの例外的なケースです。ヘッダーエントリーは以下のようになります:

```
Plural-Forms: nplurals=2; plural=n>1;
```

このような特性をもつのは以下の言語です:

Romanic family
: Brazilian Portuguese, French

3 つの形式があり、0 を特別なケースとして扱うもの
: ヘッダーエントリーは以下のようになります:

```
Plural-Forms: nplurals=3; plural=n%10==1 && n%100!=11 ? 0 : n != 0 ? 1 : 2;
```

このような特性をもつのは以下の言語です:

Baltic family
: Latvian

3 つの形式があり、1 と 2 を特別なケースとして扱うもの
: ヘッダーエントリーは以下のようになります:

```
Plural-Forms: nplurals=3; plural=n==1 ? 0 : n==2 ? 1 : 2;
```

このような特性をもつのは以下の言語です:

Celtic
: Gaeilge (Irish)

3 つの形式があり、00 または [2-9][0-9] で終わる数字を特別なケースとして扱うもの
: ヘッダーエントリーは以下のようになります:

```
Plural-Forms: nplurals=3; \
    plural=n==1 ? 0 : (n==0 || (n%100 > 0 && n%100 < 20)) ? 1 : 2;
```

このような特性をもつのは以下の言語です:

Romanic family
: Romanian

3 つの形式があり、1[2-9] で終わる数字を特別なケースとして扱うもの
: ヘッダーエントリーは以下のようになります:

```
Plural-Forms: nplurals=3; \
    plural=n%10==1 && n%100!=11 ? 0 : \
           n%10>=2 && (n%100<10 || n%100>=20) ? 1 : 2;
```

このような特性をもつのは以下の言語です:

Baltic family
: Lithuanian

3 つの形式があり、1、または 2、3、4 で終わる数字を特別なケースとして扱うもの、ただし 1[1-4] で終わる数字を除く
: ヘッダーエントリーは以下のようになります:

```
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;
```

このような特性をもつのは以下の言語です:

Slavic family
: Russian, Ukrainian, Belarusian, Serbian, Croatian

3 つの形式があり、1 と、2、3、4 を特別なケースとして扱うもの
: ヘッダーエントリーは以下のようになります:

```
Plural-Forms: nplurals=3; \
    plural=(n==1) ? 0 : (n>=2 && n<=4) ? 1 : 2;
```

このような特性をもつのは以下の言語です:

Slavic family
: Czech, Slovak

3 つの形式があり 1 と、2、3、4 で終わる数字を特別なケースとして扱うもの
: ヘッダーエントリーは以下のようになります:

```
Plural-Forms: nplurals=3; \
    plural=n==1 ? 0 : \
           n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20) ? 1 : 2;
```

このような特性をもつのは以下の言語です:

Slavic family
: Polish

4 つの形式があり 1 と、02、03、04 で終わるすべての数字を特別なケースとして扱うもの
: ヘッダーエントリーは以下のようになります:

```
Plural-Forms: nplurals=4; \
    plural=n%100==1 ? 0 : n%100==2 ? 1 : n%100==3 || n%100==4 ? 2 : 3;
```

このような特性をもつのは以下の言語です:

Slavic family
: Slovenian

ここまで読んで、あなたは思うかもしれません。`ngettext`が扱う *n*の型は‘`unsigned long`’だ。では、もっと大きな整数型の場合はどうだろうか? 負の数値については? 浮動小数値の場合は?

‘`uintmax_t`’や‘`unsigned long long`’のようなより大きな数値のための整数型の場合、これらの数値は‘`unsigned long`’の範囲に適合するように値を小さくして処理できます。この場合、単に値を‘`unsigned long`’にキャストするのは正しくありません (キャストでは`ULONG_MAX + 1`は 0、`ULONG_MAX + 2`は 1、... のようにキャストされます)。あなたは、いままで紹介してきたすべての plural form で、100(または 1000 や 1000000) で除する方法によって数式を間接的に評価できるという事実を見てきたでしょう。もし大きな数値を、下 6 桁を保持して [1000000, 1999999] という範囲の他の数値に置き換えられれば、同じ plural form の選択方法で取り扱えます。この場合のコードは以下のようになるでしょう:

```
#include <inttypes.h>
uintmax_t nbytes = ...;
printf (ngettext ("The file has %"PRIuMAX" byte.",
                  "The file has %"PRIuMAX" bytes.",
                  (nbytes > ULONG_MAX
                   ? (nbytes % 1000000) + 1000000
                   : nbytes)),
        nbytes);
```

負の数や小数については、通常は singular か plural が明解でない物質量に適用されます。このようなケースでは、`ngettext`を使う必要はなく、単にすべての値に適切な形式を指定して `gettext`を呼び出します:

```
printf (gettext ("Time elapsed: %.3f seconds"),
        num_milliseconds * 0.001);
```

*num_milliseconds*が 1000 の倍数のようなときでも、出力は以下のようになります

```
Time elapsed: 1.000 seconds
```

これは、English や他の言語でも許容できる出力です。

plural form にたいする翻訳者の考え方については、Section 12.6 [Translating plural forms], page 131 で説明しています。

11.2.7 *gettext 関数の最適化

この点を議論するには、GNU gettextの実装の優位性について話す必要があります。インターナショナライズされたプログラムは、翻訳する必要のある文字列がループ内にあるような場合に性能が劣化すると思う読者がいるかもしれません。たしかにループを実行するごとに文字列が評価されることによる劣化は無視できません。ループの実行中に文字列が変化しない場合に毎回文字列を翻訳する場合は、時間の無駄になります。以下の例で考えてみましょう:

```
{
  while (...)
    {
      puts (gettext ("Hello world"));
    }
}
```

選択した locale が実行中に変更されないような場合、翻訳結果の文字列は常に同じです。以下のようなやり方も 1 つの方法です:

```
{
  str = gettext ("Hello world");
  while (...)
    {
      puts (str);
    }
}
```

しかしこの解決策は、すべての状況で使える訳ではありません (例: 実行中に locale が変更される場合)。また、コードも読みにくくなってしまいます。

この理由により、GNU gettextは以前の結果をキャッシュしています。同じ翻訳が 2 度要求された場合、要求の間に新たなメッセージカタログがロードされていなければ、2 度目の呼び出しではgettextは結果をキャッシュから取得します。

11.3 2 つのインターフェースの比較

以下の議論は幾分誇張されたものかもしれません。これまで述べてきたように、わたしたちは Uniforum の勧告にしたがう理由があって、GNU gettextを実装しました。しかし、どのようにしてこの決定に至ったかをお見せするべきでしょう。

最初に開発プロセスを概観してみましょう。わたしたちが gettextにより提供される NLS を使ってアプリケーションを記述するときは、いつものとおり処理のことです。ユーザーの目に触れるので文字列を翻訳する必要があるときだけ、わたしたちは"..."のかわりにgettext("...")を使って翻訳を行います。各ソースファイル (または中核となるヘッダーファイル) で、以下を定義します

```
#define gettext(String) (String)
```

この定義により、システムの C ライブラリー自体で gettextがサポートされていても、それを無効化できます。このコードをコンパイルすると、NLS コードを使わない場合と同じ結果が得られます。GNU gettextのコードを見ると、gettext("...")のかわりに_("...")を使っているのがわかる

でしょう。これにより翻訳可能な文字列のために余分にタイプしなければならない文字数を 3 文字まで減らせます。

これを出荷するバージョンのプログラムにする場合は、単に以下の定義

```
#define _(String) (String)
```

を、以下の定義に置き換えるだけで済みます。

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

そして、翻訳可能な文字列を含むすべてのソースファイルにたいして xgettextプログラムを実行します。わたしたちは翻訳が利用できないものにたいしても、利用可能になったら使えるように、プログラムを実行します。

同様のことは gettext_noop呼び出しでも行うことができます (Section 4.7 [Special cases], page 27 を参照してください)。gettext_noopは通常、no-op(訳注: no-operation = 何もしない) マクロとして定義します。プロジェクトでは以下のようなコードを考慮する必要があります:

```
#define gettext_noop(String) String
#define N_(String) gettext_noop (String)
```

N_は、_と同様、省略形です。GNU gettextの po/ディレクトリーにある Makefileは、これらの省略形を認識するので、必要に応じて使うことができます。

今度は catgetsを見てみましょう。主な問題点はプログラマー向けの機能にあります。彼は、翻訳可能な文字列ごとに、毎回異なる数字 (または記号定数) を割り当てる必要があります。彼は重複したエントリー、重複したメッセージ ID、etc... にも注意を払わなければなりません。もし GNU gettext プログラムが提供するのと同じ品質をメッセージカタログにもたせたい場合、文字列にたいする説明コメントやソースコード中での場所をメッセージカタログに記述しなければなりません。これはほとんど Mission: Impossible でしょう。

しかし catgetsの優位性を語る人たちが触れる点もいくつかあります。文字列内にある単語があり、その文字列が異なるコンテキストや他の言語で使われている場合に、その単語は異なる翻訳をもつことができます。以下に例を示しましょう:

```
printf ("%s: %d", gettext ("number"), number_of_errors)

printf ("you should see %d %s", number_count,
        number_count == 1 ? gettext ("number") : gettext ("numbers"))
```

この例では、"number"という文字列を 2 回翻訳する必要があります。たとえあなたが English に類する言語を話さなくても、この単語が 2 つの文で異なる意味をもつかもしれないことに気づくでしょう。German では、1 番目にたいして"Anzahl"、2 番目には"Zahl"と翻訳する必要があります。

これであなたはこのが難解な例だということに同意するでしょう。そしてあなたは間違っていません! では問題を正確に把握して、その問題がそれほど深刻ではないことにも気づくはずです。この問題は以下のようにして簡単に解決することができます:

```
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)
```

わたしたちは、この方法で文字列の競合はすべて解決できると信じます。もし競合する文字列の一方を変更するのが困難なら、もう一方の文字列を少しだけ変更することも考慮できます。これを克服するのは不可能ではありません。

catgetsは、同じ元文字列にたいして異なる翻訳をもたせることができます。gettextでは同じ元文字列にたいして異なる翻訳をもたせることはできませんが、この種のあいまいさによる問題を解決する、よりスケーラブルな解決策があります。Section 11.2.2 [Ambiguities], page 111 を参照してください。

11.4 独自のプログラム内で libintl.a を使用する

ライブラリーのバージョン 0.9.4 から始める場合、libintl.hは自己充足的になっています (例: 追加の関数なしでプログラムで使用できます)。ヘッダーとライブラリーはMakefileにより、$(prefix)で選択されたディレクトリーに配置されます。

11.5 gettextを根底から理解する

注意**:** このセクションの文書は時代遅れになっているので、改訂する必要があります。

ソースコードを読むことは、GNU gettextの機能を完全に活用するための助けになるでしょう。しかし、(時として込み入った内容の) コードを読むために時間費やすことを望まない人々のために、幾つかのコメントを挙げておきます:

- 実行時に言語を変更する

  対話的なプログラムにおいては、使用する言語を実行時に尋ねたほうが有用であることがあります。これを理解するためには、gettext関数が使用する言語をどのように決定しているのかを知る必要があります。ここに示す方法は GNU による gettext関数の実装においてのみ正しいものです。

  dcgettext 関数は呼出しごとに、環境変数の中で最高のプライオリティを持つものを探し出して使用します。プライオリティは以下のリストで示されます。プライオリティは下にいくにつれて低くなります。

  1. LANGUAGE
  2. LC_ALL
  3. LC_xxx、選択されたロカールによる
  4. LANG

  その後、検索された値を用いてパスが設定され、可能であれば翻訳ファイルがロー ドされます。

  「今」とは、LANGUAGEが変更されたときです。上で説明した過程に従い、この変数の新しい値はdcgettext関数が呼び出された時点で決定されます。これは (おそらく) 異なったメッセージカタログがロードされることを意味します。即ち、使用する言語が変更されるのです。

  しかし、これは一つの小さなフックに過ぎません。 gcc 2.7.0 以上のコードでは幾分の最適化が図られています。この最適化は通常、dcgettext関数の呼び出しによって新しいカタログがロードされる前に行われます。しかし、もし dcgettext が呼び出されなければ、プログラムもまたLANGUAGE変数の値が変更されたことを知ることができないでしょう (Section 11.2.7 [Optimized gettext], page 121 を参照してください)。この解決方法は非常に簡単です。以下のコードを言語変更関数の前に 置けばよいのです。

```
/* Change language.  */
setenv ("LANGUAGE", "fr", 1);

/* Make change known.  */
{
  extern int  _nl_msg_cat_cntr;
```

```
    ++_nl_msg_cat_cntr;
}
```

変数_nl_msg_cat_cntrは loadmsgcat.c中で定義されています。あなたはこれが何のためのものなのか知る必要はありません。しかし、これは、ある gettext実装が GNU gettext なのか、それとも非 GNU システムのネイティブの gettext 実装なのかを決定するために使用できます。

11.6 プログラマの章についての一時的なメモ

注意**:** このセクションの文書は時代遅れになっているので、改訂する必要があります。

11.6.1 一時的な情報 - 二つの実装

言語に依存せずにメッセージを扱う二つの手法があります。一つは X/Open の catgets によるものであり、もう一つは Uniforum の gettext によるものです。catgets による方法では、整数によってメッセージを指定します。gettext による方法では、英語のメッセージによって指定します。catgets による方法は長く使われ、多くのベンダでサポートされています。gettext による方法はは Sun でサポートされていて、COSE マルチベンダイニシアティブ (COSE multivendor initiative) がサポートするようです。どちらも POSIX 標準とはなりませんでした。POSIX.1 委員会では、この件に関して様々な見解の相違がありました。

二つの手法のいずれも POSIX 標準ではありません。gettextと catgets(XPG) ルーチンのいずれを標準として採用するかについて、POSIX.1 委員会でははは様々な議論がなされました。委員会の終盤に至っても何らの合意は得られず、結局メッセージングシステムは標準規格には含まれませんでした。私は XPG3 メッセージングインターフェースに関する追記を標準に付与するのは有益であると信じていますし、“... メッセージングシステムの実例は既に実装されているのです...”

委員会は、ある一つのインターフェースの実装を使うのが良いということをどの場所でも言わないように非常に注意していました。この話題に関するこれ以上の情報については、国際化プログラミング FAQ(Programming for Internationalization FAQ) を参照して下さい。

11.6.2 一時的な情報 - catgetsについて

catgetsを基盤として使用することに関する討議の末期には、幾つかの議論がありました。その議論の両側を提示することは大切だと思いますし、これから、私はちょっとしたことに対して「悪魔の弁護士」となってみることにします。

catgetsはもっと良くデザイン出来たでだろうということは否定しません。その実装には既に指摘したような制限が少なからずあります。

しかしながら、その一貫性と標準化の度合いについては申し分ありません。UNIX ソフトウェアを書くときに繰り返し発生する問題とは、UNIX プラットフォーム間での移植性に関する問題です。それは全ての UNIX ベンダがオペレーティングシステム上を調べて改良する部分を見つけたようなものです。疑いもなく、これらの修正は革新的なものであり、現実の問題を解決するものです。しかしながら、ソフトウェアベンダがこれらの変更を多くのプラットフォームで行いつづけるには、多くの労力が必要です。

そしてこれは各 UNIX ベンダが自社のシステムを標準化することを促進します。Spec1170 に準拠するためにです。各主要 UNIX ベンダはこの標準化のために委員会を設けました。そして全ての UNIX ソフトウェア開発者はこの標準に従ってソフトウェアを作成し、異なるプラットフォームへソフトウェアを導入する際には (autoconf を使うことなく) リコンパイルするだけで済むようになる日を心待ちにし ているのです。

私の理解しているところでは、Spec1170 は X/Open Portability Guidelines のバージョン 4(XPG4) に基づいたものです。catgetsとその眷属が XPG4 で定義されているので、私はcatgetsが Spec1170 の一部であり、それがすべての UNIX システムの標準的なコンポーネントになることを信じています。

11.6.3 一時的な情報 - なぜ一つの実装なのか

メッセージカタログにアクセスするために二つの異なるシステムをインストールすることは不経済なことのように思えます。我々がcatgetsの不足しているものを改善したいと思ったのなら、なぜ新しいシステムを実装するのではなく、catgetsを (互換性を保ちながら) 拡張しようとしないのでしょうか? いずれにせよ我々は、メッセージカタログにアクセスするためのシステムをオペレーティングシステムに対して二つインストールすることになるでしょう。一つは GNU ソフトウェアのためルーチン群であり、もう一つはその他全てのソフトウェアのためのルーチン群 (catgets) です。傲慢でしょうか?

カタログにアクセスする別のシステムが実装されたと仮定してみましょう。我々がお奨めするのはどちらでしょうか? 少なくとも Linux システムに対しては、我々は可能な限り多くのソフトウェア開発者を呼び込む必要があります。そのため、我々はソフトウェア開発者が彼らのソフトウェアを移植しやすいようにする必要があります。そしてそれはcatgetsをサポートすることを意味します。我々はlibintlコードをlibc中に実装するでしょうが、libcには別のメッセージカタログに対するアクセス方法を同じように取り込まなければいけないということなのでしょうか? そして、libintlと非catgetsルーチンを組み合わせて使おうとする人達に関してはどうでしょうか?ソフトウェア開発者が彼らのソフトウェアを他のプラットフォームに移植する際、彼らはそのソフトウェアに単にlibintlを含めるだけでなく、フロントエンド (libintl) コードと、バックエンド (非catgetsアクセスルーチン) コードを付け加えようとするでしょう。

しかしメッセージカタログのサポートは氷山の一角に過ぎません。他のロカールカテゴリのデータはどうでしょうか。それらもまた、多くの相違点を持っています。我々はそれに対処することを諦めて、重複したルーチン群を別々に開発せねばならないのでしょうか (libintlをメッセージカタログサポート以上のものにすべきなのでしょうか)?

UNIX 上の改良可能な多くの部分と同じように、我々は将来に向けて改良を加えつつも、過去のものに対する互換性を落とさないようにしていました。

11.6.4 一時的な情報 - ノート

多くの実装が最終形式からかけ離れたものであったため、X/Open が標準形式を承認するのは非常に遅くなりました。私の使っている両方のシステム (古い Linux catgets と Ultrix-4) には奇妙なバリエーションがあります。

最後の変更を加えた後、私は GNU/Linux libcの gettext関数群を作成するために時間を割かねばなりませんでした。従って、将来的には Solaris が gettextを備えた唯一のシステムであるということはなくなります。

12 翻訳者の視点

12.1 イントロダクション 0

注意: このセクションの文書は時代遅れになっているので、改訂する必要があります。

GNU は国際化しつつあります! 翻訳プロジェクトは保守担当者、翻訳者、そしてユーザーを全てまとめるもので、そのため GNU ソフトウェアは徐々に多くの言語を喋ることが出来るようになります。

GNU gettextツールセットには、保守担当者がパッケージのメッセージを国際化するために必要となる全てがあります。また、パッケージが国際化された後で、翻訳者がメッセージの地域化を行う際の助けになるような便利なツールもあります。

翻訳プロジェクトを完遂するために、我々は自分の国の言葉を愛し、良く書くことが出来、そして他の翻訳者が話しているのと同じ言語で助けることの出来る (synergize) 能力を持った人間を数多く必要としています。もしあなたが翻訳チームでボランティアとして働くことを望むなら、該当する翻訳チームにメールを出して下さい。

各チームは Linux International の好意による自身のメーリングリストを持っています。*ll*`@li.org`というアドレスの *ll*をあなたの注目する言語の ISO 639 の二文字コードに置き換えることによって、その言語の翻訳チームに連絡できます。言語コードは ISO 3166 に定められている国コードと同じではありません。現時点では以下の翻訳チームが存在します。

> 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`、Turkish `tr`。

仮に中国語翻訳チームにメールを出すとすれば、`zh@li.org`となります。翻訳チームのメンバーになるには、その言語チームのメーリングリストに登録する必要があります。例えば、スウェーデン人は本文に以下の内容を記述して `sv-request@li.org` にメールを出します。

```
subscribe
```

チームのメンバーは翻訳作業に興味を持つべきだということを心に留めて置いて下さい。そうでなければ翻訳を果たすことは難しいのです。もしが希望する言語のチームがまだ存在せず、その言語のチームを作りたいという場合には `coordinator@translationproject.org` まで連絡して下さい。それによって全ての翻訳チームの調整者に連絡が取れます。

一握りの GNU パッケージには幾つかの言語に対するメッセージの翻訳が適用・提供されています。翻訳チームは組織化され始めており、これらのパッケージを起点として使っています。しかしまだまだ多くのパッケージがあり、多くの言語についてはボランティアの翻訳者がいません。もし翻訳チームでボランティアとして作業したいと思うのでしたら、`coordinator@translationproject.org` に作業することの出来る言語を明記してメールを送って下さい。

12.2 イントロダクション 1

注意: このセクションの文書は時代遅れになっているので、改訂する必要があります。

現在公式に、GNU は国際化しつつあります! 以下は 1995 年 1 月の GNU Bulletin で述べられた声明です。

> 一握りの GNU パッケージには幾つかの言語に対するメッセージの翻訳が適用・提供されています。翻訳チームは組織化され始めており、これらのパッケージを起点として使っ

ています。しかしまだまだ多くのパッケージがあり、多くの言語についてはボランティアの翻訳者がいません。もし翻訳チームでボランティアとして作業したいと思うのでしたら、‘coordinator@translationproject.org’に作業することの出来る言語を明記してメールを送って下さい。

本ドキュメントはその過程に興味を持ったり、貢献したいと考えている人々が持つ多くの疑問に答えます。願わくばざっと目を通し、GNU の国際化に対するこの集合的努力から産み出される大量のメールの幾ばくかでもを担当して下さい。

広く共用される多くのフリーソフトウェアのプログラミングは英語で行われています。そして現在のところ、英語は GNU プロジェクトに協力する国家的コミュニティ間での主要なコミュニケーション言語として使われています。このドキュメントでさえも英語で書かれています。これは当面変わらないでしょう。

しかしながら、多くのソフトウェアで自国語や自国の習慣を用いたいという、国家的コミュニティからの強い欲求があります。また、GNU ソフトウェアをそのようにするための努力が現在も行われています。この試みは今までのところプリテスタからの熱心な反応の向上によって動かされており、我々は GNU の国際化は成功すると信じています。

このドキュメントに対する内容の明確化、追加、訂正に関する提案については、coordinator@translationproject.orgまでメールを送って下さい。

12.3 議論

注意**:** このセクションの文書は時代遅れになっているので、改訂する必要があります。

この国際化の効果を目の当たりにして、幾人かのユーザが彼らの考えを表明しています。紹介され議論されたこれらの疑問の幾つかをここに挙げてみましょう。

- より小さいグループ

  幾つかの言語は多くの人々によって話されていないため、その言語を話す人々はフリーソフトウェアパッケージの当該言語版に対する必要性はあまりないと考えています。更に、幾つかの国にいるコンピュータの中にいる多くの人々は一般的に英語版ソフトウェアのほうを好むようです。

  一方で、人々は自分達の言語を非常に好きですし、彼らのお気に入りのフリーソフトウェアが彼らの母国語を喋ることが出来るように努力します。彼らは個人的な楽しみのためにそれを行います。そしてどのくらいの人間がその作業によって便益を得るかなどといったことは考えません。

- 解釈の誤り

  ある種の誤ったプロパガンダのせいか、一部のユーザは自身の言語を押し出すことについて臆病になっています。ある人は、ネットワークの向こう側にはその言語をうるさくせがむユーザがいるに違いないと考えています。

  しかしあらゆる言語には地域化される価値があります。なぜなら、その言語が大切で敬愛されるものであると考える人々がその言語の向こうにいるからです。

- 変な翻訳

  誰もがメッセージを理解出来るためには、正しい翻訳を見つけ出すことがもっとも大きな問題となります。翻訳は通常、少し変なものなのです。一部の人々は、「どちらかといえば押しが強く、嫌でときどき滑稽な」彼らの言語に対する翻訳を行うことが出来る程度には英語を扱うことが出来ます。フランス語を話す人間として、私は韓国または台湾において商品の取扱説明書を貧弱なフランス語へと翻訳した経験があります。...

  我々はときどきある種の国家的計算機文化を作り上げる必要があるというのは事実です。そして、その作業は彼らの母国語によって繋がっている多くの人間の協調作業なしに簡単に出来ることで

はありません。翻訳は彼ら自身の言語を知りそして愛する人々によってより良く行われ、より良い結果を得るという点において一緒に作業されるものなのです。

- GPL(またはLGPL) への依存

  何人かの人々は、彼らが彼らのプログラムをフリーにしたくない場合、又は別の種類の自由を与えたい場合に、GNU `gettext`を使うことによって彼らのパッケージを GNU 一般公有許諾書 (GNU General Public License) の保護の元で配布する必要があるのではないかと思案します。これに対する単純な答えは “通常はいいえ” です。

  GNU `gettext`の `gettext-runtime`の部分 (たとえば `libintl`のコンテンツ) は、GNU Lesser General Public License により保護されています。GNU `gettext`の `gettext-tools`の部分 (たとえば、GNU `gettext`パッケージの残りの部分) は、GNU General Public License により保護されています。

  パッケージ中の僅かな地域化された文字列のマーキング、又は国際化のための条件つきの数行の包含はGPL またはLGPL のコードを含んでいません。しかしながら、`libintl`内の地域化ルーチンそれ自身はLGPL の元にあり、LGPL として考慮される必要があります。これは、たとえ非フリーなプログラムでさえ、変更されていない `libintl`の完全なソースコードを配布する権利を与えます。これはまた、非フリーなプログラムでさえ、共有ライブラリーとして `libintl`を使用する権利を与えます。しかしこれは、フリーなソフトウェアーだけにたいして、静的ライブラリーとして `libintl`を使用、または他のライブラリー内に `libintl`を含める権利を与えます。

12.4 組織

注意**:** このセクションの文書は時代遅れになっているので、改訂する必要があります。

大きな尺度で見れば、真の解決方法は有志が参加できるようなある種の正しく厳密な集合を組織化することでしょう。私は、最近このアイディアについて幾つかの考察を行い、幾つかの微妙なポイントがあるであろうことを認識しています。私は、そのようなプロジェクトを開始するために、Richard Stallman に連絡することを考えましたが、まず最初に我々の間でアイディアを揺り落とすことが良いだろうと感じました。おそらく、Linux International は既にこの分野における幾らかの経験、または、有志の作業のオーケストラのようなものを持っています。あらゆる場合に、思考のための食物を!

我々は早々に何らかの方法で何かをセットアップせねばならないと考えます。同じ言語に対する作業のインターロックと重複を避けるという点で、それは多くの言語のコントリビュータを助けるでしょう。そして、更にそれらの言語 (大部分の言語では技術的な英語の翻訳について独特な多くの問題点があります) についての独特な問題を共に解決するように連絡が取れるようにします。スウェーデンのコントリビュータはこれらの問題点を認め、そして私はフランス語においてのそれらの問題点に相当に気が付いています。

確かにこれは技術的問題ではありません、しかし我々は、コントリビュータ、及び管理者間の国家チーム層のインターフェースに関わらず、ロカールコントリビュータの努力が最大限に有益になるように管理するべきです。

翻訳プロジェクトは言語コーディネータを統合するためにある準備を必要とします。一度この作業が始められたなら、発展中のプログラムのローカライズは、確かにフリーソフトウェアコミュニティにおいて永久の、そして、連続的な動きになるでしょう。GNU `gettext`が公式の現実になる前に、最小限の準備が完了し、そしてテストされているべきです。電子メールアドレス `coordinator@translationproject.org`は、これらの話題に基づくボランティア、及び、一般的な電子メールから申し出を受けるための準備でした。このアドレスは、翻訳プロジェクトのコーディネータに届きます。

12.4.1 中央による調整

そのことについて考えるよりも更に早く、GNU は誰かがそれらのグループを組織化し調整する方法を準備する必要があると私もまた考えます。ある種のグループのグループです。GNU は直ちに共同で働いているボランティアの小さなグループにこのタスクを委託することが良いだろうと、私は考えています。おそらく、この国家委員会的なグループのリストは `gnu.announce`において公表され得ます。

コーディネータとしての私の役割は単に、Ulrich をフリーソフトウェアの地域化に興味を持っているドイツ語を話すボランティアに紹介すること、そして国家的グループの準備が出来るまでの国家的登録機関のメンテナンス中に、国家的グループの最初の組織化を助けることです。実際、コーディネータは、ボランティアが国家チーム (言語または国 (局地的言語) について 1 人のコーディネータを選択するべきです) を作成するために相互と連絡を取りやすくせねばなりません。これが正しく行われたならば、コーディネーションは不可抗力的作業を除いて便利なものとなり、代理人に時間を任せることが出来るようになります。

12.4.2 国家チーム

私は、我々が個々の言語のための有志のコーディネータ/エディタを捜すことを提案します。これらの人々は彼ら自身の言語のために、様々なプログラムの翻訳ファイルを探し出し、そして、語法に対する高度で一定の標準を保証することになるでしょう。

今までの他の人々との間の私の現在の経験によれば、地域化を実現する人々はこのプロセスに対して非常に熱心であり、彼らは自分自身が地域化するプログラムよりも地域化のプロセスにほうに興味を持ち、それだけでなく多くのプログラムを地域化したいと思うものです。この事実は、各言語のためのコーディネータ/エディタを持つことは良いアイディアであることを確信させます。

我々は、問題となる言語において明瞭かつ簡潔な文章を書く際、適任となる人物を選択する必要があります。これは難しい作業です — 我々は、自分自身でそれをチェックすることができません。従って、我々は数人の人間対してに互いの記述を判断するように要請し、そして最適任者を選択する必要があります。

私は私のプレリリースを 20 人から 30 人の人々に発表ましたが、そのプレリリースが既に生み出した全ての議論をあなたは信じないでしょう。私は、真に、公式に、世界中でこの作業が開始されるときに起こるであろうことを想像すると身震いがします。例えば、相互に反論しあう二人のチェコスロヴァキアのユーザーの間を仲裁するのは私なのでしょうか?

私がこれらの公式化について判断することが出来ないように、あなたのドイツ語が私のフランス語よりはるかに良いとは限らないと推測します。私が提案するものは、各言語に対して PO ファイルをメンテナンスしその変更を判定する人々のグループを置くということです。そのような人々のグループがどのように行動するかについて、グループ間には文化的な相違点があると考えます。幾つかのグループはは緩い方法を採用し、簡単にコンセンサスの一致に達し、グループ中の誰もが保守者に関わることができます。一方は死ぬまで戦い、重い管理を国家の標準にまで組織化し、厳密なチャネルを使用するでしょう。

ドイツのチームは良い例を出しています。直ちに、彼らはおそらくお互いの翻訳を訂正する半数の人々と言語上の論点について議論する半ダースの人々です。私は全ての名前を知っているわけではありません。Ulrich Drepper はドイツのチームのコーディネートを担当しています。彼は私のプレテストのリストの購読を申し込みました。従って、私は彼に対して、連絡されるリリースの詳細について警告する必要は特にありません。

各言語を担当する翻訳チームを得るためには、それはよいアイディアだと思います。 翻訳を更に良く首尾一貫した状態にするでしょう。

12.4.2.1 サブカルチャー

フランス語を例に取ってみましょう。コンピュータの世界では、意味が異なる語彙を持つ幾つかのサブカルチャーがあります。組織化された方法でこの問題を提起することなしにあちこちでボランティアを選んでいると、プロジェクトにはすぐに国際化されたプログラムのごちゃ混ぜ状態が発生します。そしてことによると、実際にこの問題を気にする人々の間で終りなき口論が始まるでしょう。

国際化されたプログラムをフランス語へ地域化する過程において、ある種の統一を保つことは難しい (そしてデリケートな) 仕事です。フランス人のラテンな人柄 (:-) を知っていても、もし我々がこのことを間違った方法で捉えれば、我々はどことも知れぬ場所で終わってしまうか、多くのエネルギーを無駄にしてしまうことでしょう。おそらく我々は、GNU `gettext`が公式に発表される前に真剣にこの問題に取り組まなければならないでしょう。そして、それはすぐではないかと私は推測します!

12.4.2.2 組織化へのアイディア

私は、公式リリース後に次の大きな変更があると考えています。どうか、私が短い GPL メッセージのドイツ語翻訳を用いることに注目してください。我々は、フリーソフトウェアコミュニティにおける真の地域化が消え去ってしまう前に 2、3 の良い例を示す必要があります、ここでは、議論が必要ないくつかのポイントを示します。

- 各グループは、一つの FTP サーバ (少なくとも一つのマスタサーバ) を持つべきです。
- サーバ上のファイルは、最新版 (もちろん!) を反映すべきであり、そしてサーバは、対応するするアーカイブと共に RCS ディレクトリ (私は今、これを持っていません) をもまた含んでいるべきです。
- 同じく ChangeLog ファイル (これは RCS アーカイブより有益ですが、しかし Emacs によって後から自動的に生成することが出来ます) を含んでいるべきです。
- コアグループは、疑わしい変更について判定するべきです (現在、このグループは私だけで構成されていますが、私は時折他の人間に「これもまた仕事に見える」と尋ねます)。

12.4.3 メーリングリスト

GNU `gettext`に関するあらゆる問合せについては、以下に送ってください。

`coordinator@translationproject.org`

`*-pretest`リストは、私にとって本当に有益です、アイディアはおそらく、多くの GNU、及び、非 GNU パッケージへと一般化されるでしょう。しかし、保守者以外の、彼/彼女の方法!

François、我々は、チーム、チームをサポートするメーリングリストそしてログメンバを追跡するために、`gnu.ai.mit.edu`に適当なメカニズムを持っています。我々は、あなたが使うわずかな優先権を持っています。これがあなたにとって問題ないならば、私はあなたに情報を与えることができます。

事物は変化しています! 2、3 年前、Daniel Fekete と私が GNU 地域化のメーリングリスト (FSF の中にあった) に尋ねたとき、我々は作業をどこででも組織化するように礼儀正しく勧められ、そして我々はそれを実行しました。私のプリテスタと連絡を取るために、私は`majordomo`で管理される少数のメーリングリストを iro.umontreal.ca に作成しました。これらのリストは今までのところ非常に信頼できました...

私は、ドイツ語のチームがドイツにあるメーリングリストを組織化し、他の国にも組織化をさせると思います。しかし組織化が行われる前に、FSF において各国のチームのためのメーリングリストを提供することは確かに有益でしょう。そうです、私にどのようにメーリングリストを作成し扱えばよいかを説明して下さい。

我々は一時的なメーリングリストを、人々を組織化しやすいように国ごとに一つずつ作らねばなりません。なぜ一時的か、なぜなら一度再構成されたなら、各国のボランティアは彼らのリストへと戻ってきて、そして自分達が望むように管理するだろうからです。このことについては、個々のチームは自分達の国の中から自分達のリストを動かすだろうと思います。全てのチームが購読することの出来る、 中央のメーリングリストも作る必要があるでしょう。

12.5 情報の流れ

注意**:** このセクションの文書は時代遅れになっているので、改訂する必要があります。

パッケージが最終的にリリースされた後、このメッセージについての幾つかの議論があることでしょう。今、人々が更に良い幾つかのメッセージを提案したとしたら、あなたはどうしますか? Jim、私が提供する 1 ダース近い地域化されたプログラムと同様、どうか直ちにそれらのメッセージに注目して下さい。私は翻訳とそれらに関する調整の両方を受け取るのです。

私が事前にテストするものを一つ置いたならば、Ulrich はその告知を受け取り、そしてそれを最後に修正するドイツ語のチームに渡します。そして彼は保守者として私に翻訳ファイルを引き渡します。私が保守していないフリーのパッケージについては何も聞きません。私が思うに、全ての翻訳プロジェクトにおいてこのスキームが作られるでしょう。セキュリティに関わる理由のために、おそらく Ulrich(実際の国家的コーディネータ) は時折翻訳プロジェクト (Jim、私、又は Len の新人) によって保持される中央のレジストリをアップデートすべきです。

私は小さな GNU パッケージは一週間に一つずつ、より大きなパッケージは数週間か数ヵ月をかけるという責務を私自身に課し、12 月か 1 月には私はに GNU の全パッケージを国際化する準備を積極的に整えていました。しかし、それはそのようには動きません。私は最初に、私が責任を持つ全てのことを行いました。私は他の保守者の幾らかの伝道作業に対して何も持っていませんでした。しかし私もまた多くのエネルギーを失いました— 同じ議論を繰り返します。

そして、最初に地域化されたパッケージがリリースされるとき、我々は、醜悪な翻訳 :-) についての多くの反応を得るでことしょう。確かに、そして我々は事前に、パッケージ保守者と国家チームの間の情報の流れを制御することに関する良いアイディアを持つ必要があります。

どうかどこかに各 PO ファイルの迅速なヒストリを保存し始めて下さい。コメントを認めることによってファイルフォーマットがいずれ変更されるであろうことを私は知っています。各ファイルがログのようなもの、そしてコメントや不平の申し立て、又はその他の貢献をしたいと思う人々へのリファレンスを持つほうがよいでしょう。私は高速でフレキシブルなフォーマットに関する申し立てをしましたが、しかしそれはまだ GNU の意思決定者によって受け入れられていません。私がこれについてより多くの情報を得たなら、このことについてお話しすることになるでしょう。

12.6 複数形の翻訳

あなたが PO ファイルを翻訳しようとしていて、それには以下のようなエントリーが含まれているとしましょう:

```
#, c-format
msgid "One file removed"
msgid_plural "%d files removed"
msgstr[0] ""
msgstr[1] ""
```

これはどういう意味なのでしょうか? どうやって記入すればよいのでしょうか?

このようなエントリーは、メッセージに plural form があることを示しています。plural form とは、メッセージ中の数字の値が複数形として出力しなければならない値の時に出力すべき文字列です。`msgid_plural`行に記述されているのは、English におけるそのようなメッセージの一般的な形式で

す。msgid行には、English における singular form で、数字の値が 1 のときに出力するテキストが記述されています。plural form についての詳細は、Section 11.2.6 [Plural forms], page 115 で説明しています。

最初に見る必要があるのは、PO ファイルのヘッダーエントリーの Plural-Formsという行です。この行には plural form を判定するための数字と式が記述されています。まだ PO ファイルにそのような行がない場合は、追加する必要があります。これは、あなたが翻訳しようとしている言語に依存します。この情報は msginitコマンド (Chapter 6 [Creating], page 41 を参照してください) – これには既知の plural formula のデータベースが含まれています – を使うか、翻訳チームの他のメンバーに尋ねてみてください。

以下のような行について考えてみましょう:

```
"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;\n"
```

これは論理的には 1 行になります。PO ファイルの書式では、各行を 80 文字までにおさまるように長い行を分割できることを思い出してください。

nplurals の値は、3 つの plural form があることを示しています。最初に行わなければならないのは、各形式ごとに msgstrを含むエントリーを作ることです:

```
#, c-format
msgid "One file removed"
msgid_plural "%d files removed"
msgstr[0] ""
msgstr[1] ""
msgstr[2] ""
```

それから msgid_pluralを翻訳して、各 msgstr行にそれを記述します:

```
#, c-format
msgid "One file removed"
msgid_plural "%d files removed"
msgstr[0] "%d slika uklonjenih"
msgstr[1] "%d slika uklonjenih"
msgstr[2] "%d slika uklonjenih"
```

では plural form に適合するように翻訳を改善しましょう。上述の式にしたがって、msgstr[0] には 1 で終わるが 11 では終わらない数字のときの翻訳を、msgstr[1] には 2、3、4 で終わるが 12、13、14 では終わらない数字のときの翻訳、そして msgstr[2] にはそれ以外のときに使用する翻訳を記述します。これにしたがって改善したものが以下の翻訳です:

```
#, c-format
msgid "One file removed"
msgid_plural "%d files removed"
msgstr[0] "%d slika je uklonjena"
msgstr[1] "%d datoteke uklonjenih"
msgstr[2] "%d slika uklonjenih"
```

English の singular form(msgid) では、数字用の書式指定が省かれて、数字の 1 をあらわす “one” という単語に置き換えられていることに気づくでしょう。あなたが翻訳するときも同じようできるでしょうか?

```
msgstr[0] "jednom datotekom je uklonjen"
```

これは msgstr[0]を 1 のときだけ使うのか、他の数字のときも使うかによります。plural formula に当てはめて考えると、msgstr[0]が n == 1のときだけ、数字用の書式指定子を使わない特定の翻訳文を使うことができます。しかしこの例の場合、msgstr[0]は 21、31、41... などのときにも使用するので、書式指定子を省くことはできません。

12.7 メッセージの優先度: 最初に翻訳すべきメッセージを決める方法

翻訳者が週のうちパッケージに避ける時間が限られているにもかかわらず、パッケージにはとても多くのメッセージ (1000 超) があるとしましょう。そのようなときは彼女は一番ユーザーの目に触れるメッセージ、または一番頻繁に発生するメッセージを訳したいと望むでしょう。このセクションでは、このような"もっとも緊急"なメッセージをどのように決定するか説明します。これは、すでに部分的に翻訳されたメッセージカタログの中から、"次に緊急"なメッセージを決定するのにも適用できます。

最初のステップとして、彼女は、ユーザーがプログラムを使うのと同様にしてプログラムを使ってみます。彼女がこれを行っているとき、プログラムがまだ翻訳されていないメッセージの翻訳にたいする要求があると、GNU `gettext`ライブラリーはそれをログファイルに記録します。

次のステップで、彼女は PO モードを使って、それらのメッセージを翻訳するのです。

より詳細に見てみましょう。GNU `libintl`(GNU `libc`の対応する関数とは異なる) は、環境変数`GETTEXT_LOG_UNTRANSLATED`をサポートします。GNU `libintl`ライブラリーは`gettext()`、および関連する関数が翻訳を見つけられなかったとき、そのメッセージをログに記録します。ログファイルがない場合には、必要に応じて作成します。GNU `libc`によるシステムでは、ELF の 'LD_PRELOAD' メカニズムで使用できる、共有ライブラリー '`preloadable_libintl.so`'が提供されます。

GNU `libc`のシステムでは、翻訳者は最初のステップとして以下のコマンドを実行します:

```
$ LD_PRELOAD=/usr/local/lib/preloadable_libintl.so
$ export LD_PRELOAD
$ GETTEXT_LOG_UNTRANSLATED=$HOME/gettextlogused
$ export GETTEXT_LOG_UNTRANSLATED
```

他のシステムでは以下のコマンドを使います:

```
$ GETTEXT_LOG_UNTRANSLATED=$HOME/gettextlogused
$ export GETTEXT_LOG_UNTRANSLATED
```

それから彼女はプログラムを使ってみます (あなたが翻訳を提供するプログラムを使うのはよいことですし、お勧めの練習方法です。これは必要なコンテキストを与えてくれます)。これが終わったら、彼女は環境変数を削除します:

```
$ unset LD_PRELOAD
$ unset GETTEXT_LOG_UNTRANSLATED
```

次のステップは、重複を取り除くことです:

```
$ msguniq $HOME/gettextlogused > missing.po
```

この結果は PO ファイルですが、PO ファイルエディターで処理するためには、少し前処理が必要です。最初に、このファイルは多くの翻訳ドメインのメッセージを含んだマルチドメインの PO ファイルです。次に、翻訳者のコメントとソースファイルへの参照が含まれていません。以下は、影響を受ける翻訳ドメインの一覧を得る方法です:

```
$ sed -n -e 's,^domain "\(.*\)"$,\1,p' < missing.po | sort | uniq
```

それから翻訳者はドメインを 1 つずつ処理していきます。単純にするために、language、domain、source package を環境変数に設定しましょう。

```
$ lang=nl             # your language
$ domain=coreutils    # the name of the domain to be handled
$ package=/usr/src/gnu/coreutils-4.5.4   # the package where it comes from
```

彼女は、`$lang.po`の最新コピーを、翻訳プロジェクト、またはそのパッケージ (大抵は `$package/po/$lang.po`) から入手します。もし彼女が、そのパッケージの最初の翻訳者の場合は、新規作成することになります (Chapter 6 [Creating], page 41 を参照してください)。それから彼女は、以下のコマンドを使って、緊急ではないメッセージにたいして"obsolete" のマークを付与します

(それらの翻訳済み、および未翻訳のメッセージが本当に "obsolete" としてしまう訳ではありません。これは以下の編集で、PO ファイルエディターに、それらのメッセージを無視させるためです)。

```
$ msggrep --domain=$domain missing.po | grep -v '^domain' \
  > $domain-missing.po
$ msgattrib --set-obsolete --ignore-file $domain-missing.po $domain.$lang.po \
  > $domain.$lang-urgent.po
```

それから彼女は PO ファイルエディターを使って`$domain.$lang-urgent.po`を翻訳します (Chapter 8 [Editing], page 51 を参照してください)。(FIXME: KBabel と gtranslator が期待通り obsolete message を保持してくれるかどうかについては、わたしにはわかりません) そして彼女は最後に、以下のコマンドで緊急ではないメッセージ (およびすでに翻訳済みのメッセージの初期の"翻訳") を復元します:

```
$ msgmerge --no-fuzzy-matching $domain.$lang-urgent.po $package/po/$domain.pot \
  > $domain.$lang.po
```

`$domain.$lang.po`を投稿したら、彼女は次のドメインを処理できます。

13 メンテナーの視点

パッケージのメンテナーには多くの責任があります。そのうちの 1 つは、たくさんのプラットフォームでパッケージを簡単にインストールできるようにすることで、わたしたちが前に説明したマジック (Chapter 2 [Users], page 9 を参照してください) を、インストーラーとエンドユーザーにたいして働くようにすることです。

GNU gettextをディストリビューションに統合できるようにする方法はたくさんありますが、このチャプターではそれらを総括的にカバーすることはしません。かわりに GNU 標準、さらには Gnits 標準にしたがった、多くのフリーソフトウェアディストリビューションで利用可能なアプローチの詳細について議論します。なぜなら GNU gettextは、GNU プロジェクト全体のインターナショナリゼーションを助けるのを目的としているので、多くの有用でフリーなパッケージが対象となるからです。そのためこのチャプターでは、すでに configure.acがあり、GNU Autoconf を使うパッケージを対象とします。

それにもかかわらず GNU gettextは、GNU 標準やその類にしたがっていないフリーパッケージにたいしても有用です。そのようなパッケージのメンテナーは自分の想像力と独創力によりディストリビューションを組織化する必要がありますが、gettextはすべての状況で動作するでしょう (そしてそのようなパッケージはたくさん存在します)。

gettextのメソッドは現在安定しているとはいえ、gettextの各バージョン間でちょっとした調整は必要になるでしょう。そのため、このチャプターの内容は、新たなリリースによる変更にしたがって読み替える必要があります。

13.1 非フラットなディレクトリー階層

フリーなソフトウェアパッケージの中には、tarにより配布され、解凍すると単層のディレクトリーに展開されるものがあります。このようなパッケージを *flat* なディストリビューションと呼びます。それとは別に、Texinfo マニュアルや man page のためのサブディレクトリー doc/、C ライブラリーを置き換えたり保管する関数を保持する lib/、パッケージのソースの入った src/などの階層をもつ、フリーなソフトウェアパッケージも存在します。このようなディストリビューションのことを、*non-flat* であると呼びます。

flat なディストリビューションにたいして、わたしたちはあまり多くを語ることはできません。GNU gettextを新しいバージョンにすることにより、flat なディレクトリー構造は難しさが増大するという欠点があります。たくさんの PO ファイルがある場合、この単層のディレクトリーの内容は汚くなってしまうでしょう。また C ソースに含まれる GNU gettextの libintl のソース、シェルやsedのスクリプト、そして複雑な Makefile のルールは、flat な構造には適しません。これらの理由により、non-flat を使う方法を推奨します。

GNU gettext自身も non-flat な構造をもち、わたしたちはこの方法に精通していることも、わたしたちがこのチャプターでそれを説明しようとしている理由なのかもしれません。これを機会に、パッケージの構造を non-flat にするメンテナーもいるかもしれません。

13.2 前提となる作業

パッケージにたいして GNU gettextを使用するためには、準備が少し必要です。これらの作業は、要点の説明だけだと肝心な部分がわからなくなってしまうある種の一般性をもっているので、このチャプターを読んで後から見返せるように、ここで大まかに説明しておきましょう。

- gettextizeを使う前に、最初に他のパッケージをインストールする必要があります。最新のバージョンの GNU m4、GNU Autoconf、GNU gettextがインストールされていることを確認し

てください。もしインストールされていない場合、それらを最初にインストールします。GNU Autoconf をインストールする前に (たとえ *configured* 済みだとしても)、GNU `m4`が完全にインストールされていなければならないことに注意してください。

パッケージ `automake`は、メンテナーのタスクを楽にするためにデザイン・実装されています。現在の GNU `gettext`もこれらのツールを使用しているので、`intl/`や `po/`にある `Makefile`は、1 つのプロジェクトで `automake`と `libintl`を使う場合に必要なものです。

これら 4 つのパッケージはメンテナーにとって必要なだけです。パッケージを正常にインストールして翻訳されたメッセージを正しく表示するだけなら、インストールする人やパッケージのエンドユーザーには、GNU `m4`、GNU Autoconf、GNU `gettext`、GNU `automake`は必要ありません。しかしパッケージにインターナショナライズされたシェルを含めて配布する場合、これは完全に真実とはいえなくなります: ユーザーがシェルスクリプトの翻訳されたメッセージを見たいときは、GNU `gettext`のインストールが必要になります。

- パッケージは `configure.ac`か `configure.in`をもち、Autoconf を使用する必要があります。もしそうなっていない場合は、どのようにするか学ぶ必要があります。Autoconf のドキュメントはとてもよく書けているので、印刷して読んでみるのもよい考えでしょう。
- このマニュアルで前述したように C ソースを変更する必要があります。詳細は Chapter 4 [Sources], page 19 を参照してください。
- `po/`ディレクトリーには、翻訳チームから提出された *ll*`.po`という名前の PO ファイルが、すべて保存されている必要があります。パッケージがインターナショナライズされて、それが利用可能になる前に翻訳作業を完了するのは、普通は簡単ではありません! メンテナーにとって翻訳作業のサイクルを開始するのに簡単なのは、最初は PO ファイルを何も用意せず、パッケージに興味をもつ翻訳チームが現れて PO ファイルが投稿されるのを待つことです。

PO ファイルが投稿されたとき、メンテナーとしてどのように振る舞うのが理想的なのか、少し説明をしておきましょう。メンテナーとしてのあなたの役割は、その投稿が翻訳プロジェクト内の対応する翻訳チーム (わからないときは `coordinator@translationproject.org`に転送してください) によるものであることを証明し、PO ファイルのフォーマットが壊れていてインストールできないくなっていないか確認し、それらの PO ファイルを配布物の `po/`ディレクトリーに配置することです。

メンテナーとしては、翻訳が十分なのか、または完璧なのかをチェックする責任を負う必要はないので、語学に関する事柄については無視するべきです。翻訳チームは、チーム自身の運営と翻訳プロジェクトでの言語学的な選択について完全な責任を負います。翻訳チームがメンテナーにより運営されるのではないことを覚えておいてください。ユーザーからの言語上の指摘や報告などを、適切な翻訳チームに転送したり、ユーザーが翻訳チームに参加する方法を説明するような手助けをすることはできます。もっとも簡単なのは `ABOUT-NLS`ファイルを送ることでしょう。

メンテナーが翻訳チームを介さずに、自分で PO ファイルに関するバグ報告を受けるのは決して行うべきではないことです。ある問題について翻訳者が彼女のチームと一致した見解をもつのが困難なとき、彼女が直接メンテナーと交渉するようなオプションが存在するべきではありません。どんな問題にせよ、彼らは問題をチーム自身で解決するべきです。メンテナーとしては、もしチームに本当に問題があると思えるときでも、あなた自身がチームの問題を解決しようとはしないでください。

13.3 `gettextize`プログラムの呼び出し

`gettextize`プログラムは、GNU `gettext`によりインターナショナライズされたパッケージのメンテナーを助けるための対話的なツールで、2 つの目的のために使用されます:

- 最初に、GNU `gettext`を使ってパッケージをするときは、ウィザードとして使用されます。

- パッケージ内での GNU gettextサポートを、以前の GNU gettextから新しいバージョンにアップグレードするための、移行ツールとして使用します。

このプログラムは以下のタスクを処理します:

- GNU gettextによりインターナショナライズされた、すべてのパッケージで必要なファイルをパッケージにコピーします。
- 次のセクション Section 13.4 [Adjusting Files], page 139 で説明する多くのタスクを、できるだけ自動的に処理します。
- 以前のバージョンの GNU gettextで使われていた陳腐化したファイルやイディオムを削除して、現在のバージョンの GNU gettextで推奨される形式にします。
- 手動で行うべきタスクや、gettextizeで自動的に処理できないタスクの要約をプリントします。

呼び出し方は以下のようになります:

```
gettextize [ option... ] [ directory ]
```

以下のオプションを指定できます:

‘-f’
‘--force’ すでに存在するファイルを強制的に置き換えます。

‘--intl’ libintl のソースを、intl/というサブディレクトリーにインストールします。この libintl は、GNU libintlがインストールされていないシステムでインターナショナリゼーションを提供するのに使用されます。このオプションが省略された場合は、configure.acの AM_GNU_GETTEXT呼び出しが読み込まれます。しかし ‘AM_GNU_GETTEXT([external])’ とインターナショナリゼーションは、GNU gettext のないシステムでは利用できないでしょう。

‘--po-dir=*dir*’
PO ファイルを含むディレクトリを指定します。このようなディレクトリーには、特定の POT ファイルをさまざまな言語に翻訳したファイルが含まれています。このオプションは、翻訳ドメインごとに複数回指定することができます。指定されなかったときは、po/というディレクトリーが更新されます。

‘--no-changelog’
ChangeLog の作成・更新をしません。デフォルトでは gettextizeは、影響を受けるディレクトリーごとの ‘ChangeLog’というファイルに、すべての変更 (ファイルの追加・変更・削除) を記録します。

‘--symlink’
必要なファイルをコピーするかわりに、シンボリックリンクを作成します。これはディスク容量を数キロバイト節約するのには便利ですが、自分自身を含む形式の tarball の作成には特別な配慮が必要になり、メンテナーがソースに適用できるいくつかの機能が使えなくなり、システムに新しいバージョンの gettextがインストールされたときにバグ (のような状態) を招きます。

‘-n’
‘--dry-run’
変更を出力しますが、処理は行いません。普通に gettextizeを実行したときのアクションはすべて抑止され、かわりにリストが標準出力に出力されるだけになります。

‘--help’ このヘルプを表示して終了します。

‘--version’
バージョン情報を表示して終了します。

*directory*を指定した場合、そのディレクトリーはGNU gettextを使う準備をしたいパッケージの、トップレベルのディレクトリーになります。

プログラム gettextizeは、以下のファイルを提供します。しかし--force (-f) オプションを指定しなければ、既存のファイルは置き換えられません。

1. ABOUT-NLSファイルは、パッケージのメインディレクトリー (トップレベル) にコピーされます。このファイルは、プログラムでNative Language Support機能をインストールして使う方法を示す主要なファイルです。もし手軽に入手できるなら、gettextizeにより提供されるABOUT-NLSよりも、新しいコピーを使いたいと思うかもしれません。より新しいABOUT-NLSファイルのコピーを、翻訳プロジェクト、またはGNU archive siteから入手することもできます。
2. 作成されたpo/ディレクトリーには、最終的にはすべての翻訳用ファイルが保持されますが、初期状態ではGNU gettextによるpo/Makefile.in.in(ファイル名に‘.in’が2つあることに注意してください) と、いくつかの予備ファイルしか含まれていません。すでにpo/というディレクトリーがあるときは、そのディレクトリーのファイルは保持され、Makefile.in.inと予備ファイルだけが上書きされます。

   ‘--po-dir’が指定されたときは、po/のかわりに‘--po-dir’で指定されたそれぞれのディレクトリーに配置されます。
3. ‘--intl’だけが指定されたときは、intl/ディレクトリーが作成され、GNU gettextのintl/ディレクトリーから、ほとんどのファイルがコピーされます。--force (-f) も指定されたときは、まず最初にintl/が空にされます。
4. ファイルconfig.rpathは、設定サポートファイルを含むディレクトリーにコピーされます。このファイルは、autoconfマクロAM_GNU_GETTEXTで必要です。
5. プロジェクトがGNU automakeしか使っていないときは、autoconfの一連のマクロファイルが、パッケージのautoconfマクロのレポジトリー (通常はm4/というディレクトリー) にコピーされます。

シンボリックリンクがサポートされている場合、gettextizeはパッケージのディレクトリーへは実際にコピーはされず、かわりにシンボリックリンクが作成されます。これによりすべてのパッケージで必要なファイルによる重複を避けることができます。単に‘-h’オプションを指定すると、配布物のtarアーカイブを作成するときには、それらのリンクが解決されて実際のファイルが配布物のアーカイブにコピーされます。そのため、メインのMakefile.inのゴールdistにたいするtarのオプションには、‘-h’を使う必要があることを強調しておきましょう

それだけではなく、gettextizeは、影響を受ける各ディレクトリーのMakefile.amをすべて更新し、同様にトップレベルのconfigure.ac(またはconfigure.in) も更新します。

パッケージのサブディレクトリーintl/、po/、m4/にコピーされる、GNU gettextをサポートするための最新のファイルを理解するのも、興味深いでしょう。intl/と他の2つのディレクトリーの違いは、intl/はGNU gettextを使うすべてのパッケージで同じですが、他の2つのディレクトリーのものの大部分はパッケージに依存したものだという点です。

gettextizeプログラムは、置換または変更するファイルのバックアップを作成して、それらの変更をChangeLogに書き込みます。この方法により、注意深いメンテナーはgettextizeを実行した後に、それによる変更が許容できるか確認して、可能なら調整することができます。このルールの例外はintl/ディレクトリーで、このディレクトリーは完全に追加・置換、または削除されます。

gettextizeが、GNU gettextを使うパッケージのための調整すべてを処理できる訳ではないことを理解するのことも重要です。残っている作業の量は、パッケージがGNU automakeを使うか

否かによります。それでも大抵の場合、メンテナーは gettextizeを呼び出した後、Section 13.4 [Adjusting Files], page 139 を読む必要があるでしょう。

特に 'gettexize'を使った後は、'AC_COMPILE_IFELSE was called before AC_GNU_SOURCE'、または 'AC_RUN_IFELSE was called before AC_GNU_SOURCE'というエラーが発生するかもしれません。このエラーは Section 13.4.5 [configure.ac], page 141 で説明している方法で configure.acを変更することにより修正できます。

gettextizeは、GNU build system の一部ではないので、自動的には呼び出されず、パッケージメンテナーとしての責任を持たない人も呼び出さないことを理解しておくのも重要です。後者の目的のためには個別にツールが準備されています。詳細は Section 13.6.3 [autopoint Invocation], page 151 を参照してください。

13.4 作成または変更しなければならないファイル

gettextizeにより自動的に追加されたファイルをのぞいて、GNU gettextと正常に対話するために修正が必要なファイルがたくさんあります。あなたが Makefile の設計と auto-configuration 自動設定のための GNU 標準に忠実にしたがっているなら、調整は容易でしょう。ここではそれぞれについて必要な変更を順に説明します。

以下では変更が必要なファイルと、必要な変更を説明していきます。多くの例は GNU gettext 0.18.3 のディストリビューション自体か、GNU helloディストリビューション (http://www.franken.de/users/gnu/ke/helloまたは http://www.gnu.franken.de/ke/hello/) から引用しました。GNU gettextのソースコードと GNU helloを参照してみれば、これらのパッケージが GNU gettextの機能を使うよい例だということが納得できるでしょう。

13.4.1 po/内の POTFILES.in

ディレクトリー po/には、POTFILES.inというファイルが必要です。このファイルは、すべてのプログラムソースの中で、翻訳が必要だとマークされた文字列をもつファイルがどれかを告げるもので、以下のような内容です:

```
# 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
```

#マークのコメントと空行は無視されます。それ以外の行は翻訳用にマークされた文字列を含むソースファイルをリストした行 (Section 4.4 [Mark Keywords], page 23 を参照してください) で、相対パスは POTFILES.inのあるディレクトリーではなく、ディストリビューション全体のトップレベルからの相対パスです。

flexや bisonのような、それ自身では翻訳可能な文字列を提供しないようなツールにより C ファイルが自動生成されるときは、po/POTFILES.inには自動生成された C ファイルではなく、本当の

ソースファイル (flexのときは.lで終わるファイル、bisonのときは.yというファイル) を記述することをお勧めします。

13.4.2 po/内の LINGUAS

ディレクトリー po/には、LINGUASというファイルも必要です。このファイルは利用可能な翻訳がリストされています。これは空白区切りのリストで、#マークのコメントと空行は無視されます。以下は例です:

```
# Set of available languages.
de fr
```

この例は German と French の PO ファイルが利用可能で、パッケージでは現在それらの言語がサポートされていることを意味しています。インストール時に、インストールされる言語にさらに制限をかけたいときは、ファイルLINGUASを変更するのではなく、環境変数LINGUASを使用します (Chapter 14 [Installers], page 153 を参照してください)。

LINGUASファイルには、'en@quot'と 'en@boldquot'という"言語"を追加することをお勧めします。en@quotは English のメッセージカタログ (en) の亜種で、非対称な体裁の ‘’と ‘’による ASCII の置き換えではなく、本当のクォーテーションマークを使います。en@boldquotは en@quotの亜種で、クォーテーション文字を太字のフォントで出力します。これは VT100 のエスケープシーケンスをサポートする端末エミュレーター (xtermや Linux の console。Emacs の *M-x shell*モードは該当しません) で使用されます。

これらの追加のメッセージカタログ 'en@quot'および 'en@boldquot'は、翻訳者が作成したのではなく、自動的に作成されたものです。これらのファイルをサポートするためには po/ディレクトリーに、Rules-quot、quot.sed、boldquot.sed、en@quot.header、en@boldquot.header、insert-header.sinというファイルが必要です。これらのファイルは、gettextizeを実行することによりインストールされます。

13.4.3 po/内の Makevars

ディレクトリー po/には、Makevarsというファイルもあります。このファイルには、プロジェクトで固有の変数が含まれています。po/Makevarsが作成されるときに、po/Makefileが挿入されます。そのため、変数は POT ファイルが作成・更新されたときや、メッセージカタログがインストールされたときに効果を及ぼします。

あなたのパッケージが単一のメッセージドメイン (1 つの po/ディレクトリーしかない) のときは、最初の 3 つの変数は変更する必要はありません。別々の場所に複数の po/ディレクトリーをもつパッケージの場合だけ、Makevarsの最初に定義された 3 つの変数を調整する必要があります。

XGETTEXT_OPTIONS変数のかわりに、autoconf マクロの AM_XGETTEXT_OPTIONにより、xgettextのオプションを指定することもできます。詳細は Section 13.5.6 [AM_XGETTEXT_OPTION], page 149 を参照してください。

13.4.4 po/内の Makefileの拡張

po/ディレクトリーの、Rules-*と呼ばれるファイルは、po/Makefileが作成されたときに追加されたファイルです。これらのファイルは、po/Makefile.in.inに干渉することなく、特定の PO ファイルの Makefile にルールを追加する機会を与えてくれます。

GNU gettext には、カタログ en@quot.poおよび en@boldquot.poをビルドするルールを含む、Rules-quotファイルが含まれます。en@quot.poの効果は、環境変数LANGUAGEに 'en@quot'をセットすると、クォーテーションを示す代替の ASCII grave accent と ASCII apostrophe のかわりに、対称性をもつ Unicode の正しいクォーテーションマークが表示されることです。このカタログを有効

にするには、単に en@quotを po/LINGUASに追加します。en@boldquot.poの効果は、LANGUAGE に ‘en@boldquot’をセットすると、正しいクォーテーションマークが得られるだけでなく、ターミナルやコンソールで表示されるクォーテーションマークの文字に、太字フォントが使用されることです。これは GUI プログラムではなく、コマンドラインのプログラムにとってだけ便利なカタログです。po/LINGUASファイルに en@boldquotを追加するだけで、このカタログを利用できます。

同様にして、sr locale – Cyrillic 文字で記述された Serbian – から、sr@latin locale – Latin アルファベットで記述された Serbian – のためのメッセージカタログを構築するルールを作成することができます。Section 9.4 [msgfilter Invocation], page 76 を参照してください。

13.4.5 トップレベルの configure.ac

configure.acまたは configure.in – これは autoconfが configureスクリプトを生成するときのソースになるファイルです。

1. パッケージとバージョンを宣言します。

   これは以下のように宣言します:

   ```
   PACKAGE=gettext
   VERSION=0.18.3
   AC_DEFINE_UNQUOTED(PACKAGE, "$PACKAGE")
   AC_DEFINE_UNQUOTED(VERSION, "$VERSION")
   AC_SUBST(PACKAGE)
   AC_SUBST(VERSION)
   ```

   GNU automakeを使っている場合は、以下のようになるでしょう:

   ```
   AM_INIT_AUTOMAKE(gettext, 0.18.3)
   ```

   もちろん、パッケージ名の ‘gettext’と、バージョン番号の ‘0.18.3’は、あなたのパッケージの名前とバージョン番号で置き換えます。これらは配布物のパッケージされた tarのファイル名 (この例では gettext-0.18.3.tar.gz) にそのまま使用されます。

2. インターナショナリゼーションにたいするサポートのチェック。

   以下のマクロは、インターナショナリゼーションにたいするサポートを発動するためにメインとなる、m4のマクロです。この行を configure.acに追加します:

   ```
   AM_GNU_GETTEXT
   ```

   マクロは configure 時に多くのチェックと処理を行いますが、呼び出しは故意に単純にしています。

   gettextizeを呼び出すとき、‘--intl’オプションを指定せず、intl/を作成しない場合は、以下の呼び出しが読み込まれます:

   ```
   AM_GNU_GETTEXT([external])
   ```

3. 出力ファイルの作成。

   AC_OUTPUT 命令は configure.acファイルの最後にあり、以下の 2 つの方法で変更する必要があります:

   ```
   AC_OUTPUT([existing configuration files intl/Makefile po/Makefile.in],
   [existing additional actions])
   ```

   AC_OUTPUTの最初の引数の変更は、intl/および po/ディレクトリーを置き換えるための変更です。接尾辞 ‘.in’は、po/だけに使用されることに注意してください。これにより配布される本当のファイルは po/Makefile.in.inであることがわかります。

   gettextizeを呼び出すとき、‘--intl’オプションを指定せず、intl/を作成しない場合は、AC_OUTPUTの行に intl/Makefileを追加する必要はありません。

必要な変更をした後は、‘aclocal -I m4’や‘autoconf’(または‘autoreconf’) などのコマンドは、以下のようなトレース情報を出力して失敗するようになります:

```
configure.ac:44: warning: AC_COMPILE_IFELSE was called before AC_GNU_SOURCE
../../lib/autoconf/specific.m4:335: AC_GNU_SOURCE is expanded from...
m4/lock.m4:224: gl_LOCK is expanded from...
m4/gettext.m4:571: gt_INTL_SUBDIR_CORE is expanded from...
m4/gettext.m4:472: AM_INTL_SUBDIR is expanded from...
m4/gettext.m4:347: AM_GNU_GETTEXT is expanded from...
configure.ac:44: the top level
configure.ac:44: warning: AC_RUN_IFELSE was called before AC_GNU_SOURCE
```

configure.acファイルの‘AC_PROG_CC’より後、かつ‘AM_GNU_GETTEXT’より前の箇所 (おそらく‘AC_PROG_CC’呼び出しのすぐ近く) に、‘AC_GNU_SOURCE’の明示的な呼び出しを追加する必要があります。この順番は、GNU autoconf による制限により必要です。

13.4.6 トップレベルの config.guess、config.sub

サブディレクトリー intl/の作成を省略しない場合、配布物に config.guessおよび config.subという、GNU のファイルを追加する必要があります。これらのファイルが必要なのは、intl/ディレクトリーが locale の文字エンコーディングを決定するという、プラットフォームに依存したサポートを行うため、プラットフォームを識別しなければならないからです。

最新バージョンの config.guessおよび config.subを、http://savannah.gnu.org/の‘config’プロジェクトから入手できます。以下は入手するためのコマンドです

```
$ wget -O config.guess 'http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.guess;
$ wget -O config.sub 'http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.sub;hb=HE
```

バージョンは最新ではありませんが、GNU automakeと GNU libtoolパッケージにも含まれています。

config.guessおよび config.subは通常、配布物のトップレベルに配置されます。しかし、他の設定ファイル (install-sh、ltconfig、ltmain.sh、missingなど) と同様に、サブディレクトリーに配置することもできます。ファイルを移動すること以外に必要なのは、configure.acに以下の行を追加することです。

```
AC_CONFIG_AUX_DIR([subdir])
```

13.4.7 トップレベルの mkinstalldirs

初期のバージョンの GNU gettext では、配布物に GNU mkinstalldirsスクリプトを追加する必要がありました。これは今では必要ありません。使用している automake が、automake 1.9 以降であれば削除することができます。

13.4.8 トップレベルの aclocal.m4

配布物に aclocal.m4ファイルがない場合、1 番単純なのは、GNU gettextのサブディレクトリー m4/の codeset.m4、fcntl-o.m4、gettext.m4、glibc2.m4、glibc21.m4、iconv.m4、intdiv0.m4、intl.m4、intldir.m4、intlmacosx.m4、intmax.m4、inttypes_h.m4、inttypes-pri.m4、lcmessage.m4、lib-ld.m4、lib-link.m4、lib-prefix.m4、lock.m4、longlong.m4、nls.m4、po.m4、printf-posix.m4、progtest.m4、size_max.m4、stdint_h.m4、threadlib.m4、uintmax_t.m4、visibility.m4、wchar_t.m4、wint_t.m4、xsize.m4を 1 つのファイルに結合する方法です。intl/ディレクトリーを作成しなかった場合、結合する必要があるのは gettext.m4、iconv.m4、lib-ld.m4、lib-link.m4、lib-prefix.m4、nls.m4、po.m4、progtest.m4だけです。

GNU automake 1.8 以降を使っていない場合は、もっと新しい automake の配布物から、上記のファイルに mkdirp.m4ファイルを追加する必要があります。

すでに aclocal.m4ファイルがある場合は、前述のマクロファイルを既存の aclocal.m4にマージする必要があります。以前にリリースされた GNU gettextからアップグレードしたようなときは、ほとんどの場合マクロ (AM_GNU_GETTEXT、...) を置き換える必要があることに注意してください。なぜならそれらのマクロは、GNU gettextがリリースされるときは通常、少し変更されるからです。これらの内容は、わたしたちが "奇妙" なシステムに出会う度に増えていくかもしれません。

GNU automake 1.5 以降を使用している場合には、マクロファイルを m4/というサブディレクトリーに配置して、以下の行を追加すれば充分です。

```
ACLOCAL_AMFLAGS = -I m4
```

上記のような行を、トップレベルの Makefile.amに追加してください。

GNU automake 1.10 以降を使用している場合は、さらに簡単になります。以下の行を追加してください

```
ACLOCAL_AMFLAGS = --install -I m4
```

上記のような行を、トップレベルの Makefile.amに追加して、'aclocal --install -I m4'を実行します。これは aclocal.m4を更新する前に、必要なファイルを自動的に m4/サブディレクトリーに追加します。

これらのマクロはインターナショナリゼーションのサポート機能と関連情報をチェックします。1 度うまく安定させられれば、多分これらのマクロを、標準の Autoconf に統合できるでしょう。なぜなら、これらの断片的な m4コードは、GNU gettextを使うプロジェクトでは同一だからです。

13.4.9 トップレベルの acconfig.h

初期のバージョンの GNU gettextでは、acconfig.hファイル内で ENABLE_NLS、HAVE_GETTEXT and HAVE_LC_MESSAGES、HAVE_STPCPY、PACKAGE and VERSIONを定義することが要求されました。これは今では必要ないので、パッケージが intl/ディレクトリーから独自に使用していなければ、削除することができます。

13.4.10 トップレベルの config.h.in

configureにより定義される C マクロを保持するインクルードファイルのテンプレートを、通常は config.h.inと呼び、手動または自動で保守されるかもしれません。

gettextizeが intl/ディレクトリーを作成している場合、ファイル名は config.h.inで、トップレベルになければなりません。gettextizeの '--intl'オプションを指定しないで、intl/ディレクトリーを作成しなかったときは、ファイル名と場所は自由に選ぶことができます。

プログラム 'autoheader'を使って自動的に保守されている場合、なにも行う必要はありません。これは特に GNU automakeを使っているケースです。

手動で保守していて gettextizeが intl/ディレクトリーを作成している場合は、'autoheader' を使うように変更するべきです。intl/ディレクトリーのために追加する C マクロのリストは、手動で保守するには長すぎます。そして、このリストは GNU gettextのバージョンが異なることにより変化するのです。

手動で保守している場合で、gettextizeを '--intl'オプションなしで呼び出したために、intl/ディレクトリーが作成されていないなら、config.h.inに以下の行を追加して"逃げる"ことができます:

```
/* Define to 1 if translation of program messages to the user's
   native language is requested. */
#undef ENABLE_NLS
```

13.4.11 トップレベルの Makefile.in

以下は、トップレベルにあるメインの Makefile.inファイルにたいして必要な変更です。

1. ゴール 'dist:'が正常に動作 (以降で説明します) するように、Makefile.inの最初の部分に、以下の行を追加します:

```
PACKAGE = @PACKAGE@
VERSION = @VERSION@
```

2. 定義 DISTFILESに、ファイル ABOUT-NLSを追加して、このファイルが配布されるように変更します。

3. どのサブディレクトリーの Makefile.inを処理する場合でも、サブディレクトリー 'intl'および 'po'も処理するようにしてください。Makefiles内の特別なルールは、インターナショナリゼーションが必要ない場合を処理するためのものです。

   Makefiles を使用している場合、それが automake により作成されたものか、手入力されたものかにかかわらず、GNU のコーディング規約にしたがうように注意してください。新しいサブディレクトリーを処理しなければならないために影響を受けるゴールには、'installdirs'、'install'、'uninstall'、'clean'、'distclean'が含まれます。

   以下は標準的な処理順の例です。この例ではゴール 'dist:'のために使用される SUBDIRSを、Makefile.inの中で定義しています。

```
SUBDIRS = doc intl lib src po
```

   'make'の調整では、ヘッダーファイル libintl.hを使うコードが含まれるディレクトリーより、intlディレクトリーが前にくるように注意してください。intl の前に libと srcの前に intlがあるのは、これが理由です。

4. デリケートなポイントは、intl/Makefileと po/Makefileの両方のゴール 'dist:'が、メインの Makefileによって、後から正しいディレクトリーにセットアップされる点です。以下はゴール 'dist:'がどのようなものかという例です:

```
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)
```

GNU automakeを使っているときは、Makefile.amから Makefile.inが自動的に生成されますが、Makefile.amに必要な修正は、'gettextize'の実行よって修正済みであることに注意してください。

13.4.12 src/内の Makefile.in

メインの Makefile.inで行ったいくつかの修正は、あなたのパッケージソースの Makefile.in(ここでは src/サブディレクトリーにあると仮定します) でも必要です。以下は、src/Makefile.in内で必要な修正のすべてです:

1. ゴール 'dist:'を考慮して、src/Makefile.inの先頭の部分に以下の行が必要になります:

   ```
   PACKAGE = @PACKAGE@
   VERSION = @VERSION@
   ```

2. まだ定義されていなければ、top_srcdirを定義する必要があります。これは cppのインクルードファイルのための定義で、以下の行を追加するだけです:

   ```
   top_srcdir = @top_srcdir@
   ```

3. 後ですべての Makefile.inで、一様なのゴール 'dist:'とするために、subdirを 'src'と定義したいと思うかもしれません。以下は、このゴール 'dist:'のために必要な定義です:

   ```
   subdir = src
   ```

4. 以下のようにプログラムの main関数は通常、bindtextdomain(Section 4.2 [Triggering], page 19 を参照してください) を呼び出します:

   ```
   bindtextdomain (PACKAGE, LOCALEDIR);
   textdomain (PACKAGE);
   ```

   プログラムに LOCALEDIR を知らせるために、以下の行を Makefile.inに追加します (Autoconf のバージョン 2.60 以降の場合):

   ```
   datadir = @datadir@
   datarootdir= @datarootdir@
   localedir = @localedir@
   DEFS = -DLOCALEDIR=\"$(localedir)\" @DEFS@
   ```

   Autoconf のバージョンが 2.60 より古い場合は、以下の行を追加します:

   ```
   datadir = @datadir@
   localedir = $(datadir)/locale
   DEFS = -DLOCALEDIR=\"$(localedir)\" @DEFS@
   ```

   @datadir@のデフォルトは '$(prefix)/share'なので、$(localedir)のデフォルトは、'$(prefix)/share/locale'になることに注意してください。

5. あなたは最後のリンクで、ライブラリーとして@LIBINTL@または@LTLIBINTL@が使われることを保証する必要があります。@LIBINTL@は libtoolなしで使用され、@LTLIBINTL@は libtoolとともに使用されます。これを達成するには、以下のようにこれらを LIBSで管理します:

   ```
   LIBS = @LIBINTL@ @LIBS@
   ```

   GNU gettextでインターナショナライズされたパッケージには、ヘルパー関数を含むライブラリーを、ディレクトリー lib/にビルドするものがたくさんあります (少なくとも GNU gettextライブラリー自身が必要とするいくつかの関数が必要です)。しかし lib/の中の関数のいくつかは、ユーザーに翻訳が必要なメッセージをあたえる関数です。これに注意してサポートのためのライブラリー (libsupport.aとしましょう) を、上記の例の@LIBINTL@と@LIBS@の前に配置します:

   ```
   LIBS = ../lib/libsupport.a @LIBINTL@ @LIBS@
   ```

6. あなたは、あらゆる状況において、ディレクトリー intl/が、C プリプロセッサーに検索されることも保証する必要があります。そのためには、'-I../intl'と '-I$(top_srcdir)/intl'の両方が C コンパイラーに与えられるように管理する必要があります。

7. ゴール‘dist:’は、他のものと一致している必要があります。以下はそのための合理的な定義です:

```
distdir = ../$(PACKAGE)-$(VERSION)/$(subdir)
dist: Makefile $(DISTFILES)
for file in $(DISTFILES); do \
  ln $$file $(distdir) 2>/dev/null || cp -p $$file $(distdir) || exit 1; \
done
```

GNU automakeを使用している場合、Makefile.inは Makefile.amから自動的に生成されるので、最初の 3 つと最後の変更は必要ないことに注意してください。Makefile.amに必要な変更は以下になります:

1. プログラムに LOCALEDIR を知らせるために、特定のモジュールにたいしては以下の行を:

```
<module>_CPPFLAGS = -DLOCALEDIR=\"$(localedir)\"
```

またはコンパイル単位については以下のように Makefile.amに記述します。

```
AM_CPPFLAGS = -DLOCALEDIR=\"$(localedir)\"
```

これはすべてのモジュール、またはコンパイル単位のためのものです。さらに Autoconf のバージョンが 2.60 より古いものを使用している場合、以下の行を追加して‘localedir’を定義します:

```
localedir = $(datadir)/locale
```

2. 最後のリンクが@LIBINTL@または@LTLIBINTL@を使うことを保証するために、以下を Makefile.amに追加します:

```
<program>_LDADD = @LIBINTL@
```

特定のプログラムごとには上記のように記述します。

```
LDADD = @LIBINTL@
```

これはすべてのプログラムの場合です。プログラムのリンクに libtoolを使うときは、プログラム用に@LIBINTL@ではなく、@LTLIBINTL@を使う必要があることを忘れないでください。

3. intl/ディレクトリーがあり、その内容が gettextizeにより作成されたものである場合は、以下のような行を Makefile.amに追加して、すべての状況において、C プリプロセッサーがインクルードファイルをそこから検索することを保証するようにしてください:

```
AM_CPPFLAGS = -I../intl -I$(top_srcdir)/intl
```

13.4.13 lib/内の gettext.h

GNU gettextにより提供される、パッケージのインターナショナリゼーションはオプションであり、2 つの状況でオフに切り替えることが考えられます:

- インストーラーで、‘./configure --disable-nls’が指定されたとき。ブートディスク用のユーティリティーをビルドするときのように、これは機能よりも小さいバイナリーを生成する方が重要な場合に有用かもしれません。これは、3.0 より古いバージョンの GCC で、コード品質にたいする C コンパイラーの特定の警告を取得するためにも有用でしょう。
- パッケージにサブディレクトリー intl/が含まれておらず、libintl.h ヘッダー (および、それに関連付けられた libintl ライブラリー) がまだシステムにインストールされていない場合は、コンパイルエラーよりもインターナショナリゼーションのサポートなしでパッケージをビルドする方が望ましいでしょう。

C プリプロセッサーのマクロは、これら 2 つのケースを検知するのに使用できます。通常、libintl.hが見つかって、明示的に利用不可されていなければ、autoconf が設定ファイルを生成するときに、ENABLE_NLSマクロが 1 に定義されます。しかし上記以外の状況では、このマクロは定義されず、それゆえ C では 0 に評価されます。

gettext.hは、ENABLE_NLSマクロにもとづいて<libintl.h>を使用する、便利なヘッダーファイルです。ENABLE_NLSがセットされていると、<libintl.h>がインクルードされ、セットされていない場合は libintl.h 関数のために代用の no-op(訳注: no-op = no operation = 何もしない) が定義されます。わたしたちは直接<libintl.h>を使うのではなく、"gettext.h"を使うことを推奨します。そうすれば古いシステムへの可搬性が保証され、もし望むならインストーラーでインターナショナリゼーションをオフにできます。

```
#include "gettext.h"
```

C のソースコードは下記の行を、上記のように書き換えます (訳注: 下が修正前で、上が修正後です。通常とは逆の順序で説明しているので間違えないでください)。

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

gettext.hの場所は通常、補助のインクルードファイルを含んだディレクトリーです。多くの GNU パッケージには、ヘルパー関数を含む lib/ディレクトリーがあるので、gettext.hはそこに配置すればよいでしょう。他のパッケージでは、srcディレクトリーに配置することができます。

gettext.hをパブリックな場所にインストールしないでください。このファイルを必要とするすべてのパッケージは、パッケージ自身にそのファイルのコピーが含まれているからです。

13.5 configure.ac内での autoconf マクロの使用

GNU gettextは、パッケージの configure.ac(または configure.in) で使用されるマクロをインストールします。詳細については、Section “Introduction” in *The Autoconf Manual* を参照してください。その中でも主要なマクロは、もちろん AM_GNU_GETTEXTです。

13.5.1 gettext.m4内の AM_GNU_GETTEXT

AM_GNU_GETTEXTマクロは、C ライブラリーおよび libintlライブラリー (どちらも共有または静的なライブラリーをサポートしています) に分割された GNU gettext の関数ファミリー、またはパッケージの intl/ディレクトリーをテストします。このマクロは、ビルド用にパッケージの po/ディレクトリーを準備するのに、AM_PO_SUBDIRSも呼び出します。

AM_GNU_GETTEXTは、オプションの引数を 3 つ指定でき、一般的な書式は以下のようになります

```
AM_GNU_GETTEXT([intlsymbol], [needsymbol], [intldir])
```

*intlsymbol*には、‘external’または ‘no-libtool’が指定できます。デフォルト (指定されなかったとき、または空のとき) は、‘no-libtool’です。intl/ディレクトリーのないパッケージでは、intlsymbol に ‘ external ’ を指定する必要があります。intl/ ディレクトリーのあるパッケージでは、*intlsymbol*に ‘no-libtool’を指定することもできるし、‘external’を指定して、他の場所でマクロ AM_GNU_GETTEXT_INTL_SUBDIRを使用することにより、それをオーバーライドすることもできます。この intl/の実体を指定する 2 つの方法は、同じことを行います。どちらもビルド時には、静的なライブラリー$(top_builddir)/intl/libintl.aを作成します。

*needsymbol*に ‘need-ngettext’が指定されると、ngettext()をもたない (libc または libintl の)GNU gettext 実装は無視されます。*needsymbol*に ‘need-formatstring-macros’が指定されると、ISO C 99 <inttypes.h>書式文字列マクロをサポートしない GNU gettext 実装は無視されます。*needsymbol*だけを指定することもできます。他の場所で AM_GNU_GETTEXT_NEEDを指定することでも、これらの指定を満たすことはできます。1 つ以上指定したときは、もっとも強い指定が使用されるか、AM_GNU_GETTEXT_NEEDマクロを複数回呼び出します。これらの指定は、‘need-formatstring-macros’が ‘need-ngettext’を含むような階層になっています。

*intldir*は、intl を探すのに使用されます。空の場合は、‘$(top_builddir)/intl/’という値が使用されます。

AM_GNU_GETTEXTマクロは、GNU gettext が利用可能で、使用できるかどうかを決定するマクロです。利用できる場合は、変数 USE_NLSに 'yes'をセットし、これは autoconf が生成する設定ファイル (通常は config.hというファイル) の ENABLE_NLSに 1 を定義し、Makefile で使用される変数 LIBINTLと LTLIBINTLにリンカーオプションをセットし (LIBINTLは libtool なしのときで、LTLIBINTLは libtool を使用するとき)、必要なときは CPPFLAGSのオプションに '-I'を追加し、利用できない場合は USE_NLSに 'no'をセットし、LIBINTLと LTLIBINTLを空にセットして、CPPFLAGS を変更しません。

AM_GNU_GETTEXTが対処する複雑さは、以下のようなものです:

- いくつかのオペレーティングシステムは、C ライブラリー (例: glibc) に gettextをもちます。GNU libintlは、GNU gettextの一部としてインストールされたのかもしれません。
- GNU libintlがインストールされていて、検索パス (インクルードファイルの検索パスは CPPFLAGS、ライブラリーの検索パスは LDFLAGS) にあるが、必要ない場合。
- glibc をのぞく、GNU の mo ファイルを取り扱えないオペレーティングシステムのネイティブの gettextは、必要な locale 依存の機能をもたず、カタログのテキストのエンコーディングから、ユーザーの locale のエンコーディングにメッセージを変換できません。
- GNU libintlがインストールされていて、実行時ライブラリーの検索パスにあるが、必要ない場合。LD_LIBRARY_PATHのような環境変数による設定を無視するために、このマクロは適切な実行時の検索パスオプションを、変数 LIBINTLおよび LTLIBINTLに追加します。これはほとんどシステムで動作しますが、SCO のように共有ライブラリーに制限のあるいくつかのオペレーティングシステムではうまく動作しません。
- GNU libintlは、POSIX/XSI の iconvに依存します。このマクロは iconv を使うために必要なリンカーオプションをチェックして、変数 LIBINTLおよび LTLIBINTLに追加します。

13.5.2 gettext.m4内の AM_GNU_GETTEXT_VERSION

AM_GNU_GETTEXT_VERSIONマクロは、パッケージで使用する GNU gettext インフラストラクチャーのバージョン番号を宣言します。

このマクロの使用はオプションで、これを使用するプログラムは autopointだけです (Section 13.6 [CVS Issues], page 150 を参照してください)。

13.5.3 gettext.m4内の AM_GNU_GETTEXT_NEED

AM_GNU_GETTEXT_NEEDマクロは、GNU gettext の実装に関する制約を宣言するもので、構文は以下のようになります

```
AM_GNU_GETTEXT_NEED([needsymbol])
```

*needsymbol*に 'need-ngettext'を指定した場合、ngettext()関数をもたない、(libc または libintl の)GNU gettext 実装は無視されます。*needsymbol*に 'need-formatstring-macros'を指定した場合、ISO C 99 <inttypes.h>の書式文字列マクロをサポートしない GNU gettext 実装は無視されます。

AM_GNU_GETTEXTの 2 番目のオプション引数も考慮されます。

AM_GNU_GETTEXT_NEED呼び出しは、AM_GNU_GETTEXT呼び出しの前後どちらでもよく、順番は関係ありません。

13.5.4 intldir.m4内の AM_GNU_GETTEXT_INTL_SUBDIR

AM_GNU_GETTEXT_INTL_SUBDIRマクロは、AM_GNU_GETTEXTの最初の引数に 'external'を指定して呼び出した場合でも、ビルドのために intl/サブディレクトリーも参照します。

AM_GNU_GETTEXT_INTL_SUBDIR呼び出しは、AM_GNU_GETTEXT呼び出しの前後どちらでもよく、順番は関係ありません。

このマクロは GNU automake 1.10 以降、または GNU autoconf 2.61 以降で使用できます。

13.5.5 po.m4内の AM_PO_SUBDIRS

このマクロは C、C++、Objective C 以外のプログラム言語 (例: sh、Python、Lisp) による、インターナショナライズされたプログラムで使う必要があります。PO ファイルによるローカリゼーションをサポートするプログラム言語のリストは、Chapter 15 [Programming Languages], page 154を参照してください。

AM_PO_SUBDIRSマクロは、インターナショナリゼーションを使う必要があるかを決定します。使う必要がある場合には USE_NLS変数に 'yes'をセットし、必要ない場合には 'no'をセットします。このマクロは、各 po/ディレクトリーの Makefile の変数にたいする適切な値も決定します。

13.5.6 po.m4内の AM_XGETTEXT_OPTION

AM_XGETTEXT_OPTIONマクロは、パッケージの po/ディレクトリーでの xgettext呼び出しで使用する、コマンドラインオプションを登録するマクロです。

たとえば、'error_at_line'という関数を定義しているソースファイルがあり、その関数の 5 番目の引数には、書式文字列を指定する場合には、以下のように使うことができます

```
AM_XGETTEXT_OPTION([--flag=error_at_line:5:c-format])
```

これは、この関数の 5 番目の引数にたいする 'gettext'呼び出しにたいして、これを翻訳可能な 'c-format'の文字列だとマークするよう、xgettextに指示します。

xgettextに指定できるオプションのリストは、Section 5.1 [xgettext Invocation], page 33を参照してください。

このマクロの使用は、po/Makevarsの中の 'XGETTEXT_OPTIONS'変数の代用となります。

13.5.7 iconv.m4内の AM_ICONV

AM_ICONVマクロは、C ライブラリー (または iconvライブラリーに分離された)POSIX/XSI iconv関数が提供されているかテストするマクロです。もし見つかったときは am_cv_func_iconv 変数に 'yes'をセットし、autoconf が生成する設定ファイル (通常は config.hというファイル) の HAVE_ICONVに 1 を定義し、iconv()の 2 番目の引数の型が 'const char **'または 'char **'で定義さされているかにより、ICONV_CONSTに 'const'または空を定義し、Makefile の中で使用されるリンカーオプションの変数 LIBICONVおよび LTLIBICONVをセット (LIBICONVは libtool なしのとき、LTLIBICONVは libtool ありのとき)、必要なら CPPFLAGSのオプションに '-I'を追加します。見つからなかったときは、LIBICONVおよび LTLIBICONVに空をセットして、CPPFLAGSを変更しません。

AM_ICONVが対処する複雑さは、以下のようなものです:

- C ライブラリーに iconvのあるオペレーティングシステムとしては、たとえば glibc があり、ライブラリー libiconvに分割されているシステムには、たとえば OSF/1、FreeBSD があります。種類の如何にかかわらず、GNU libiconvがインストールされたオペレーティングシステムの場合、オペレーティングシステムのネイティブの iconvのかわりに使用されます。
- GNU libiconvがインストールされていて、検索パス (インクルードファイルの検索パス CPPFLAGS、ライブラリーの検索パス LDFLAGS) にあるが、必要ない場合。
- GNU libiconvは、いくつかのオペレーティングシステムのネイティブの iconvにたいしてバイナリーの非互換があります (例: FreeBSD)。適合していない iconv.hおよび libiconv.so の使用は、プログラムのクラッシュを招きます。

- GNU libiconvがインストールされていて、実行時ライブラリーの検索パスにあるが、必要ない場合。LD_LIBRARY_PATHのような環境変数による設定を無視するために、このマクロは適切なランタイムのサーチパスオプションを、変数LIBICONVに追加します。これはほとんどシステムで動作しますが、SCO のように共有ライブラリーに制限のあるいくつかのオペレーティングシステムではうまく動作しません。

gettext.m4が依存しているので、iconv.m4は GNU gettext の一部として配布されます。

13.6 CVS による統合

多くのプロジェクトでは分散開発におけるバージョンコントロールとソースのバックアップに CVS を使用しています。このセクションでは cvs、gettextize、autopoint、autoconfの使用をどのように管理するかについてのアドバイスを与えます

13.6.1 分散開発におけるバージョンミスマッチを避ける

複数の開発者による CVS を使ったプロジェクト開発では、gettextの新しいバージョンにアップグレードしたいと望む一人の開発者が、gettextizeを実行して Section 13.4 [Adjusting Files], page 139 にリストした変更をほどこした後に、その変更を CVS にコミットするようなことが時折あります。

プロジェクトのすべての開発者が、パッケージの GNU gettextに、同じバージョンのものを使用することを強く推奨します。別の言い方をすると、gettextizeを実行したら、開発者はプロジェクト全体と同じ方法で必要な変更をほどこして CVS にコミットする必要があります。さもないと以下のような損傷が発生します:

- 開発者の間での、明らかなバージョンの不一致。configure.acおよび configure.in、Makefile.amおよび Makefile.in内の gettextに関係する特定の箇所は、gettextのバージョンに依存しており、異なるバージョンの gettextによる基礎的なファイルの使用により、容易にビルドエラーが発生します。
- 明らかでないバージョンの不一致。このようなバージョン不一致は、開発者が発見できないようなパッケージの動作不良も招きます。明らかでないバージョンの不一致による最悪のケースは、パッケージのインターナショナリゼーションが動作しないケースです。
- リリースのリスク。すべての開発者は、パッケージにたいして一定のテストを暗黙に行います。このテストはリリースの数週間から数日前にかけて重要です。もしも誰かが他の開発者とは異なるバージョンの GNU gettextを使ってリリース用の tar ファイルを作ったら、その配布物は同じバージョンの gettextを使ってテストされたものに比べてテストされていないことになり、たとえばそれがプラットフォームに固有の未知のバグにもなり得るのです。

13.6.2 CVS バージョンコントロールの配下に置くファイル

CVS レポジトリーのコンテキストで作成されるファイル、たとえば configure.acにより生成される configure、*parser*.yから生成される *parser*.c、gettextizeや autopointにより自動インストールされる po/Makefile.in.inのようなファイルを取り扱うには、基本的に 3 つの方法があります。

1. 生成されるすべてのファイルを、常にレポジトリーにコミットする。
2. 生成されるすべてのファイルを、時々(たとえば毎リリースごとに) レポジトリーにコミットする。
3. 生成されるファイルを、レポジトリーにコミットされない。

これら 3 つの方法には、それぞれ異なる利点と欠点があります。

1. 1 番目の方法の利点は、誰でも CVS からその時点で動作するビルドをチェックアウトできる点です。欠点は以下のとおりです:
   1a メンテナーによる頻繁な"cvs commit"操作が必要です。
   1b レポジトリーサイズの増加が早くなります。
2. 2 番目の方法の利点は、誰もがチェックアウトでき、通常は"./configure; make"は動作します。欠点は以下のとおりです:
   2a レポジトリーからチェックアウトした人の PATH に GNU `automake`、GNU `autoconf`、GNU `m4`のようなツールがインストールされている必要があり、ときには特定のバージョンが必要になる。
   2b リリース版が作成されて生成されるファイルもコミットされた後に他の開発者が"cvs update"を行うと、生成されるファイルで競合が発生する点。この競合は簡単に解決できますが、煩わしいものです。
3. 3 番目の方法の利点はメンテナーの作業負荷が軽減されることです。欠点はレポジトリーからチェックアウトした人の PATH に GNU `automake`, GNU `autoconf`, GNU `m4`のようなツールがインストールされている必要があるだけではなく、"./configure; make"をできるようになる前に、パッケージ固有の pre-build(ビルド前) ステップが必要になることです。

1 番目と 2 番目の方法では、変更されたファイルや、`gettextize`呼び出しにより生成・更新されたファイルは、CVS にコミットする必要があります。

3 番目の方法では、`gettextize`が"コピー"するすべてのファイルを、CVS レポジトリーから除外できます。そのかわりに `configure.ac`(または `configure.in`) を、以下のような形式で記述します

```
AM_GNU_GETTEXT_VERSION(0.18.3)
```

さらにパッケージの pre-build スクリプトに‘`autopoint`’呼び出しを追加します。CVS からチェックアウトする人は誰でも、この‘`autopoint`’呼び出しにより CVS から除外された `gettext`の基礎的なファイルが適切な場所にコピーされます。

`AM_GNU_GETTEXT_VERSION`の引数に使用されているバージョン番号は、パッケージが使いたい `gettext`インフラストラクチャーのバージョン番号です。これは‘`autopoint`’プログラムの最小のバージョン番号でもあります。もし `AM_GNU_GETTEXT_VERSION(0.11.5)`と記述した場合、開発者は 0.11.5 以上のバージョンを使用でき、すべての開発者のビルドがバージョン 0.11.5 のインフラストラクチャーで動作します。メンテナーがパッケージにたいしてバージョン 0.12.1 を指定して gettextize を実行したとき、`AM_GNU_GETTEXT_VERSION(0.11.5)`は `AM_GNU_GETTEXT_VERSION(0.12.1)`に変更され、今後 CVS を使う開発者は GNU `gettext` 0.12.1 以降をインストールする必要があります。

13.6.3 `autopoint`プログラムの呼び出し

```
autopoint [option]...
```

`autopoint`は、gettext の基礎となるファイルを、ソースパッケージにコピーするプログラムです。このプログラムは `AM_GNU_GETTEXT_VERSION(`*`version`*`)`の形式で呼び出されるマクロにより、パッケージの `configure.in`(または `configure.ac`) ファイルからパッケージで使用される gettext のバージョンを抽出して、そのバージョンに該当する基礎となるファイルをパッケージにコピーします。

13.6.3.1 Options

‘`-f`’
‘`--force`’ 既存のファイルを強制的に上書きします。

‘-n’
‘--dry-run’
変更を出力しますが、処理は行いません。普通に autopointを実行したことによるファイルのコピーはすべて抑止され、かわりに標準出力にリストが出力されます。

13.6.3.2 Informative output

‘--help’ このヘルプを表示して終了します。

‘--version’
バージョン情報を表示して終了します。

autopointは、GNU gettextのバージョン 0.10.35 から、現在の 0.18.3 までをサポートします。0.18.3 より新しいバージョンの gettextを使用してパッケージに autopointを適用するためには、少なくとも同じバージョンの GNU gettextをインストールする必要があります。

GNU automakeを使用するパッケージでの autopointの呼び出しは、aclocal呼び出しの後に行い、その後で autoconfおよび autoheaderを呼び出します。これは、autopointが aclocal.m4を作成するために、autopointがいくつかの autoconf マクロをインストールするのが理由です。この aclocal.m4 は、autoconf によるパッケージの configure スクリプトを作成と、autoheader によるパッケージのインクルードファイルのテンプレート config.h.in を作成するために使用されます。このマクロファイルは aclocal.m4を作成するために aclocalにより使用され、aclocal.m4はパッケージの configureを作成するために aclocalにより使用され、インクルードファイルテンプレートであるパッケージの config.h.inを作成するために autoheaderにより使用されます。

‘autopoint’という名前は ‘auto-po-intl-m4’を省略したものです。このツールは主に po、intl、m4ディレクトリーのファイルをコピーします。

13.7 配布用 tarball の作成

GNU automakeを使うプロジェクトでは、配布用の tarball を作成する通常のコマンドは、‘make dist’または ‘make distcheck’で、これにより必要に応じて自動的に PO ファイルが更新されます。

GNU automakeを使用していない場合、メンテナーはこのような更新をリリースの前に行う必要があります:

```
$ ./configure
$ (cd po; make update-po)
$ make distclean
```

14 インストーラーと配布者の視点

デフォルトでは、内部的に GNU gettextをフルに使っているパッケージは、翻訳されたメッセージを使えるような方法でインストールされます。*configuration* 実行時には、これらのパッケージはホストシステムですでに GNU gettextの機能が提供されているかを、自動的に検出する必要があります。もし提供されていない場合、GNU gettextライブラリーが自動的に準備・使用されます。インストーラーは、configure 時のこの動作を変更するための特別なオプションを使うことができます。コマンド '`./configure --with-included-gettext`'により、そのシステムの gettextをバイパスしてかわりに同梱された GNU gettextを使用します。また '`./configure --disable-nls`'では、翻訳されたメッセージを利用しないプログラムを生成します。

インターナショナライズされたパッケージには通常、多くの *ll*`.po`ファイルがあります。翻訳が利用不可になっていなければ、パッケージのインストールによりそれらが利用可能になります。しかし configure に先立ち環境変数の LINGUASがセットされていると、インストールされる対象が制限されます。LINGUASはスペースで区切られた 2 文字のコードのリストで、利用できる言語を指定します。

15 その他のプログラミング言語

gettextが提供するものの大部分はC(これは暗黙でC++にも同様に適用できます) に焦点をあてていますが、それ以外にも他の多くのプログラム言語やスクリプト言語、その他のテキストデータ、たとえばGUIリソースやパッケージの説明にもgettextの手法を用いることができます。

15.1 言語実装者の視点

すべてのプログラム言語およびスクリプト言語は、gettextをサポートするのに適した文字列の表記をもっています。gettextをサポートするとは、以下のことを意味します:

1. 翻訳可能な文字列にたいする書式を言語に追加する必要があります。原則としてはgettextの関数呼び出しですが、省略した書式はインターナショナライズされたプログラムの可読性を向上する助けとなります。たとえばCでは`_("string")`、GNU awkでは`_"string"`という書式が使用されます。
2. 実行時のgettext呼び出し (または等価な処理) により、このような翻訳可能な文字列を評価するための用意をする必要があります。
3. 同様に、その言語でngettext、dcgettext、dcngettextの関数を利用可能にする必要があります。これらの関数が使用されることは少ないかもしれません。しかしngettextは正しくpluralを処理するために、そしてdcgettextとdcngettextはLC_TIMEやLC_MONETARYなどの、LC_MESSAGES以外のlocale関連の環境変数を処理する等の、特別の目的のために必要です。後者の関数についてはCのヘッダーファイル<locale.h>の定数LC_*が通常、環境変数や文字列で参照するため、その言語からも参照できるようにする必要があります。
4. その言語からtextdomain関数を利用可能にするか、TEXTDOMAINのような"魔法"の変数を用意するなどして、プログラマーがメッセージドメインを明示できるようにする必要があります。同様に、bindtextdomain関数のように、プログラマーがメッセージカタログをどこから検索するかを、明示できるようにする機能を提供する必要があります。
5. `setlocale (LC_ALL, "")`を、言語が実行されたスタートアップ時に呼び出すか、プログラマーが呼び出して処理できるようにするべきです。localeカテゴリーのLC_MESSAGESとLC_CTYPEがどちらもセットされていないときは、gettextはno-opとして振る舞うことを思い出してください。
6. プログラマーには、プログラムから翻訳可能な文字列をPOファイルに抽出する方法が必要です。GNU xgettextは、非常に多くの異なるプログラム言語をサポートするように拡張されています。どうすればサポートされるかについては、GNU gettextのメンテナーに連絡してください。文字列の抽出機能が、あなたの言語のパーサーに統合されれば、GNU xgettextをあなたの言語の文字列抽出機能のフロントエンドとすることができます。
7. 言語のライブラリーに、書式文字列と書式文字列の引数を位置番号や名前で示せるような機能をもたせる必要があります。これはいくつかの言語やメッセージでは、2つ以上の代替の引数にたいして、翻訳者はそれらの代替の引数を異なった順序で出力する必要があるからです。詳細は、Section 4.6 [c-format Flag], page 26 を参照してください。
8. 言語に2つ以上の実装があって、それらのすべてがgettextを実装している訳ではないにもかかわらず、それらの異なる実装間でプログラムに可搬性をもたせなければならないような場合には、no-i18nエミュレーションを提供する必要があります。これにより、実際に翻訳された文字列がなくても、あなたの実装向けに記述されたプログラムを他の実装で利用することができます。
9. プログラマーが翻訳文字列にマークを付与するタスクを助けるために、EmacsのPOモードが使用される場合があります (Section 4.5 [Marking], page 24 を参照してください)。遠慮なく

GNU gettextのメンテナーに連絡してください。そうすれば彼らが po-mode.elにあなたの言語にたいするサポートを追加することができます。

実装から考えると、可搬性と著作権の面において異なる効果をもつ、3 つのアプローチが利用できます。

- Chapter 13 [Maintainers], page 135 で説明している方法で、あなたのパッケージの GNU gettext用の intl/ディレクトリーを使って統合する方法があります。これにより、すべてのプラットフォームでインターナショナリゼーションが可能になります。この場合、パッケージは法的には GNU General Public License の下に配布されることに注意してください。そして GNU project はフリーソフトウェアの蓄積にたいするあなたの貢献を歓迎するでしょう。
- C ライブラリーの GNU gettext関数にたいしてリンクする方法があります。たとえば、autoconf が gettext()を ngettext()をテストしてこの状況を検知します。当面のところ、このテストは GNU システムでは成功しますが、他のシステムでは成功しません。また、厳密な著作権の制限はありません。
- GNU gettextの機能をエミュレート、もしくは再実装する方法があります。この方法には、完全な可搬性と著作権の制限がないという利点があります。しかし GNU gettextの機能 (環境変数 LANGUAGE、locale aliase のデータベース、自動的な文字コード変換、plural の処理のような機能) を再実装する必要があるという欠点もあります。

15.2 プログラマーの視点

プログラマーにとって、一般的な手続きは C 言語の場合と同じです。Emacs の PO モードによるマークづけは他の言語もサポートしており、GNU xgettextの文字列抽出も、ファイルの拡張子やコマンドラインのオプションで他の言語を識別できます。実行時の言語にしたがって実行されるために、setlocaleを必要としない言語もいくつかあります。

15.3 翻訳者の視点

翻訳者の作業は C 言語の場合と同じです。唯一の違いは書式文字列の翻訳で、彼女は書式文字列にたいする、その言語特有の位置引数を理解する必要があります。

15.3.1 C フォーマット文字列

C の書式文字列は POSIX(IEEE P1003.1 2001) のセクション XSH 3 fprintf() http://www.opengroup.org/onlinepubs/007904975/functions/fprintf.htmlで説明されています。また fprintf() の man http://www.linuxvalley.it/encyclopedia/ldp/manpage/man3/printf.3.php, http://informatik.fh-wuerzburg.de/student/i510/man/printf.html も参照してください。

以下のような、引数の位置を再指定する書式文字列があったとします

```
"Only %2$d bytes free on '%1$s'."
```

これは以下の文と同じ意味をあらわします

```
"'%s' has only %d bytes free."
```

これは POSIX/XSI の機能であり、ISO C 99 には明記されていませんが、翻訳者はこの再配置の機能を信頼することができます: ネイティブでは printf()や fprintf()などがこの機能をサポートしていないプラットフォームも存在しますが、libintl.a(または libintl.so) が再配置の関数を提供していて、<libintl.h>がこれらの再配置のための関数を自動的に有効化するからです。

Farsi(Persian) そして多分 Arabic のための特別な機能として、翻訳者は数値の書式指定子に 'I' フラグを挿入できます。この場合、たとえば"%d"は"%Id" に翻訳されます。このフラグを指定すると、GNU libcのあるシステムでは、ASCII での数字の出力が、locale カテゴリーの LC_CTYPEで定義された 'outdigits'により置き換えられます。他のシステムでは、gettext関数がフラグを取り除くため、何の影響もありません。

プログラマーはこのフラグを未翻訳の文字列に挿入するべきではないことに注意してください (文字列 *msgid*の書式指定のフラグに 'I'を挿入すると、glibc がないシステムで NLS が無効になっているときに、未定義の動作を招きます)。

15.3.2 Objective C フォーマット文字列

Objective C の書式文字列は、C の書式文字列と似ています。Objective C の場合は、追加の書式指定子として、実行時に引数を Object *タイプとして評価するための"%@"をサポートします。

15.3.3 Shell フォーマット文字列

Shell の書式文字列は GNU gettext と 'envsubst'プログラムにより、$*variable* または${*variable*}という形式で参照されるシェル変数がサポートされています。${*variable-default*}、${*variable*:-*default*}、${*variable=default*}、${*variable*:=*default*}、${*variable+replacement*}、${*variable*:+*replacement*}、${*variable*?*ignored*}、${*variable*:?*ignored*}で参照される、シェルスクリプト内だけで有効な形式はサポートされません。*variable*の名前には、半角の英数字か ASCII 文字のアンダースコアーしか含められません。また数字で開始することはできず、空も指定できません。そのような変数にたいする参照は無視されます。

15.3.4 Python フォーマット文字列

Python には 2 つの書式文字列があります。'python-format'としてラベルづけされた Python のビルトイン書式オペレーター%と、'str'オブジェクトの formatに適用できます。

Python の%書式文字列については Python Library reference / 2. Built-in Types, Exceptions and Functions / 2.2. Built-in Types / 2.2.6. Sequence Types / 2.2.6.2. String Formatting Operations で説明されています。http://www.python.org/doc/2.2.1/lib/typesseq-strings.htmlを参照してください。

Python カッコつき書式文字列 (Python brace format strings) は、PEP 3101 – Advanced String Formatting、http://www.python.org/dev/peps/pep-3101/で説明されています。

15.3.5 Lisp フォーマット文字列

Lisp の書式文字列は Common Lisp HyperSpec の chapter 22.3 Formatted Output、http://www.lisp.org/HyperSpec/Body/sec_22-3.htmlで説明されています。

15.3.6 Emacs Lisp フォーマット文字列

Emacs Lisp の書式文字列は Emacs Lisp reference の section Formatting Strings、http://www.gnu.org/manual/elisp-manual-21-2.8/html_chapter/elisp_4.html#SEC75に記述されています。バージョン 21 の XEmacs では、FSF Emacs がサポートしていない書式文字列内に番号付けられた引数指定をサポートすることに注意してください。

15.3.7 librep フォーマット文字列

librep の書式文字列は librep manual の sectionFormatted Output、`http://librep.sourceforge.net/librep-manual.html#Formatted%20Output`、`http://www.gwinnup.org/research/docs/librep.html#SEC122`に記述されています。

15.3.8 Scheme フォーマット文字列

Scheme の書式文字列は SLIB manual の section Format Specification に記述されています。

15.3.9 Smalltalk フォーマット文字列

Smalltalk の書式文字列は、GNU Smalltalk documentation の class `CharArray`、methods '`bindWith:`'と '`bindWithArguments:`'で説明されています。`http://www.gnu.org/software/smalltalk/gst-manual/gst_68.html#SEC238`を参照してください。要約すると、指定子は '`%`' で開始され、'`%`'か非 0 の数字 ('`1`' to '`9`') が後に続きます。

15.3.10 Java フォーマット文字列

Java の書式文字列は JDK documentation の class `java.text.MessageFormat`、`http://java.sun.com/j2se/1.4/docs/api/java/text/MessageFormat.html`で説明されています。ICU documentation、`http://oss.software.ibm.com/icu/apiref/classMessageFormat.html`も参照してください。

15.3.11 C#フォーマット文字列

C#の書式文字列は.NET documentation の class `System.String`と `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpConFormattingOverview.asp`で説明されています。

15.3.12 awk フォーマット文字列

awk の書式文字列は gawk documentation の section Printf、`http://www.gnu.org/manual/gawk/html_node/Printf.html#Printf`で説明されています。

15.3.13 Object Pascal フォーマット文字列

Object Pascal の書式文字列は documentation of the Free Pascal runtime library の section Format、`http://www.freepascal.org/docs-html/rtl/sysutils/format.html`で説明されています。

15.3.14 YCP フォーマット文字列

YCP の書式文字列は libycp documentation `file:/usr/share/doc/packages/libycp/YCP-builtins.html`で説明されています。要約すると、指定子は '`%`'で開始され、'`%`'か非 0 の数字 ('`1`'から '`9`') が続きます。

15.3.15 Tcl フォーマット文字列

Tcl の書式文字列は`format.n`の man、`http://www.scriptics.com/man/tcl8.3/TclCmd/format.htm`で説明されています。

15.3.16 Perl フォーマット文字列

Perl には 2 種類の書式文字列があります。'perl-format'とラベルつけされた Perl のビルトイン関数 printfと、'perl-brace-format'とラベルつけされた libintl-perl関数__xで利用することができます。

Perl の printfの書式文字列は、'man perlfunc'の sprintfの section で説明されています。

perl brace の書式文字列は CPAN のパッケージ libintl-perl の Locale::TextDomain(3pm) manpage で説明されています。要約すると、Perl format はカッコ ('{'と '}') の間に記述されたプレースホルダーを使用します。このプレースホルダーは簡単に識別できる構文でなければなりません。

15.3.17 PHP フォーマット文字列

PHP の書式文字列は、phpdoc/manual/function.sprintf.html、または http://www.php.net/manual/en/function.sprintf.phpで説明されています。

15.3.18 GCC internal フォーマット文字列

これらの書式文字列は、GCC のソース内で使用されるものです。このような書式文字列では、指定子は '%'で開始され、オプションのサイズ指定子 'l'、オプションフラグの '+'、他のオプションフラグとして '#'が続きます。終端させるための指定子は、'%'の場合はリテラルのパーセント記号、'c'の場合は文字、's'は文字列、'i'および 'd'は整数、'o', 'u', 'x'は符号なし整数、'.*s'は前に幅指定をともなう文字列、'H''location_t *'型のポインター、'D'は一般的な宣言、'F'は関数宣言、'T'は型、'A'は関数の引数、'C'は tree code、'E'は式、'L'はプログラム言語、'O'はバイナリー演算子、'P'は関数パラメーター、'Q'は assignment operator、'V'は const/volatile qualifier を表します。

15.3.19 GFC internal フォーマット文字列

これらの書式文字列は GNU Fortran Compiler のソース内で使用されます。これは GCC ソースの Fortran 用フロントエンドです。このような書式文字列では、指定子は '%'で開始され、終端させるための指定子は、'%'はリテラルのパーセント記号、'C'は現在のソース位置、'L'はソースの位置、'c'は文字、's'は文字列、'i' and 'd'は整数、'u'は符号なし整数で、'i'、'd'、'u'の場合は前には、サイズ指定子 'l'が指定されている場合があります。

15.3.20 Qt フォーマット文字列

Qt の書式文字列は documentation of the QString class file:/usr/lib/qt-4.3.0/doc/html/qstring.htmlで説明されています。要約すると、指定子は '%'とその後ろの数字から成り立ちます。1 つの書式文字列内に同じ指定子を 2 回以上使うことはできません。

15.3.21 Qt フォーマット文字列

Qt の書式文字列は documentation of the QObject::tr method file:/usr/lib/qt-4.3.0/doc/html/qobject.htmlで説明されています。要約すると、使用できる指定子は '%n'だけです。

15.3.22 KDE フォーマット文字列

KDE 4 の書式文字列は次のように定義されています: 指定子は '%'とその後ろの非 0 の 10 進数からなります。書式文字列内に '%n'がある場合、それ以外の '%1'、...、'%(n-1)'も存在しなければなりません。

15.3.23 Boost フォーマット文字列

Boost の書式文字列は、http://www.boost.org/libs/format/doc/format.htmlの documentation of the boost::format class で説明されています。要約すると、指定子は '%1$+5d'

のような C の書式文字列と同じ文法、または ‘%|1$+5d|’や ‘%|1$+5|’のように垂直バー囲まれたものか、‘%1%’のように単にパーセント記号で囲まれたものです。

15.3.24 Lua フォーマット文字列

Lua の書式文字列は、Lua reference manual の section String Manipulation で説明されています。http://www.lua.org/manual/5.1/manual.html#pdf-string.formatを参照してください。

15.3.25 Java Script フォーマット文字列

JavaScript 仕様には書式文字列が定義されていないとはいえ、多くの JavaScript 実装は printf-like な関数を提供します。xgettextは、Gjs、Seed、Node.JS を含む著名な JavaScript 実装で使用される、一般的な書式文字列のセットを理解します。そのような書式文字列では、書式指定は ‘%’で開始され、以下の指定子で終端されます: ‘%’はリテラルのパーセント、‘c’は文字、‘s’は文字列、‘b’、‘d’、‘o’、‘x’、‘X’は整数、‘f’は浮動小数点数、‘j’は JSON オブジェクトを表します。

15.4 メンテナーの視点

メンテナーにとっては、一般的な手続きで C 言語の場合での方法と異なるのは、以下の 2 つです。

- GNU gettext を使用しない言語では、intl/ディレクトリーは必要ないので省略できます。これはメンテナーが gettextizeプログラムを ‘--intl’オプションなしで呼び出すこと、そして autoconf マクロ AM_GNU_GETTEXTを ‘AM_GNU_GETTEXT([external])’と呼び出すことを意味します。
- もし使用されているプログラム言語が 1 つだけの場合には、po/Makevars(Section 13.4.3 [po/Makevars], page 140 を参照してください) の変数 XGETTEXT_OPTIONSは、xgettextのその言語向け特有のオプションにする必要があります。パッケージが gettextのサポートを複数のプログラム言語を使用している場合、po/Makefile.in.in内の POT ファイルの構築ルールを変更する必要があります。この場合は、1 つのプログラム言語ごとに、言語に適したオプションで xgettext呼び出しを行い、msgcatにより結果ファイルを結合する方法をお勧めします。

15.5 個別のプログラミング言語

15.5.1 C、C++、Objective

RPM gcc、gpp、gobjc、glibc、gettext

ファイル拡張子
C: c、h
C++: C、c++、cc、cxx、cpp、hpp
Objective C: m

文字列構文 "abc"

gettext の略記
_("abc")

gettext/ngettext 関数
gettext、dgettext、dcgettext、ngettext、dngettext、dcngettext

textdomain
textdomain関数

bindtextdomain
: `bindtextdomain`関数

setlocale　プログラマーは、`setlocale (LC_ALL, "")`を呼び出さなければなりません。

必要条件
```
#include <libintl.h>
#include <locale.h>
#define _(string) gettext (string)
```

GNU gettext の使用またはエミュレート
: 使用

抽出プログラム
: `xgettext -k_`

位置の書式　`fprintf "%2$d %1$d"`
C++は `autosprintf "%2$d %1$d"` (Section “Introduction” in *GNU autosprintf* を参照してください)

可搬性　autoconf(gettext.m4)、および#if ENABLE_NLS

po-mode でのマーキング
: yes

`examples`ディレクトリーの以下の例が利用できます: `hello-c`、`hello-c-gnome`、`hello-c++`、`hello-c++-qt`、`hello-c++-kde`、`hello-c++-gnome`、`hello-c++-wxwidgets`、`hello-objc`、`hello-objc-gnustep`、`hello-objc-gnome`

15.5.2 sh - シェルスクリプト

RPM　bash、gettext

ファイル拡張子
: `sh`

文字列構文　`"abc"`、`'abc'`、abc

gettext の略記
: `"‘gettext \"abc\"‘"`

gettext/ngettext 関数
: プログラム: `gettext`、`ngettext`
シェル関数: `eval_gettext`、`eval_ngettext`

textdomain
: 環境変数 `TEXTDOMAIN`

bindtextdomain
: 環境変数 `TEXTDOMAINDIR`

setlocale　自動

必要条件　`. gettext.sh`

GNU gettext の使用またはエミュレート
: 使用

抽出プログラム
: `xgettext`

位置の書式 —

可搬性 完全な可搬性がある

po-mode でのマーキング
: —

`examples`ディレクトリーの例 `hello-sh`が利用できます

15.5.2.1 インターナショナリゼーションのためにシェルスクリプトを準備する

インターナショナリゼーションにむけたシェルの準備は、概念的には Chapter 4 [Sources], page 19 で説明したステップと似ています。以下はシェルスクリプトのための具体的なステップです。

1. スクリプトの冒頭付近に、以下の行を挿入します。

   ```
   . gettext.sh
   ```

   `gettext.sh`は、関数 `eval_gettext`(Section 15.5.2.6 [eval_gettext Invocation], page 164 を参照してください) と、`eval_ngettext`(Section 15.5.2.7 [eval_ngettext Invocation], page 164 を参照してください) を提供するシェルの関数ライブラリーです。`gettext.sh`は、`PATH`に記述された検索パスに配置されている必要があります。

2. 環境変数 `TEXTDOMAIN`および `TEXTDOMAINDIR`を、set して export します。`TEXTDOMAIN`は通常はパッケージ名かプログラム名、`TEXTDOMAINDIR`は`$prefix/share/locale`に対応する絶対パスです。ここで`$prefix`はインストールした場所です。

   ```
   TEXTDOMAIN=@PACKAGE@
   export TEXTDOMAIN
   TEXTDOMAINDIR=@LOCALEDIR@
   export TEXTDOMAINDIR
   ```

3. Section 4.3 [Preparing Strings], page 20 で説明しているように、翻訳する文字列を準備します。

4. 翻訳可能な文字列を単純にするため、コマンドによる文字列の置き換え (("`...`"または `"$(...)"`)) や、デフォルト値が含まれる変数 (例：`${variable-default}`)、`$0`, `$1`, ... のような引数参照、一時的なシェル変数 (例: `$?`) を含めないようにします。こうすることにより、常に単純な local コードを再構築することができます。たとえば、

   ```
   echo "Usage: $0 [OPTION] FILE..."
   ```

   は以下のように書き換えます

   ```
   program_name=$0
   echo "Usage: $program_name [OPTION] FILE..."
   ```

   同様に、

   ```
   echo "Remaining files: `ls | wc -l`"
   ```

   は以下のように書き換えます

   ```
   filecount="`ls | wc -l`"
   echo "Remaining files: $filecount"
   ```

5. 個々の翻訳可能な文字列にたいして、出力コマンド '`echo`'や '`$echo`'を、'`gettext`'(文字列にシェル変数の参照が含まれない場合) や '`eval_gettext`'(シェル変数を参照する場合) に変更して、その後ろに引数を指定しない '`echo`'コマンドを記述します (これは行末の改行のためです)。plural を扱う場合は同様に、'`echo`'コマンドを '`ngettext`'または '`eval_ngettext`'の呼び出しに置き換えて、後に引数なしの '`echo`'コマンドを記述します。

この変更を行う際には、シェル変数への参照の前にある$記号をエスケープするためのバックスラッシュを追加する必要もあります。そうすれば‘eval_gettext’関数が変数が置き換える前の、翻訳可能な文字列を受けとることができます。例えば以下のような文字列で考えてみましょう

```
echo "Remaining files: $filecount"
```

は以下のように書き換えます

```
eval_gettext "Remaining files: \$filecount"; echo
```

出力コマンドが‘echo’ではないときも、バッククォートにより‘echo’を使うことができます。バッククォートの内側では、バックスラッシュを 2 重に指定しなければ効果がないことに注意しなければなりません (これはバッククォートすることにより、バックスラッシュが 1 階層分消費されるからです)。例として、‘echo’がエラーをシグナルするシェル関数だとすると、

```
error "file not found: $filename"
```

は最初に以下へ変換され

```
error "‘echo \"file not found: \$filename\"‘"
```

となり、その後以下のようになります

```
error "‘eval_gettext \"file not found: \\\$filename\"‘"
```

15.5.2.2 gettext.shの内容

gettext.shには GNU gettext のランタイム パッケージが含まれており、以下が提供されます:

- $echo 変数 echoには、最初の引数 (引数文字列内のバックスラッシュの解釈されません) と、改行を出力するコマンドがセットされます。
- eval_gettext Section 15.5.2.6 [eval_gettext Invocation], page 164 を参照してください。
- eval_ngettext Section 15.5.2.7 [eval_ngettext Invocation], page 164 を参照してください。

15.5.2.3 gettextプログラムの呼び出し

```
gettext [option] [[textdomain] msgid]
gettext [option] -s [msgid]...
```

gettextは、母国語に翻訳されたテキストメッセージを表示するプログラムです。

Arguments

‘-d *textdomain*’
‘--domain=*textdomain*’
: *textdomain*から、翻訳されたメッセージを取得します。*textdomain*は通常、パッケージか、プログラムやプログラムのモジュールと一致します。

‘-e’
: いくつかのエスケープシーケンスの展開を可能にします。これは echo プログラムややシェルのビルトインコマンドにたいする互換のためのオプションです。対象となるエスケープシーケンスは‘\a’、‘\b’、‘\c’、‘\f’、‘\n’、‘\r’、‘\t’、‘\v’、‘\\’、および‘\’と、その後の 3 桁以内の 8 進数です (これらは System V の‘echo’と同様に処理されます)。

‘-E’
: このオプションは‘echo’プログラムとシェルのビルトインコマンドとの互換性のためだけのもので、何の効果も及ぼしません。

‘-h’
‘--help’
: このヘルプを表示して終了します。

‘-n’
: 行末の改行を抑止します。デフォルトでは、gettextは出力に改行を付与します。

‘-V’
‘--version’
バージョン情報を表示して終了します。

‘[*textdomain*] *msgid*’
*textdomain*から、*msgid*に対応する翻訳されたメッセージを取得します。

パラメーター *textdomain*を指定しなかった場合、環境変数 TEXTDOMAINにより domain が決定されます。メッセージカタログが標準のディレクトリーで見つからなかった場合には、環境変数 TEXTDOMAINDIRで他の場所を指定できます。

-sオプションを使うと、プログラムは‘echo’コマンドのように振る舞います。しかし、このプログラムは引数を単に標準出力にコピーするのではなく、選択されたカタログにメッセージが見つかった場合は翻訳されたメッセージを出力します。

注意: xgettextは gettext呼び出しで 1 つだけ引数を指定した形式 (オプションを指定せず、環境変数から暗黙に *textdomain*を取得する形式) だけをサポートします。

15.5.2.4 ngettextプログラムの呼び出し

ngettext [*option*] [*textdomain*] *msgid msgid-plural count*

ngettextは数に依存した文法をもつテキストメッセージを母国語に翻訳して表示するプログラムです。

Arguments

‘-d *textdomain*’
‘--domain=*textdomain*’
*textdomain*から翻訳されたメッセージを取得します。*textdomain*は通常、パッケージか、プログラムやプログラムのモジュールと一致します。

‘-e’ いくつかのエスケープシーケンスの展開を可能にします。これは‘gettext’プログラムにたいする互換のためのオプションです。対象となるエスケープシーケンスは、‘\a’、‘\b’、‘\c’、‘\f’、‘\n’、‘\r’、‘\t’、‘\v’、‘\\’、および‘\’と 3 桁以内の 8 進数です (System V の‘echo’と同様に処理されます)。

‘-E’ このオプションは‘gettext’プログラムとの互換性のためだけのもので、何の効果も及ぼしません。

‘-h’
‘--help’ このヘルプを表示して終了します。

‘-V’
‘--version’
バージョン情報を表示して終了します。

‘*textdomain*’
*textdomain*から、翻訳されたメッセージを取得します。

‘*msgid msgid-plural*’
msgid(English の singular)、および msgid-plural(English の plural) を翻訳します。

‘*count*’ この値にもとづいて、singular と plural のどちらを使うか選択します。

パラメーター *textdomain*を指定しなかった場合、環境変数 TEXTDOMAINにより domain が決定されます。メッセージカタログが標準のディレクトリーで見つからなかった場合には、環境変数 TEXTDOMAINDIRで他の場所を指定できます。

注意: xgettextは ngettext呼び出しで 3 つの引数を指定した形式 (オプションを指定せず、環境変数から暗黙に *textdomain*を取得する形式) だけをサポートします。

15.5.2.5 envsubstプログラムの呼び出し

```
envsubst [option] [shell-format]
```

envsubstは環境変数をその値に置き換えるプログラムです。

Operation mode

‘-v’
‘--variables’
*shell-format*の中に出現する変数を出力します。

Informative output

‘-h’
‘--help’ このヘルプを表示して終了します。

‘-V’
‘--version’
バージョン情報を表示して終了します。

通常では、標準入力から$VARIABLEまたは${VARIABLE}という形式で参照される環境変数を読み込んで、対応する値に置き換えてから標準出力にコピーします。*shell-format*が与えられたときは、*shell-format*で指定された環境変数だけが置き換えられます。指定しなかった場合は、標準入力の中に出現するすべての環境変数が置き換えられます。

この置き換えは、ダブルクォートされた文字列をシェルがアンクォートするときに行われる置き換え処理のサブセットです。他の${*variable-default*}、$(*command-list*)、‘*command-list*‘などにたいする置き換えは、セキュリティー上の理由から envsubstではなくシェルによって処理されます。

--variablesを指定したときは、*shell-format*に含まれる環境変数名を 1 行に 1 つずつ出力します (標準入力は無視されます)。

15.5.2.6 eval_gettextプログラムの呼び出し

```
eval_gettext msgid
```

この関数は、テキストメッセージを母国語に翻訳して、翻訳した結果文字列に含まれる$記号のついた変数にたいして置き換え処理を行ってから出力します。$変数にたいする置き換えは、*msgid*に含まれるシェル変数にたいしてだけ行われることに注意してください。

15.5.2.7 eval_ngettextプログラムの呼び出し

```
eval_ngettext msgid msgid-plural count
```

この関数は、数に依存した文法をもつテキストメッセージを母国語に翻訳して、翻訳した結果文字列に含まれる$記号のついた変数にたいして置き換え処理を行ってから出力します。$変数にたいする置き換えは、*msgid*または *msgid-plural*に含まれるシェル変数にたいしてだけ行われることに注意してください。

15.5.3 bash - Bourne-Again シェルスクリプト

GNU bash 2.0 以降には、変数の中の文字列を翻訳して置き換えるための特別な略記法$"msgid"があります。しかし、これによりもたらされるセキュリティーホールと可搬性の問題により、この機能の使用には賛成できません。

$"..."によるセキュリティーホールとは、その文字列にたいする翻訳を検索した後、'eval'が 2 重引用符に囲まれた文字列や$ 記号、バッククォートされた文字列にたいして行うのと同様なことをbashが行う点にあります。

1. エンコーディングが BIG5、BIG5-HKSCS、GBK、GB18030、SHIFT_JIS、JOHAB のいずれかを使う locale では、2byte 文字の 2byte 目が 0x60の文字が存在します。たとえば、これらの locale では\xe0\x60というバイト並びは 1 つの文字です。bashの多くのバージョン (bash-2.05 以降、および mbsrtowcs() 関数を持たないプラットフォーム向けの新しいバージョン) は、文字境界を認識しないので、特定の Chinese 文字をバッククォートと認識します。このため翻訳の一部がコマンドリストとして実行されてしまうことが起こり得るのです。この状況は翻訳者が気をつけていても起こり得ます。翻訳者が翻訳を UTF-8 エンコーディングで提供したとしても、その翻訳は gettext()関数によって翻訳者のエンコーディングからユーザーの locale のエンコーディングに変換され、その変換によって"危険な"\x60というバイトが生成される可能性があるからです。
2. 故意にせよ不注意にせよ、翻訳者は翻訳の中にバッククォート"`...`"や、$カッコ"$(...)"を使用することもあり得るので、それらに囲まれた文字列はコマンドリストとしてシェルにより実行されてしまいます。

可搬性の問題とは、bashをインターナショナリゼーションのサポート付きでビルドしなければならないことです。これは libc に gettext()関数がないシステムでは、通常できません。

15.5.4 Python

RPM python

ファイル拡張子
py

文字列構文 '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 の略記
_('abc')など

gettext/ngettext 関数
gettext.gettext、gettext.dgettext、gettext.ngettext、
gettext.dngettext、ugettext、ungettext

textdomain
gettext.textdomain関数、または gettext.install(*domain*)関数

bindtextdomain
gettext.bindtextdomain関数、または gettext.install(*domain*,*localedir*)関数

setlocale gettext エミュレーションでは使用されません

必要条件　`import gettext`

GNU gettext の使用またはエミュレート
　　エミュレート

抽出プログラム
　　`xgettext`

位置の書式　`'...%(ident)d...' % { 'ident': value }`

可搬性　完全な可搬性がある

po-mode でのマーキング
　　—

`examples`ディレクトリーの例 `hello-python`を利用できます。

書式文字列についての注意: Python は`'...%d...'`のような名前なし引数の書式文字列をサポートと、`'...%(ident)d...'`のような名前つき引数の書式文字列をサポートする。以下の 2 つの理由により、インターナショナライズされたプログラムでは後者の方が好ましい

- 書式文字列が 2 つ以上の引数をとるときに、書式文字列に名前付きの引数を使っていれば、翻訳者は引数を異なる順序で使用するような翻訳を提供できます。例えば以下のような書式文字列

  ```
  "'%(volume)s' has only %(freespace)d bytes free."
  ```

  から

  ```
  "Only %(freespace)d bytes free on '%(volume)s'."
  ```

  に書き換えることができます。さらに識別名により翻訳者にコンテキストを提供できます。
- 多くの言語の plural form のコンテキストでは、singular form のための書式文字列で、数値引数は使用されません。English でも、`"1 hour"`より`"one hour"`と書くのが好まれたりします。このように、書式文字列から特定の引数を除外するのは、名前付き引数の構文でのみ可能なことです (名前なし引数の場合、Python は – C とは異なり – 与えられたすべての引数が書式文字列で使用されているかチェックするからです)。

15.5.5 GNU clisp - Common Lisp

RPM　clisp 2.28 以降

ファイル拡張子
　　`lisp`

文字列構文　`"abc"`

gettext の略記
　　`(_ "abc")`、`(ENGLISH "abc")`

gettext/ngettext 関数
　　`i18n:gettext`、`i18n:ngettext`

textdomain
　　`i18n:textdomain`

bindtextdomain
　　`i18n:textdomaindir`

setlocale　自動

必要条件 —

GNU gettext の使用またはエミュレート
使用

抽出プログラム
`xgettext -k_ -kENGLISH`

位置の書式 `format "~1@*~D ~0@*~D"`

可搬性 gettext のないプラットフォームでは、翻訳しません

po-mode でのマーキング
—

`examples`ディレクトリーの例 `hello-clisp`が利用できます

15.5.6 GNU clisp ソース

RPM clisp

ファイル拡張子
`d`

文字列構文 `"abc"`

gettext の略記
```
ENGLISH ? "abc" : ""
GETTEXT("abc")
GETTEXTL("abc")
```

gettext/ngettext 関数
`clgettext`、`clgettextl`

textdomain
—

bindtextdomain
—

setlocale 自動

必要条件 `#include "lispbibl.c"`

GNU gettext の使用またはエミュレート
使用

抽出プログラム
`clisp-xgettext`

位置の書式 `fprintf "%2$d %1$d"`

可搬性 gettext のないプラットフォームでは、翻訳しません

po-mode でのマーキング
—

15.5.7 Emacs Lisp

RPM emacs、xemacs

ファイル拡張子
`el`

文字列構文 `"abc"`

gettext の略記
`(_"abc")`

gettext/ngettext 関数
`gettext`、`dgettext`(xemacs のみ)

textdomain
スペシャルフォーム `domain`(xemacs のみ)

bindtextdomain
関数 `bind-text-domain`(xemacs のみ)

setlocale 自動

必要条件 —

GNU gettext の使用またはエミュレート
使用

抽出プログラム
`xgettext`

位置の書式 `format "%2$d %1$d"`

可搬性 XEmacs のみ。ビルド時に `I18N3`を定義しない場合は、翻訳しません

po-mode でのマーキング
—

15.5.8 librep

RPM librep 0.15.3 以降

ファイル拡張子
`jl`

文字列構文 `"abc"`

gettext の略記
`(_"abc")`

gettext/ngettext 関数
`gettext`

textdomain
`textdomain`関数

bindtextdomain
`bindtextdomain`関数

setlocale —

必要条件 `(require 'rep.i18n.gettext)`

GNU gettext の使用またはエミュレート
使用

抽出プログラム
`xgettext`

位置の書式 `format "%2$d %1$d"`

可搬性 gettext のないプラットフォームでは、翻訳しません

po-mode でのマーキング
—

`examples`ディレクトリーの例 `hello-librep`が利用できます

15.5.9 GNU guile - Scheme

RPM guile

ファイル拡張子
`scm`

文字列構文 `"abc"`

gettext の略記
`(_ "abc")`

gettext/ngettext 関数
`gettext`、`ngettext`

textdomain
`textdomain`

bindtextdomain
`bindtextdomain`

setlocale `(catch #t (lambda () (setlocale LC_ALL "")) (lambda args #f))`

必要条件 `(use-modules (ice-9 format))`

GNU gettext の使用またはエミュレート
使用

抽出プログラム
`xgettext -k_`

位置の書式 —

可搬性 gettext のないプラットフォームでは、翻訳しません

po-mode でのマーキング
—

`examples`ディレクトリーの例 `hello-guile`が利用できます

15.5.10 GNU Smalltalk

RPM smalltalk

ファイル拡張子
`st`

文字列構文 `'abc'`

gettext の略記
`NLS ? 'abc'`

gettext/ngettext 関数
`LcMessagesDomain>>#at:`、`LcMessagesDomain>>#at:plural:with:`

textdomain
`LcMessages>>#domain:localeDirectory:`(オブジェクト `LcMessagesDomain`を返します)
例: `I18N Locale default messages domain: 'gettext' localeDirectory: '/usr/local/share/locale'`

bindtextdomain
`LcMessages>>#domain:localeDirectory:`、上記参照

setlocale `I18N Locale default`を use する場合は自動

必要条件 `PackageLoader fileInPackage: 'I18N'!`

GNU gettext の使用またはエミュレート
エミュレート

抽出プログラム
`xgettext`

位置の書式 `'%1 %2' bindWith: 'Hello' with: 'world'`

可搬性 完全な可搬性がある

po-mode でのマーキング
—

`examples`ディレクトリーの例 `hello-smalltalk`が利用できます

15.5.11 Java

RPM java、java2

ファイル拡張子
`java`

文字列構文 `"abc"`

gettext の略記
`_("abc")`

gettext/ngettext 関数
`GettextResource.gettext`、`GettextResource.ngettext`、`GettextResource.pgettext`、`GettextResource.npgettext`

textdomain
: —
  かわりに `ResourceBundle.getResource`を使用してください

bindtextdomain
: —
  かわりに CLASSPATH を使用してください

setlocale 自動

必要条件 —

GNU gettext の使用またはエミュレート
: —
  Java specific message catalog format を使用してください

抽出プログラム
: `xgettext -k_`

位置の書式 `MessageFormat.format "{1,number} {0,number}"`

可搬性 完全な可搬性がある

po-mode でのマーキング
: —

文字列をインターナショナライズ可能とマークする前に、文字列結合演算子を使って、`MessageFormat`アプリケーションで変換が必要な文字列を結合してください。たとえば`"file "+filename+" not found"`は 、`MessageFormat.format("file {0} not found", new Object[] { filename })`となります。これを行った後のみ、文字列をマークして抽出することが可能になります。

GNU gettext は Java 本来がもつ、`ResourceBundle`という名前のインターナショナリゼーションのメカニズムを使います。`ResourceBundle`には、`.properties`と`.class`という、2 つのファイルフォーマットがあります。`.properties`ファイルは、PO ファイルのように翻訳者が直接編集できるテキスト形式ですが、plural form をサポートしません。`.class`は、`.java`のソースコードからコンパイルされた形式で、plural form をサポートします (適切な API を通じてのアクセスが提供されています。以下を参照してください)。

PO ファイルを`.properties`ファイルに変換するためには、`msgcat`プログラムの`--properties-output`オプションを使うことができます。逆に`.properties`ファイルを PO ファイルに変換するには、`msgcat`プログラムの`--properties-input`オプションを使うことができます。PO ファイルを扱うすべてのツールは、`--properties-input`および/または`--properties-output`オプションにより、`.properties`も同様に取り扱うことができます。

PO ファイルを ResourceBundle class に変換するためには、`msgfmt`プログラムの、`--java`(または`--java2`) オプションを使うことができます。逆に ResourceBundle を PO ファイルに変換するためには、`msgunfmt`の`--java`オプションを使うことができます。

プログラムから ResourceBundle にアクセスするために、異なる 2 つの API を使うことができます。これら 2 つの API は、ResourceBundle が GNU gettext により生成されたものであるか、それともこれ以外の`.class`や`.properties`ファイルであるかによらず、すべての種類の ResourceBundle を扱えることに注意してください。

1. `java.util.ResourceBundle` API

   特徴は、これの `getString`関数が、翻訳された文字列を戻すことです。翻訳がない場合には、`MissingResourceException`が発生することに注意してください。

   これは、標準の API に採用されるためには有利な特徴といえます。さらに、この API は追加のライブラリーを必要とせず、`msgcat`によって生成された`.properties`ファイルか、`msgfmt`によって生成された`.class`ファイルだけが必要です。しかし、この API は plural を処理できず、それは plural を処理する PO ファイルから `msgfmt`で生成されたリソースの場合も同様です。

2. `gnu.gettext.GettextResource` API

   javadoc2 directory (`javadoc2/index.html`) に、Javadoc 1.1 style format の Reference documentation があります。

   この API の `gettext`関数は、翻訳された文字列を戻します。翻訳がない場合には、*msgid*が変更されずに戻されることに注意してください。

   この API の有利な点は、plural を処理する `ngettext`関数があること、そして特定の context にたいする制約をもつ文字列を処理する `pgettext`と `npgettext`のある点です。

   この API を使うために必要なのは、GNU gettext パッケージの一部である `libintl.jar`だけで、これは LGPL のもとで配布されています。

2 番目の API を使うための例としては、`examples`ディレクトリーの、`hello-java`、`hello-java-awt`、`hello-java-swing`、`hello-java-qtjambi`が利用できます。

では API の使い方と、‘`getString`’の略記をしてみましょう。以下の 3 つの用法から選択することができましょう:

- (Java 1.5 以降の場合) あなたのプロジェクトに ‘`Util`’という一意なクラスがあるとき、`ResourceBundle`のインスタンスを保持する static な変数を定義する場合、以下のような略記を定義します:

  ```
  private static ResourceBundle myResources =
    ResourceBundle.getBundle("domain-name");
  public static String _(String s) {
    return myResources.getString(s);
  }
  ```

  そして、インターナショナライズする文字列を含むすべてのクラスに、以下の宣言を含めます

  ```
  import static Util._;
  ```

  これで以下のようにして略記を使うことができます:

  ```
  System.out.println(_("Operation completed."));
  ```

- あなたのプロジェクトに ‘`Util`’という一意なクラスがあるとき、`ResourceBundle`のインスタンスを保持する static な変数を定義する場合、以下のような略記を定義します:

  ```
  public static ResourceBundle myResources =
    ResourceBundle.getBundle("domain-name");
  ```

  そして、インターナショナライズする文字列を含むすべてのクラスに、以下の宣言を含めます

  ```
  private static ResourceBundle res = Util.myResources;
  private static String _(String s) { return res.getString(s); }
  ```

  これで以下のようにして略記を使うことができます:

  ```
  System.out.println(_("Operation completed."));
  ```

- resource bundle の定義だけを含む、とても短い名前の ‘`S`’というクラスを追加して、以下のような略記を定義します:

```
public class S {
  public static ResourceBundle myResources =
    ResourceBundle.getBundle("domain-name");
  public static String _(String s) {
    return myResources.getString(s);
  }
}
```

これで以下のようにして略記を使うことができます:

```
System.out.println(S._("Operation completed."));
```

3 つの用法のどれを選ぶかは、あなたのプロジェクトが Java 1.5 より前のバージョンにたいする互換性を必要とするかに依存します。もしその必要がある場合には、プロジェクトのすべてのクラスに 1 文字のクラスを追加するより、すべてのクラスに 2 行追加するほうがよいでしょう。

15.5.12 C#

RPM pnet、pnetlib 0.6.2 以降、または mono 0.29 以降

ファイル拡張子
: `cs`

文字列構文 `"abc"`, `@"abc"`

gettext の略記
: _("abc")

gettext/ngettext 関数
: `GettextResourceManager.GetString`, `GettextResourceManager.GetPluralString`
`GettextResourceManager.GetParticularString` `GettextResourceManager.GetParticular`

textdomain
: `new GettextResourceManager(domain)`

bindtextdomain
: —
実行可能ファイルを含むディレクトリーのサブディレクトリーにコンパイルされた message catalog が配置されます

setlocale 自動

必要条件 —

GNU gettext の使用またはエミュレート
: —
C# specific message catalog format を使用してください

抽出プログラム
: `xgettext -k_`

位置の書式 `String.Format "{1} {0}"`

可搬性 完全な可搬性がある

po-mode でのマーキング
: —

文字列をインターナショナライズ可能とマークする前に、文字列結合演算子を使って、String.Format呼び出しで変換が必要な文字列を結合してください。たとえば"file "+filename+" not found"は、String.Format("file {0} not found", filename)となります。これを行った後のみ、文字列をマークして抽出することが可能になります。

GNU gettext は、ResourceManagerおよびResourceSetという名前の、C#/.NET 本来のインターナショナリゼーションメカニズムを使います。アプリケーションはResourceManagerのメソッドを使って、母国語に翻訳された文字列を取得します。メッセージカタログファイルをインメモリーに展開したものが、ResourceSetのインスタンスです。ResourceManagerは、翻訳を検索する必要が生じると、ResourceSetのインスタンスをロード・アクセスします。

C#のランタイムが直接ロードできるResourceSetには、.resourcesファイル、および.dllファイルという、2 つの形式があります:

- .resourcesの形式はバイナリーのファイルで、通常はresgenまたはmonoresgenユーティリティーにより生成され、この形式は plural form をサポートしません。.resourcesは、.NETの.exeファイルに埋め込むこともできます。これは、メッセージカタログをロードするためにファイルシステムにアクセスするかどうかに影響を及ぼすだけで、メッセージカタログの内容には影響しません。
- 一方.dllの形式は、ソースコードの.csからコンパイルされたバイナリーファイルで、plural form をサポートします (これは以下で記述するように、GNU gettext API を通じてアクセスすることにより提供されます)。

これら.NET の.dllや.exeファイルは、特定のプラットフォームに限定されたものではないことに注意してください。これらのファイル形式、および C#のための GNU gettext は、任意のプラットフォームで使用できます。

msgfmtプログラムに'--csharp-resources'オプションを指定することにより、PO ファイルを.resourcesファイルに変換できます。また、msgunfmtプログラムに'--csharp-resources'オプションを指定することにより、.resourcesファイルから PO ファイルに逆変換できます。これらの処理は、resgenプログラム (pnetパッケージ) や、monoresgenプログラム (mono/mcsパッケージ)でできる場合もあります。これらのプログラムは、.resourcesファイルから PO ファイルへの逆変換もできます。しかし、この文書を記述している時点 (2004 年 1 月) では、monoresgenコンバーターにはバグが多く、resgenコンバーターは PO ファイルのエンコーディングを無視することに注意してください。

msgfmtプログラムに--csharpオプションを指定することにより、PO ファイルを.dllファイルに変換できます。GettextResourceSet(このクラスも ResourceSetのサブクラスです) のサブクラスを含んだ、.dllファイルを得ることができます。また、msgunfmtプログラムに--csharpオプションを指定することにより、サブクラス GettextResourceSetを含む.dllファイルを PO ファイルに逆変換できます。

.resources形式に比べて、.dll形式には、以下のような利点があります:

1. 自由なローカライズ: アプリケーションをビルド・配布した後でも、ユーザーは自分の翻訳を自由に追加できます。一方、システムにより提供される ResourceManagerのコンストラクターをプログラマーが使った場合、アプリケーション用の一連の.resourcesファイルは、アプリケーションのビルド時に指定しなければならず、後から拡張はできません。
2. Plural の処理: .dll形式のメッセージカタログは、plural を処理するための関数GetPluralStringをサポートします。一方、.resourcesファイルは、含まれているデータと、単一の文字列にもとづく検索だけをサポートします。
3. Context の処理: .dll形式のメッセージカタログは、context にもとづく問い合わせをおこなうための関数 GetParticularStringおよび GetParticularPluralStringをサポートしま

す。一方、.resourcesファイルは、含まれているデータと、単一の文字列にもとづく検索だけをサポートします。

4. GettextResourceManagerは、.dll形式の中のメッセージカタログのロードと、メッセージ単位でのロードも提供します。たとえばAustrian(de_AT) のlocaleでは、メッセージがAustrianのメッセージカタログにないときに、German(de) のメッセージカタログが使用されます。つまり、Austrianの翻訳者は、Germanの翻訳とは異なるいくつかのメッセージだけを、Austrianに翻訳する必要があるということです。一方、.resourcesファイルでは、各メッセージカタログは、それ自身に含まれるすべてのメッセージの翻訳を提供しなければなりません。
5. GettextResourceManagerは、翻訳が見つからないときはEnglishの *msgid*を戻すというフォールバック付きで、.dll形式のメッセージカタログをロードします。一方、.resourcesファイルでは、その.resourcesが言語中立な場合、フォールバックを明示的に提供しなければなりません。

プログラム用のAPIという面では、プログラマーは標準のResourceManager APIと、GNU GettextResourceManager APIのどちらを使うこともできます。前者のResourceManagerのサブクラスが、後者のGettextResourceManagerなので、後者の方が拡張されています。

1. System.Resources.ResourceManager API

   このAPIは、.resources形式のリソースにたいして動作します。

   ResourceManagerを生成するには、以下のようにします

   ```
   new ResourceManager(domainname, Assembly.GetExecutingAssembly())
   ```

   GetString関数は、文字列にたいする翻訳を戻します。翻訳がない場合はnullが戻されることに注意してください (例: これはフォールバックのリソースファイルの場合にも適用されます)。
2. GNU.Gettext.GettextResourceManager API

   このAPIは、.dll形式のリソースにたいして動作します。

   Reference documentationは、csharpdoc directory (csharpdoc/index.html) にあります。

   ResourceManagerを生成するには、以下のようにします

   ```
   new GettextResourceManager(domainname)
   ```

   このAPIのGetString関数は、翻訳された文字列を戻します。翻訳がない場合には、*msgid*が変更されずに戻されることに注意してください。

   GetPluralString関数は、Cの ngettext関数のように、文字列にたいしてplural処理をした翻訳を戻します。

   GetParticularString関数は、Cの pgettext関数のように、特定のcontextが指定された文字列の翻訳を戻します。翻訳がない場合には、*msgid*が変更されずに戻されることに注意してください。

   GetParticularPluralString関数は、Cの npgettext関数のように、特定のcontextが指定された文字列にたいして、plural処理をした翻訳を戻します。

   このAPIを使うために必要なのは、GNU gettextパッケージの一部である GNU.Gettext.dllだけで、これはLGPLのもとで配布されています。

2つのアプローチをミックスすることもできます: たとえばGNU.Gettext.GettextResourceManagerコンストラクターは使うが、ResourceManager型と GetStringメソッドだけを使うような場合です。これはPOファイル用のツールに適合させたいが、ResourceManagerを使う既存のソースコードを変更したくなくて、(まだ)GetPluralStringメソッドが必要ないときには適しているでしょう。

2番目のAPIを使うためには、examplesディレクトリーの hello-csharp、hello-csharp-formsの2つの例が利用できます。

では API の使い方と、‘GetString’の略記をしてみましょう。以下の 3 つの用法から選択することができましょう:

- あなたのプロジェクトに‘Util’という一意なクラスがあるとき、ResourceManagerのインスタンスを保持する static な変数を定義する場合は、以下のような略記を定義します:

```
public static GettextResourceManager MyResourceManager =
  new GettextResourceManager("domain-name");
```

  そして、インターナショナライズする文字列を含むすべてのクラスに、以下の宣言を含めます

```
private static GettextResourceManager Res = Util.MyResourceManager;
private static String _(String s) { return Res.GetString(s); }
```

  これで以下のようにして略記を使うことができます:

```
Console.WriteLine(_("Operation completed."));
```

- resource manager の定義だけを含む、とても短い名前の‘S’というクラスを追加して、以下のような略記を定義します:

```
public class S {
  public static GettextResourceManager MyResourceManager =
    new GettextResourceManager("domain-name");
  public static String _(String s) {
     return MyResourceManager.GetString(s);
  }
}
```

  これで以下のようにして略記を使うことができます:

```
Console.WriteLine(S._("Operation completed."));
```

2 つの用法のどちらを選ぶかは、すべてのクラスに上記の 2 行をコピーするのがよいか、それともすべてのクラスに 1 文字のクラスを追加するのがよいかによります。

15.5.13 GNU awk

RPM gawk 3.1 以降

ファイル拡張子
: `awk`

文字列構文 `"abc"`

gettext の略記
: `_"abc"`

gettext/ngettext 関数
: `dcgettext`、gawk-3.1.0 に `dcngettext`はありません

textdomain
: `TEXTDOMAIN`変数

bindtextdomain
: `bindtextdomain`関数

setlocale 自動、ただし gawk-3.1.0 には `setlocale (LC_MESSAGES, "")`はありません

必要条件 —

GNU gettext の使用またはエミュレート
: 使用

抽出プログラム
: `xgettext`

位置の書式 `printf "%2$d %1$d"`(GNU awk のみ)

可搬性 gettext のないプラットフォームでは翻訳されません。非 GNU の awk では、`dcgettext`、`dcngettext`、`bindtextdomain`を自分で定義しなければなりません。

po-mode でのマーキング
: —

`examples`ディレクトリーの、例 `hello-gawk`が利用できます

15.5.14 Pascal - フリー Pascal コンパイラー

RPM fpk

ファイル拡張子
: `pp`、`pas`

文字列構文 `'abc'`

gettext の略記
: 自動

gettext/ngettext 関数
: —
: かわりにデータ型 `ResourceString`を使用してください

textdomain
: —
: , かわりに関数 `TranslateResourceStrings`を使用してください

bindtextdomain
: —
: , かわりに関数 `TranslateResourceStrings`を使用してください

setlocale 自動、ただし使うのは LANG だけで、LC_MESSAGES や LC_ALL は使いません

必要条件 `{$mode delphi}`、または`{$mode objfpc}`
`uses gettext;`

GNU gettext の使用またはエミュレート
: エミュレート (部分的)

抽出プログラム
: `xgettext`(または `rstconv`) サポートのある `ppc386`

位置の書式 `uses sysutils;`
`format "%1:d %0:d"`

可搬性 ?

po-mode でのマーキング
: —

Pascal コンパイラーには、ResourceStringデータ型にたいする特別なサポートがあります。これは.rstファイルを生成します。このファイルは、xgettextまたはrstconvを使用することにより、.potファイルに変換されます。実行時には、gettextユニット内の TranslateResourceStrings 関数を使用して、この.potファイルの翻訳に対応する.moファイルをロードできます。

examplesディレクトリーの、例 hello-pascalが利用できます。

15.5.15 wxWidgets ライブラリー

RPM wxGTK、gettext

ファイル拡張子
cpp

文字列構文 "abc"

gettext の略記
_("abc")

gettext/ngettext 関数
wxLocale::GetString、wxGetTranslation

textdomain
wxLocale::AddCatalog

bindtextdomain
wxLocale::AddCatalogLookupPathPrefix

setlocale wxLocale::Init、wxSetLocale

必要条件 #include <wx/intl.h>

GNU gettext の使用またはエミュレート
エミュレート、include/wx/intl.hと src/common/intl.cppを参照してください。

抽出プログラム
xgettext

位置の書式 wxString::Format はシステムに wprintf()と vswprintf()関数があり、それらが POSIX 準拠の位置指定をサポートする場合のみ、位置指定をサポートします。

可搬性 完全な可搬性がある

po-mode でのマーキング
yes

15.5.16 YCP - YaST2 スクリプト言語

RPM libycp、libycp-devel、yast2-core、yast2-core-devel

ファイル拡張子
ycp

文字列構文 "abc"

gettext の略記
_("abc")

gettext/ngettext 関数
: 引数を 1 つ、または 3 つの`_()`

textdomain
: `textdomain`命令

bindtextdomain
: —

setlocale —

必要条件 —

GNU gettext の使用またはエミュレート
: 使用

抽出プログラム
: `xgettext`

位置の書式 `sformat "%2 %1"`

可搬性 完全な可搬性がある

po-mode でのマーキング
: —

`examples`ディレクトリーの、例`hello-ycp`が利用できます

15.5.17 Tcl - Tk のスクリプト言語

RPM tcl

ファイル拡張子
: `tcl`

文字列構文 `"abc"`

gettext の略記
: `[_ "abc"]`

gettext/ngettext 関数
: `::msgcat::mc`

textdomain
: —

bindtextdomain
: —
: かわりに`::msgcat::mcload`を使用してください

setlocale 自動、LANG は使いますが、LC_MESSAGES と LC_ALL は無視されます

必要条件
```
package require msgcat
proc _ {s} {return [::msgcat::mc $s]}
```

GNU gettext の使用またはエミュレート
: — Tcl specific message catalog format を使用してください

抽出プログラム
: `xgettext -k_`

位置の書式 `format "%2\$d %1\$d"`

可搬性 完全な可搬性がある

po-mode でのマーキング
: —

`examples`ディレクトリーの 2 つの例 `hello-tcl`と `hello-tcl-tk`が利用できます

文字列をインターナショナライズ可能とマークする前に、代用の変数を `format`アプリケーションで変換する必要がある文字列にします (例: `"file $filename not found"`は `[format "file %s not found" $filename]`)。これを行った後のみ、文字列をマークして抽出できます。上記の例では、マークした後は `[format [_ "file %s not found"] $filename]`(または `[msgcat::mc "file %s not found" $filename]`) となります。msgcat::mc 関数は、2 つ以上の引数が与えられたときは、暗黙的に `format`を呼び出すことに注意してください。

15.5.18 Perl

RPM perl

ファイル拡張子
: `pl`、`PL`、`pm`、`perl`、`cgi`

文字列構文

- `"abc"`
- `'abc'`
- `qq (abc)`
- `q (abc)`
- `qr /abc/`
- `qx (/bin/date)`
- `/pattern match/`
- `?pattern match?`
- `s/substitution/operators/`
- `$tied_hash{"message"}`
- `$tied_hash_reference->{"message"}`
- その他 (この問題は‘`man perlsyn`’コマンドで詳細に論じられています)

gettext の略記
: `__`(2 連のアンダースコアー)

gettext/ngettext 関数
: `gettext`、`dgettext`、`dcgettext`、`ngettext`、`dngettext`、`dcngettext`

textdomain
: `textdomain`関数

bindtextdomain
: `bindtextdomain`関数

bind_textdomain_codeset
: `bind_textdomain_codeset`関数

setlocale
: Use `setlocale (LC_ALL, "");`

必要条件
: `use POSIX;`
`use Locale::TextDomain;`(libintl-perl パッケージに含まれています。これは CPAN(Comprehensive Perl Archive Network) http://www.cpan.org/) で入手できます。

GNU gettext の使用またはエミュレート
: プラットフォームに依存: gettext_pp はエミュレート、gettext_xs は GNU gettext を使っています

抽出プログラム
: `xgettext -k__ -k\$__ -k%__ -k__x -k__n:1,2 -k__nx:1,2 -k__xn:1,2 -kN__ -k`

位置の書式
: どちらの種類の位置指定付き書式もサポートしています
`printf "%2\$d %1\$d", ...`(Perl 5.8.0 以降)
`__expand("[new] replaces [old]", old => $oldvalue, new => $newvalue)`

可搬性
: `libintl-perl`パッケージはプラットフォームに依存しませんが、Perl の中核となるものの一部です。ターゲットとなるシステムにパッケージがインストールされていない場合、必要な関数のダミーの実装を提供するのは、プログラマーの責任です。

po-mode でのマーキング
: —

ドキュメント
: `libintl-perl`に含まれています。これは CPAN(http://www.cpan.org/) で利用できます

`examples`ディレクトリーの、例`hello-perl`が利用できます

Perl 用の`xgettext`パーサーのバックエンドと、他のプログラム言語用のパーサーのバックエンドには、重大な違いがあります。これは Perl 自体に、他の言語との間に重大な違いがあるからです。Perl 用のパーサーのバックエンドは、文字列をマークするための機能を、他のバックエンドより多く提供しますが、これには Perl 固有の制限がいくつかあり、その中でもっとも問題となるのは言語の不完全性によるものでしょう。

15.5.18.1 Perl コードをパースするときの一般的な問題

Perl だけが Perl をパースできるということを、しばしば耳にしますが、これは真実ではありません。結局 Perl をパースすることはできず、実行することができるだけなのです。Perl にはビルトインでさまざまなあいまいさがあり、それを解決できるのは実行時だけなのです。

以下は一般的な例を示した例です:

```
print gettext "Hello World!";
```

これは関数呼び出しの"堅実"な例に見えるかもしれませんが、そうでもありません:

```
open gettext, ">testfile" or die;
print gettext "Hello world!"
```

上記のコンテキストでは、文字列`gettext`はファイルハンドルのように見えます。しかし必ずしもそうではありません:

```
use Locale::Messages qw (:libintl_h);
open gettext ">testfile" or die;
print gettext "Hello world!";
```

この例では、Perl のインクルードパスで最初に見つかる Locale::Messagesモジュールがエクスポートする gettext関数により提供されるファイル名は、多分文法エラーになるでしょう。しかし実際の Locale::Messagesモジュールが以下のようなものだったらどうなるでしょうか?

```
use vars qw (*gettext);

1;
```

このケースでは、文字列 gettextは再びファイルハンドルとして解釈されるので、testfileというファイルを作成して、そのファイルに “Hello world!” という文字列を書き込む例になります。より高度なフロー分析による制御も、助けにはなりません:

```
if (0.5 < rand) {
   eval "use Sane";
} else {
   eval "use InSane";
}
print gettext "Hello world!";
```

わたしたちが期待するような gettext関数をエクスポートするモジュールが Saneで、gettextというハンドルの出力ストリームを書き込み用にオープンするのが InSaneというモジュールの場合、わたしたちには実行時に何が起こるのか知る糸口がなく、完全に予測不能です。本当のところ Perl には、実行時にそれ自身のシンボルテーブルにシンボルを追加するために非常に多くの方法があり、実行することなく特定のコードの断片を解釈するのは不可能なのです。

もちろん xgettextは翻訳可能な文字列を走査するときに、走査する Perl のソースを実行するわけではなく、コードを記述した人が何を意図したのかを推測するために発見的な手法を用います。

他にもスラッシュとクエスチョンマークの曖昧さという問題があります。これらの解釈はコンテキストに依存します:

```
# A pattern match.
print "OK\n" if /foobar/;

# A division.
print 1 / 2;

# Another pattern match.
print "OK\n" if ?foobar?;

# Conditional.
print $x ? "foo" : "bar";
```

スラッシュは除算のための演算子と、パターンマッチをあらわすための両方の用途で使用されます。一方、クエスチョンマークは 3 項演算の条件判定と、パターンマッチでも使用されます。他の awkのようなプログラム言語にも同様な問題はありますが、このようにソースを誤って解釈してしまう問題は Perl のソースの場合が特に顕著なのです。たとえば awkでは、命令文は決して 1 行を超えないので、パーサーはパースエラーから復帰して、次の行から残りの入力ストリームを正しく処理できます。これと異なり Perl は、コンテキストの意味とは無関係に、入力ストリームに次の区切り文字 (スラッシュまたはクエスチョンマーク) が出現することでパターンマッチが終端されます。本来は除算演

算子として使用されているスラッシュがパターンマッチと誤って解釈された場合、おそらく残りの入力ファイルは正常にパースされないでしょう。

以下に、あいまいさの解決が不可能なケースを示します:

```
$x = wantarray ? 1 : 0;
```

Perl のビルトイン関数である wantarrayは引数をとらないので、Perl のパーサーはクエスチョンマークが正規表現の開始ではなく、3 項演算子だと知ることができます。

```
sub wantarrays {}
$x = wantarrays ? 1 : 0;
```

今度は状況が異なります。関数 wantarraysは、可変個の引数をとります (他のプロトタイプがない任意の Perl 関数と同様です)。この場合、クエスチョンマークはパターンマッチの区切り文字として解釈されるので、このコードの断片はコンパイルされません。

```
sub wantarrays() {}
$x = wantarrays ? 1 : 0;
```

さて今度は関数がプロトタイプされているので、Perl は関数が引数をとらないことを知っているので、クエスチョンマークは再び 3 項演算子として解釈されます。しかし残念ながら xgettextはそうではありません。

xgettextの Perl 用パーサーは、関数にプロトタイプがあるのか、そしてそのプロトタイプがどのように見えるのか知ることができないので、経験的に推測することになります。ある関数が Perl のビルトイン関数で、この関数が引数を受け付けない場合は、それに続くクエスチョンマーク (またはスラッシュ) は演算子として扱われ、それ以外の場合は以降に続く正規表現のための区切り文字として扱われます。Perl のビルトイン関数で引数をとらないのは wantarray、fork、time、times、getlogin、getppid、getpwent、getgrent、gethostent、getnetent、getprotoent、getservent、setpwent、setgrent、endpwent、endgrent、endhostent、endnetent、endprotoent、endserventです。

もし xgettextが、あなたのソースから文字列を抽出するのに失敗する場合には、このセクションで説明したようなスラッシュ(またはクエスチョンマーク) を探してみる必要があります。もしかしたら xgettextの Perl パーサーの、他のバグである可能性もあります (もちろんそのようなバグは報告する必要があるでしょう)。そのうちに、あなたは xgettextに"挑戦"するより、マナーにのっとってコードを整形する必要があると思うでしょう。

特に、引数をとらない関数をパーサーが認識できないときは、カッコを使ってください:

```
$x = somefunc() ? 1 : 0;
$y = (somefunc) ? 1 : 0;
```

実際のところは Perl のパーサーも、このような状況にたいする問題をもっていて、警告を発します。

15.5.18.2 xgettext が探すキーワードはどれ?

xgettextに--keyword(または-k) オプションを指定しない場合は、Perl ソース中の以下のキーワードを認識します:

- gettext
- dgettext
- dcgettext
- ngettext:1,2

  1 番目 (singular) と 2 番目 (plural) の引数が抽出されます。

- `dngettext:1,2`
  1 番目 (singular) と 2 番目 (plural) の引数が抽出されます。
- `dcngettext:1,2`
  1 番目 (singular) と 2 番目 (plural) の引数が抽出されます。
- `gettext_noop`
- `%gettext`
  ハッシュ`%gettext`の中のハッシュ キーが抽出されます。
- `$gettext`
  ハッシュ リファレンス`$gettext`の指すハッシュの中のハッシュ キーが抽出されます。

15.5.18.3 ハッシュキーを抽出する方法

通常の実行時のメッセージ翻訳は、翻訳が格納されたデータベースから元の文字列を検索して、、翻訳された文字列を戻します。これを Perl で実装する “自然な” 方法は、ハッシュのルックアップで、もちろん `xgettext`もこのような実装をサポートしています。

```
print __"Hello world!";
print $__{"Hello world!"};
print $__->{"Hello world!"};
print $$__{"Hello world!"};
```

上の 4 つの行は、すべて同じことを行います。Perl の `Locale::TextDomain`モジュールは、関数`__()`に tied されたハッシュ`%__`をデフォルトでエクスポートします。また、このモジュールは`%__`にたいするリファレンス`$__`もエクスポートします。

`xgettext`に`--keyword`(または`-k`) オプションを指定したときに、その引数の 1 文字目がパーセント記号 (%) の場合、残りの部分のキーワードはハッシュの名前として解釈されます。引数の 1 文字目がダラー記号 ($) の場合、残りの部分のキーワードはハッシュにたいするリファレンスとして解釈されます。

Perl のコードとして許容できるなら (大抵の場合は問題ないはずです)、ハッシュキーを囲むシングルクォーテーション (またはダブルクォーテーション) は省略できることに注意してください:

```
print $gettext{Error};
```

ルールを正確にいうと: ハッシュキーが有効な C の識別子 (!) の場合 (例: 識別子の 1 文字目がアンダースコアーまたは ASCII 文字で、その後ろに任意の個数のアンダースコアー、または半角英数が続くような識別子)、その識別子を囲むクォート記号を省略できます。その他の Unicode 文字は、`use utf8` pragma を使っていない限り、認められません。

15.5.18.4 何が文字列で、何がクォート風の式なのか?

Perl は文字列の構成方法として、過剰ともいえるほどの異なる方法を提供しています。関数の引数や、ハッシュをルックアップするカッコのなかで使用できるこれらの文字列は、`xgettext`でも概ねサポートされています。

- **double-quoted strings**

  ```
  print gettext "Hello World!";
  ```
- **single-quoted strings**

  ```
  print gettext 'Hello World!';
  ```

- 演算子 **qq**

```
print gettext qq |Hello World!|;
print gettext qq <E-mail: <guido\@imperia.net>>;
```

qq演算子は完全にサポートされています。演算子のための区切り文字には、4 種類のカッコ (round、angle、square、curly) を入れ子にして使うことも含めて、任意の区切り文字を使用できます。

- 演算子 **q**

```
print gettext q |Hello World!|;
print gettext q <E-mail: <guido@imperia.net>>;
```

q演算子は完全にサポートされています。演算子のための区切り文字には、4 種類のカッコ (round、angle、square、curly) を入れ子にして使うことも含めて、任意の区切り文字を使用できます。

- 演算子 **qx**

```
print gettext qx ;LANGUAGE=C /bin/date;
print gettext qx [/usr/bin/ls | grep '^[A-Z]*'];
```

qx演算子は完全にサポートされています。演算子のための区切り文字には、4 種類の括弧 (round、angle、square、curly) を入れ子にして使うことも含めて、任意の区切り文字を使用できます。

この例の場合、演算子内部の文字列にたいして gettextは使われません。これは qx演算子の中に指定したコマンドの出力を使って、gettextを呼び出します。これはインターフェースを統一するために提供されている機能です (パーサーはすべての文字列と引用符類を抽出します)。

- ヒアドキュメント

```
print gettext <<'EOF';
program not found in $PATH
EOF

print ngettext <<EOF, <<"EOF";
one file deleted
EOF
several files deleted
EOF
```

ヒアドキュメントは認識されます。ヒアドキュメントを終端する区切り文字列がシングルクォーテーションでくくられていた場合、文字列中の変数は展開されません。区切り文字列がダブルクォーテーションでくくられている場合、または区切り文字でくくられていない場合には、文字列中の変数は展開されます。

数字ではじめまる区切り文字列はサポートされていません!

15.5.18.5 文字列内挿の無効な使い方

Perl では、変数による文字列の補間ができます。これはローカライズされるプログラムに恩恵をもたらしますが、問題を引き起こすこともあります。

以下のような文字列の生成はエラーを発生させます:

```
print gettext "This is the program $0!\n";
```

このような場合 Perl は実行時に、関数 `gettext()`の引数内の、変数`$0`を補間するでしょう。これにより、この引数は文字列定数ではなく文字列変数となります (`$0`はグローバル変数で、実行されている perl のスクリプト名が保持されています)。補間は Perl が文字列引数を `gettext()`に渡す前に行われるので、文字列は実行時にのみ決定可能なスクリプト名に依存することになります。その結果、実行時に翻訳を見つけるのはほとんど不可能になります (メッセージカタログで補間された文字列が偶然見つかってしまった場合を除きます)。

そのため `xgettext`プログラムは、抽出された文字列の中に変数を見つけると、致命的なエラーとして解析を打ちきります。これは一般的に、このような補間文字列すべてをコンパイル時に安全に処理することができないからです。あなたが自分が何をしているか完全に知っている場合には、以下のようにしてこの挙動を回避できます:

```
my $know_what_i_am_doing = "This is program $0!\n";
print gettext $know_what_i_am_doing;
```

パーサーは、変数やその他の単語は認識しませんが、文字列と引用符の類だけなら認識するので、上記の文は許されます。しかし、多分あなたは元文字列をメッセージカタログに抽出するための、別の方法を見つける必要があるでしょう。

`--extract-all`(または`-a`) オプションを指定して呼び出すと、変数の補間ができるようになります。理論的な根拠: 一般的にはソースをインターナショナライズ用に準備するために、このオプションを使うでしょう。

補間される文字列・補間されない文字列と、引用符類の表現についての詳細は、'`man perlop`'の manpage を参照してください。以下は、(致命的なエラーとならない) 安全な補間です:

- エスケープシーケンス`\t`(tab, HT, TAB)、`\\n`(newline、NL)、`\r`(return、CR)、`\f`(form feed、FF)、`\b`(backspace、BS)、`\a`(alarm、bell、BEL)、`\e` (escape、ESC)
- `\033`のような 8 進文字
  400 から 777 の範囲の 8 進シーケンスは、`use utf8` pragma が記述されているか否かにかかわらず、UTF-8 表現に翻訳されることに注意してください。
- hex chars, like `\x1b`
- `\x{263a}`のような 16 進文字
  このエスケープは、`use utf8` pragma が記述されているか否かにかかわらず、UTF-8 表現に翻訳されることに注意してください。
- `\c[`(CTRL-[) のような制御文字
- `\N{LATIN CAPITAL LETTER C WITH CEDILLA}`のような名前つき Unicode 文字
  このエスケープは、`use utf8` pragma が記述されているか否かにかかわらず、UTF-8 表現に翻訳されることに注意してください。

以下のエスケープは、部分的には安全だとみなされます:

- `\l` 次の文字を小文字にします
- `\u` 次の文字を大文字にします
- `\L` `\E` までを小文字にします
- `\U` `\E` までを大文字にします
- `\E` 大文字小文字変換を終端します
- `\Q` `\E` までの非単語文字をクォートします

これらのエスケープは、文字列に ASCII 文字しか含まれていないときだけ、安全とみなされます。ASCII で定義された範囲外の翻訳文字は locale 依存であり、実際には実行時しか処理できません。xgettextは、これらの locale 依存の翻訳を抽出時に処理しません。

修飾子\Qを除き、たとえこれらの翻訳が有効だとしても、一般的には無用であり、ソースがわかりにくくなるだけです。コンパイル時に翻訳が安全に処理できるなら、あなたも、意図することを同じように記述できるはずです。

15.5.18.6 文字列内挿の有効な使い方

Perl は他のプログラム言語のソースや、任意のファイルフォーマットを生成するために使用されることもあります。このような使用方法の例として有名なものには、HTML コードを出力する Web アプリケーションがあります。

以下の HTML の例のように、翻訳可能なメッセージを含む言語 (またはプログラム言語) を、散在させて記述したい状況に出会うこともあるでしょう:

```
print gettext <<EOF;
<h1>My Homepage</h1>
<script language="JavaScript"><!--
for (i = 0; i < 100; ++i) {
    alert ("Thank you so much for visiting my homepage!");
}
//--></script>
EOF
```

parser によりヒアドキュメント全体が抽出されて、埋め込み JavaScript の断片が含まれた HTML コードが、抽出結果である PO ファイルに出現します。上記のような構築法の多用には、そのパッケージの翻訳者がもっと難易度の低いパッケージを探すというリスクがあります。このような場合には以下のような代替の表現を考慮する必要があるでしょう:

```
print <<EOF;
<h1>$gettext{"My Homepage"}</h1>
<script language="JavaScript"><!--
for (i = 0; i < 100; ++i) {
    alert ("$gettext{'Thank you so much for visiting my homepage!'}");
}
//--></script>
EOF
```

この例ではコードの翻訳可能な部分だけが抽出されるので、結果となる PO ファイルは、不本意ながら可読性の点においては改善されるでしょう。

すべての文字列の中の hash のルックアップ、および quote 風な表現は補間することができます (quote風というな表現は、補完という処理が抱えているテーマでもあります。詳細についてはmanpage 'man perlop'を参照してください)。しかし 2 重の補間は無効になります:

```
# TRANSLATORS: Replace "the earth" with the name of your planet.
print gettext qq{Welcome to $gettext->{"the earth"}};
```

最初の位置の qqによりクォートされた文字列が、xgettextの引数として認識されて、無効な可変補間かどうかをチェックされますそのため、hash-dereferencing の$記号は、parser によって “invalid interpolation” エラーとなります。

以下の、正規表現の中の hash lookup の補間は有効です:

```
if ($var =~ /$gettext{"the earth"}/) {
   print gettext "Match!\n";
}
s/$gettext{"U. S. A."}/$gettext{"U. S. A."} $gettext{"(dial +0)"}/g;
```

15.5.18.7 カッコを使用すべきとき

Perl では、関数の引数を囲うカッコは、ほとんどの場合は任意です。常に xgettextは、すべての認識されたキーワード (hash および hash references にたいするものは除く) を、適切にプロトタイプされた関数の名前とみなし、(願わくば)Perl 自体がカッコを必要とする場所だけにカッコが必要です。それゆえ、以下の例の構築例はすべて使用できます:

```
print gettext ("Hello World!\n");
print gettext "Hello World!\n";
print dgettext ($package => "Hello World!\n");
print dgettext $package, "Hello World!\n";

# The "fat comma" => turns the left-hand side argument into a
# single-quoted string!
print dgettext smellovision => "Hello World!\n";

# The following assignment only works with prototyped functions.
# Otherwise, the functions will act as "greedy" list operators and
# eat up all following arguments.
my $anonymous_hash = {
   planet => gettext "earth",
   cakes => ngettext "one cake", "several cakes", $n,
   still => $works,
};
# The same without fat comma:
my $other_hash = {
   'planet', gettext "earth",
   'cakes', ngettext "one cake", "several cakes", $n,
   'still', $works,
};

# Parentheses are only significant for the first argument.
print dngettext 'package', ("one cake", "several cakes", $n), $discarded;
```

15.5.18.8 長い行を理解するには

長いメッセージを必要とすることは、しばしば厄介で読みにくいコーディングスタイルを導き出します。Perl は読みにくいコードを記述するのを防ぐいくつかのオプションをもっており、xgettext もそれにたいしてベストを尽くします。これをおこなうにはドット演算子 (文字列連結演算子) が手軽でしょう:

```
print gettext ("This is a very long"
               . " message that is still"
               . " readable, because"
               . " it is split into"
               . " multiple lines.\n");
```

Perl は、このような文字列定数の断片を、コンパイル時に 1 つの長い文字列に結合できるほどにはスマートであり、それは xgettextも同様です。あなたは処理結果の POT には 1 つの長いメッセージしかないことを見出すでしょう。

将来の Perl 6 では、ドット (‘.’) は dereferencing 用となり、文字列の結合演算子として、おそらくアンダースコア (‘_’) が使われることに注意してください。xgettextでは、この新しい文法はまだサポートされていません。

改行を埋め込むのが問題ない (むしろ望んでいる) 場合には、クォートされた文字列の内側のどこでも改行を挿入できます:

```
print gettext ("<em>In HTML output
embedded newlines are generally no
problem, since adjacent whitespace
is always rendered into a single
space character.</em>");
```

ヒアドキュメントの使用を考えるかもしれません:

```
print gettext <<EOF;
<em>In HTML output
embedded newlines are generally no
problem, since adjacent whitespace
is always rendered into a single
space character.</em>
EOF
```

行は実際に改行されることを忘れないでください (例: これらは改行文字に変換されて、POT ファイル中に出現します)。

15.5.18.9 バグ、落とし穴、動作しない事柄

ここまでのセクションでは、Perl のソースから翻訳可能な文字列を抽出点において、xgettextがとてもスマートであることが証明されました。しかし動作すると期待されていたエキゾチックな構成物のうちのいくつかは、多かれ少なかれ動作しません。

それに関係する制限のうちの 1 つは、クォートされた文字列内の変数の補間に関する実装で見つけることができます。クォートされた文字列の場合、単純な hash lookup しか使うことができません:

```
print <<EOF;
$gettext{"The dot operator"
         . " does not work"
         . "here!"}
Likewise, you cannot @{[ gettext ("interpolate function calls") ]}
inside quoted strings or quote-like expressions.
EOF
```

これは有効な Perl のコードであり、実行時には実際に gettext関数を呼び出します。しかしxgettextの Perl parser は、文字列の認識に失敗します。これほど明確ではない実相の制限は、正規表現の補間で見つけることができます:

```
s/<!--START_OF_WEEK-->/gettext ("Sunday")/e;
```

修飾子 eは、評価可能なステートメントとして解釈して置き換えを行います。その結果、実行時にgettext()関数が呼び出されまが、この場合も parser は文字列 “Sunday” を抽出することに失敗します。この機能を本当に使いたいならば、シンプルな回避策は一時的な変数を使うことです:

```
my $sunday = gettext "Sunday";
s/<!--START_OF_WEEK-->/$sunday/;
```

hash slices も手軽ですが、認識されません:

```
my @weekdays = @gettext{'Sunday', 'Monday', 'Tuesday', 'Wednesday',
                        'Thursday', 'Friday', 'Saturday'};
# Or even:
@weekdays = @gettext{qw (Sunday Monday Tuesday Wednesday Thursday
                         Friday Saturday) };
```

これは tied hash %gettext の完全に有効な使い方ですが、文字列は認識されないため、抽出もされません。

現在のバージョンにたいするその他の注意点は、識別子の中の非アスキー文字にたいするお粗末なサポートがあげられます。'A'-'Z'、'a'-'z'、'0'-'9'、およびアンダースコアー'_' の範囲外の文字を識別子に使った場合、あなたは深刻な問題に直面するでしょう。

これらの存在しない機能のうちのいくつかは将来のバージョンで実装されるかもしれませんが、最小限の努力により回避できるものばかりなので、開発の優先度は低くなっています。

たちの悪いのは、普通のテキストの中の一部にすでに brace が含まれているような brace format strings の問題です。たとえば、プログラムの使い方を説明する文字列は、プログラムの中で普通に出会うものです:

```
die "usage: $0 {OPTIONS} FILENAME...\n";
```

この Perl の brace format strings を含んだコードをインターナショナライズしようとすると、問題が起きます:

```
die __x ("usage: {program} {OPTIONS} FILENAME...\n", program => $0);
```

‘{program}’は placeholder です。一方 ‘{OPTIONS}’は placeholder ではなく、おそらく翻訳される必要があります。しかし最初のものを認識して、他のものをそのままにしておくようにxgettext の Perl parser に教える術はありません。

この問題を回避するためには 2 つの方法が考えられます。プログラムが (printf()で位置パラメーターを扱える)Perl 5.8.0 以降で実行されることがわかっている場合か、翻訳者が引数の順番を変える必要がないことがわかっている場合 – たとえば文字列に 1 つしか brace の placeholder がない場合や、上記の例のように構文を説明するためのものの場合 – 文字列を no-perl-brace-format とマークしてprintf()を使うことができます:

```
# xgettext: no-perl-brace-format
die sprintf ("usage: %s {OPTIONS} FILENAME...\n", $0);
```

もっと可搬性のある Perl の brace format を使いたいときは、placeholders をリテラルの braces の中に配置します:

```
die __x ("usage: {program} {[}OPTIONS{]} FILENAME...\n",
         program => $0, '[' => '{', ']' => '}');
```

Perl の brace を使った書式文字列は、エスケープするための仕組みを知りません。このエスケープするための仕組みがどのようなものであれ、これはプログラマーに困難な時をもたらし、Perl の brace を使った書式文字列の翻訳を困難にするか、format 命令が実行されるときの実行時の性能を劣

化させるでしょう。このような特別なケースでは、`printf()`のために幸せな時間のほとんどを費やすことになります。

15.5.19 PHP ハイパーテキストプリプロセッサー

RPM mod_php4、mod_php4-core、phpdoc

ファイル拡張子
`php`、`php3`、`php4`

文字列構文 `"abc"`、`'abc'`

gettext の略記
`_("abc")`

gettext/ngettext 関数
`gettext`、`dgettext`、`dcgettext`
PHP 4.2.0 からは `ngettext`、`dngettext`、`dcngettext`

textdomain
`textdomain`関数

bindtextdomain
`bindtextdomain`関数

setlocale プログラマーは、`setlocale (LC_ALL, "")`を呼び出さなければなりません。

必要条件 —

GNU gettext の使用またはエミュレート
使用

抽出プログラム
`xgettext`

位置の書式 `printf "%2\$d %1\$d"`

可搬性 gettext のないプラットフォームでは、上記の関数は利用できません。

po-mode でのマーキング
—

`examples`ディレクトリーの、例`hello-php`が利用できます

15.5.20 Pike

RPM roxen

ファイル拡張子
`pike`

文字列構文 `"abc"`

gettext の略記
—

gettext/ngettext 関数
`gettext`、`dgettext`、`dcgettext`

textdomain
: `textdomain`関数

bindtextdomain
: `bindtextdomain`関数

setlocale `setlocale` function

必要条件 `import Locale.Gettext;`

GNU gettext の使用またはエミュレート
: 使用

抽出プログラム
: —

位置の書式 —

可搬性 gettext のないプラットフォームでは、上記の関数は利用できません。

po-mode でのマーキング
: —

15.5.21 GNU Compiler Collection ソース

RPM gcc

ファイル拡張子
: `c`、`h`

文字列構文 `"abc"`

gettext の略記
: `_("abc")`

gettext/ngettext 関数
: `gettext`、`dgettext`、`dcgettext`、`ngettext`、`dngettext`、`dcngettext`

textdomain
: `textdomain`関数

bindtextdomain
: `bindtextdomain`関数

setlocale プログラマーは、`setlocale (LC_ALL, "")`を呼び出さなければなりません。

必要条件 `#include "intl.h"`

GNU gettext の使用またはエミュレート
: 使用

抽出プログラム
: `xgettext -k_`

位置の書式 —

可搬性 autoconf マクロを使用します

po-mode でのマーキング
: yes

15.5.22 Lua

RPM lua

ファイル拡張子
`lua`

文字列構文

- `"abc"`
- `'abc'`
- `[[abc]]`
- `[=[abc]=]`
- `[==[abc]==]`
- ...

gettext の略記
`_("abc")`

gettext/ngettext 関数
`gettext.gettext`、`gettext.dgettext`、`gettext.dcgettext`、
`gettext.ngettext`、`gettext.dngettext`、`gettext.dcngettext`

textdomain
`textdomain`関数

bindtextdomain
`bindtextdomain`関数

setlocale 自動

必要条件 `require 'gettext'`、または`-l gettext`オプションで lua インタープリターを実行

GNU gettext の使用またはエミュレート
使用

抽出プログラム
`xgettext`

位置の書式 —

可搬性 gettext のないプラットフォームでは、上記の関数は利用できません。

po-mode でのマーキング
—

15.5.23 Java Script

RPM js

ファイル拡張子
`js`

文字列構文

- `"abc"`
- `'abc'`

gettext の略記
: `_("abc")`

gettext/ngettext 関数
: `gettext`、`dgettext`、`dcgettext`、`ngettext`、`dngettext`

textdomain
: `textdomain`関数

bindtextdomain
: `bindtextdomain`関数

setlocale 自動

必要条件 —

GNU gettext の使用またはエミュレート
: 使用、またはエミュレート

抽出プログラム
: `xgettext`

位置の書式 —

可搬性 gettext のないプラットフォームでは、上記の関数は利用できません。

po-mode でのマーキング
: —

15.6 インターナショナライズ可能なデータ

以下は、GNU gettext を使用してインターナショナライズできる、その他のデータ型のリストです。

15.6.1 POT - Portable Object Template

RPM gettext

ファイル拡張子
: `pot`、`po`

抽出プログラム
: `xgettext`

15.6.2 Resource String Table

RPM fpk

ファイル拡張子
: `rst`

抽出プログラム
: `xgettext`、`rstconv`

15.6.3 Glade - GNOME user interface description

RPM glade、libglade、glade2、libglade2、intltool

ファイル拡張子
`glade`、`glade2`、`ui`

抽出プログラム
`xgettext`、`libglade-xgettext`、`xml-i18n-extract`、`intltool-extract`

16 結びの言葉

そして最後に、Native Language Support に関してもっと研究したり読みたい人のために、いくつかの指標を示して GNU gettextのマニュアルを結びたいと思います。

16.1 GNU gettextの歴史

国際対応の重要性やアルゴリズムは、非公式にではありますがが毎日のように数年に渡って GNU で論議されてきました。ときには GNU libcに関係して、あるときは Hurdに関係して、あるいはその他のものに関係してです (誰もはっきりとは覚えていません)。そしてそれから作業は実際に始まりましたが、これは先だっての議論とは独立したものでした。

全ては、Patrick D'Cruze が GNU fileutilsバージョン 3.9.2 の国際対応についてのアイディアそして主導権を得たとき、つまり 1994 年の 7 月に始まりました。彼はそれから保守担当者である Jim Meyering に、国際対応に関する変更をどのようにして正式リリースに含めるかについて尋ねました。最初のドラフトは#ifdefの塊で面食らうような代物でしたので、Jim はもっと良い方法を求めていました。Patrick と Jim はこの分野においての試みや経験を共有していました。そして、結局のところこの作業は GNU に大きなインパクトを与えると感じたので、Jim は標準がどのようなものであるかを知りたくなり、そして彼は Richard Stallman にコンタクトを取りました。Richard はその時点で、glocaleとなるコード全体のデザインについて非常に手早くかつ丁寧に記述しました。

Jim は glocaleを実装し、Patrick や Richard から多くのフィードバックを受けました。もちろん、Mitchum DSouza(catgetsのようなパッケージを記述した) や Roland McGrath, また David MacKenzie や Pinard、Paul Eggert らからのフィードバックもありました。様々な方向性が入り乱れ、しかもそれら全てに互換性があるというわけではなかったため、二、三のテストリリースの後、glocaleは破棄されました。

Jim がある程度距離と時間をおき、そして二人目のパパになった間に、Roland は GNU libcを国際対応させたいと望んでおり、Ulrich Drepper がそのプロジェクトに加わりました。glocaleから作業を始める代わりに、Ulrich は一からコードを書き直しましたが、それは glocaleの作業で明らかになったガイドラインにより一層従う形になっていました。それから Ulrich は以前のフォーラムから新しいプロジェクト用に人員を獲得し、glocaleをまず msgutilsと改称しました。それは後に nlsutilsという名前になり、さらに gettextという名称になりました。これは 1995 年 5 月頃に Richard によって公式に受諾されました。

Ulrich Drepper が 1995 年の四月に GNU gettextで書いたことに言及してまとめとしようかと思います。PO モードを含んだパッケージの最初の公式リリースは 1995 年の 7 月に行われ、そのバージョン番号は 0.7 でした。その他の人々は Ulrich の回りの議論フォーラムで作成された、小さなコードのかけらやテストの結果といった諸々のものを寄贈しました。彼らの名前は GNU gettextに付随する THANKSファイルに納められました。

この作業が進んでいた間、François は最初に半ダースの GNU パッケージに glocaleを、続いて gettextを現在のように適用してプリテストの状態にし、進化するツールを微調整するための効果的なユーザ環境を準備しました。彼はまた、翻訳プロジェクトを組織化し統合する責任をも引き受けました。 Patrick D'Cruze は多くのネイティブランゲージのための 20 の非モデレートなメーリングリスト、そして二つのモデレートされたリスト (一つは全てのチームに直ちに届くもの、もう一つは国際化されたフリーソフトウェアパッケージの全ての自発的な管理者に連絡するためのもの) を作成し管理しました。そしてほぼ一年の間に多くの国々の人々の間で非公式な情報交換が行われ、翻訳者チームが 1995 年 5 月に 発足しました。

François はまた Greg McGary の協力を受け、1995 年の六月に PO モードを書きあげています。これは Ulrich のパッケージに寄贈されました。彼はまた GNU `gettext`の Texinfo マニュアルも寄贈しています。

1997 年に、Ulrich Drepper は GNU libc 2.0 をリリースし、これには関数 `gettext`、`textdomain`、`bindtextdomain`が含まれていました。

2000 年に、Ulrich Drepper は plural form 処理 (`ngettext`関数) を GNU libc に追加しました。その後 2001 年、彼は GNU libc 2.2.x をリリースし、これには完全なインターナショナリゼーションをもつ、最初のフリーな C ライブラリーでした。

GNU libc の General Maintainer としての役割により Ulrich は極めて多忙になったため、2000 年に GNU gettext のメンテナンスを Bruno Haible に譲り渡しました。Bruno もツールに plural form 処理を追加し、UTF-8 と CJK locale のサポートの追加、および PO ファイルを取り扱うための新しいツールをいくつか記述しました。

16.2 参考文献

注意**:** このセクションの文書は時代遅れになっているので、改訂する必要があります。

Eugene H. Dorr(`dorre@well.com`) は *Internationalization Reference List* という国際化対応に関する興味深い文献の保守を行っています。 これは次の場所で入手可能です。

```
ftp://ftp.ora.com/pub/examples/nutshell/ujip/doc/i18n-books.txt
```

Michael Gschwind(`mike@vlsivie.tuwien.ac.at`) は Programming for Internationalisation というタイトルの「良くある質問」(Frequently Asked Questions (FAQ)) のリストを保守しています。この FAQ は異なる言語慣習、 キャラクタセットなどを扱うことが可能なプログラムの記述について論じています。そしてこれは、特に Usenet: ISO 8859-1 及び全てのキャラクタセットエンコーディングに対して適用できます。これは `comp.unix.questions`、`comp.std.internat`、`comp.software.international`、`comp.lang.c`、`comp.windows.x`、`comp.std.c`、`comp.answers`、`news.answers`から定期的に公布されています。この文書は以下にあります:

```
ftp://ftp.vlsivie.tuwien.ac.at/pub/8bit/ISO-programming
```

Patrick D'Cruze(`pdcruze@li.org`) は NLS に関するチュートリアルを書きました。そして Jochen Hein(`Hein@student.tu-clausthal.de`) はそれに関する保守作業を引き継ぎました。この文書は次の場所にあります。

```
ftp://sunsite.unc.edu/pub/Linux/utils/nls/catalogs/Incoming/...
     ...locale-tutorial-0.8.txt.gz
```

このサイトは以下のサイトへミラーリングされています。

```
ftp://ftp.ibp.fr/pub/linux/sunsite/
```

同じチュートリアルのフランス語版は以下にあります。

```
ftp://ftp.ibp.fr/pub/linux/french/docs/
```

ここには、フランス語に訳された Linux 関係のドキュメントもあります。

Appendix A Language Codes

ISO 639 は多くの言語に対して 2 文字のコード、より珍しい言語に対して 3 文字のコードを規定しています。翻訳プロジェクトが言語に対して使用している省略形には、この標準が採用されています。

A.1 Usual Language Codes

一般的に使用される言語にたいしては、ISO 639-1 標準が 2 文字のコードを定義しています。

‘aa’ Afar.

‘ab’ Abkhazian.

‘ae’ Avestan.

‘af’ Afrikaans.

‘ak’ Akan.

‘am’ Amharic.

‘an’ Aragonese.

‘ar’ Arabic.

‘as’ Assamese.

‘av’ Avaric.

‘ay’ Aymara.

‘az’ Azerbaijani.

‘ba’ Bashkir.

‘be’ Belarusian.

‘bg’ Bulgarian.

‘bh’ Bihari.

‘bi’ Bislama.

‘bm’ Bambara.

‘bn’ Bengali; Bangla.

‘bo’ Tibetan.

‘br’ Breton.

‘bs’ Bosnian.

‘ca’ Catalan.

‘ce’ Chechen.

‘ch’ Chamorro.

‘co’ Corsican.

‘cr’ Cree.

‘cs’ Czech.
‘cu’ Church Slavic.
‘cv’ Chuvash.
‘cy’ Welsh.
‘da’ Danish.
‘de’ German.
‘dv’ Divehi; Maldivian.
‘dz’ Dzongkha; Bhutani.
‘ee’ Éwé.
‘el’ Greek.
‘en’ English.
‘eo’ Esperanto.
‘es’ Spanish.
‘et’ Estonian.
‘eu’ Basque.
‘fa’ Persian.
‘ff’ Fulah.
‘fi’ Finnish.
‘fj’ Fijian; Fiji.
‘fo’ Faroese.
‘fr’ French.
‘fy’ Western Frisian.
‘ga’ Irish.
‘gd’ Scottish Gaelic.
‘gl’ Galician.
‘gn’ Guarani.
‘gu’ Gujarati.
‘gv’ Manx.
‘ha’ Hausa.
‘he’ Hebrew (formerly iw).
‘hi’ Hindi.
‘ho’ Hiri Motu.
‘hr’ Croatian.

‘ht’ Haitian; Haitian Creole.

‘hu’ Hungarian.

‘hy’ Armenian.

‘hz’ Herero.

‘ia’ Interlingua.

‘id’ Indonesian (formerly in).

‘ie’ Interlingue; Occidental.

‘ig’ Igbo.

‘ii’ Sichuan Yi; Nuosu.

‘ik’ Inupiak; Inupiaq.

‘io’ Ido.

‘is’ Icelandic.

‘it’ Italian.

‘iu’ Inuktitut.

‘ja’ Japanese.

‘jv’ Javanese.

‘ka’ Georgian.

‘kg’ Kongo.

‘ki’ Kikuyu; Gikuyu.

‘kj’ Kuanyama; Kwanyama.

‘kk’ Kazakh.

‘kl’ Kalaallisut; Greenlandic.

‘km’ Central Khmer; Cambodian.

‘kn’ Kannada.

‘ko’ Korean.

‘kr’ Kanuri.

‘ks’ Kashmiri.

‘ku’ Kurdish.

‘kv’ Komi.

‘kw’ Cornish.

‘ky’ Kirghiz.

‘la’ Latin.

‘lb’ Letzeburgesch; Luxembourgish.

‘`lg`’ Ganda.

‘`li`’ Limburgish; Limburger; Limburgan.

‘`ln`’ Lingala.

‘`lo`’ Lao; Laotian.

‘`lt`’ Lithuanian.

‘`lu`’ Luba-Katanga.

‘`lv`’ Latvian; Lettish.

‘`mg`’ Malagasy.

‘`mh`’ Marshallese.

‘`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; Navaho.

‘`ny`’ Chichewa; Nyanja.

‘`oc`’ Occitan; Provençal.

‘`oj`’ Ojibwa.

‘`om`’ (Afan) Oromo.

‘`or`’ Oriya.

‘os’ Ossetian; Ossetic.
‘pa’ Panjabi; Punjabi.
‘pi’ Pali.
‘pl’ Polish.
‘ps’ Pashto; Pushto.
‘pt’ Portuguese.
‘qu’ Quechua.
‘rm’ Romansh.
‘rn’ Rundi; Kirundi.
‘ro’ Romanian.
‘ru’ Russian.
‘rw’ Kinyarwanda.
‘sa’ Sanskrit.
‘sc’ Sardinian.
‘sd’ Sindhi.
‘se’ Northern Sami.
‘sg’ Sango; Sangro.
‘si’ Sinhala; 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.

‘ve’ Venda.

‘vi’ Vietnamese.

‘vo’ Volapük; Volapuk.

‘wa’ Walloon.

‘wo’ Wolof.

‘xh’ Xhosa.

‘yi’ Yiddish (formerly ji).

‘yo’ Yoruba.

‘za’ Zhuang.

‘zh’ Chinese.

‘zu’ Zulu.

A.2 Rare Language Codes

より珍しい言語にたいしては、ISO 639-2 が 3 文字のコードを定義しています。以下は、その言語を話す人が少なくとも 100 万人いるおうな、生きている言語だけに絞り込んだリストです。

‘ace’ Achinese.

‘awa’ Awadhi.

‘bal’ Baluchi.

‘`ban`’ Balinese.

‘`bej`’ Beja; Bedawiyet.

‘`bem`’ Bemba.

‘`bho`’ Bhojpuri.

‘`bik`’ Bikol.

‘`bin`’ Bini; Edo.

‘`bug`’ Buginese.

‘`ceb`’ Cebuano.

‘`din`’ Dinka.

‘`doi`’ Dogri.

‘`fil`’ Filipino; Pilipino.

‘`fon`’ Fon.

‘`gon`’ Gondi.

‘`gsw`’ Swiss German; Alemannic; Alsatian.

‘`hil`’ Hiligaynon.

‘`hmn`’ Hmong.

‘`ilo`’ Iloko.

‘`kab`’ Kabyle.

‘`kam`’ Kamba.

‘`kbd`’ Kabardian.

‘`kmb`’ Kimbundu.

‘`kok`’ Konkani.

‘`kru`’ Kurukh.

‘`lua`’ Luba-Lulua.

‘`luo`’ Luo (Kenya and Tanzania).

‘`mad`’ Madurese.

‘`mag`’ Magahi.

‘`mai`’ Maithili.

‘`mak`’ Makasar.

‘`man`’ Mandingo.

‘`men`’ Mende.

‘`min`’ Minangkabau.

‘`mni`’ Manipuri.

‘mos’ Mossi.

‘mwr’ Marwari.

‘nap’ Neapolitan.

‘nso’ Pedi; Sepedi; Northern Sotho.

‘nym’ Nyamwezi.

‘nyn’ Nyankole.

‘pag’ Pangasinan.

‘pam’ Pampanga; Kapampangan.

‘raj’ Rajasthani.

‘sas’ Sasak.

‘sat’ Santali.

‘scn’ Sicilian.

‘shn’ Shan.

‘sid’ Sidamo.

‘srr’ Serer.

‘suk’ Sukuma.

‘sus’ Susu.

‘tem’ Timne.

‘tiv’ Tiv.

‘tum’ Tumbuka.

‘umb’ Umbundu.

‘wal’ Walamo.

‘war’ Waray.

‘yao’ Yao.

Appendix B Country Codes

ISO 3166 標準は、多くの国と地域に対して、2 文字のコードを定義します。Translation Project 内で使用される国にたいする略記は、この標準が由来です。

‘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.

‘AX’ Aaland Islands.

‘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 Republic.
'CG'	Congo (Rep.).
'CH'	Switzerland.
'CI'	Côte 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 (United Kingdom).
'GD'	Grenada.
'GE'	Georgia.
'GF'	French Guiana.
'GG'	Guernsey.
'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.

‘IM’ Isle of Man.

‘IN’ India.

‘IO’ British Indian Ocean Territory.

‘IQ’ Iraq.

‘IR’ Iran.

‘IS’ Iceland.

‘IT’ Italy.

‘JE’ Jersey.

‘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.

‘ME’ Montenegro.

‘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.
'RS'	Serbia.
'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 Islands.

‘TD’ Chad.

‘TF’ French Southern and Antarctic Lands.

‘TG’ Togo.

‘TH’ Thailand.

‘TJ’ Tajikistan.

‘TK’ Tokelau.

‘TL’ Timor-Leste.

‘TM’ Turkmenistan.

‘TN’ Tunisia.

‘TO’ Tonga.

‘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 and the Grenadines.
'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.
'ZA'	South Africa.
'ZM'	Zambia.
'ZW'	Zimbabwe.

Appendix C Licenses

このパッケージのファイルは、特定のファイルやディレクトリーに示されたライセンスにより保護されます。以下はサマリーです:

- `libintl`および`libasprintf`ライブラリーは、GNU Lesser General Public License(LGPL)で保護されます、このライセンスのコピーは Section C.2 [GNU LGPL], page 221 に含まれます。
- このパッケージの実行可能ファイルおよび`libgettextpo`ライブラリーは、GNU General Public License(GPL) により保護されます。このライセンスのコピーは Section C.1 [GNU GPL], page 215 に含まれます。
- This manual is free documentation. It is dually licensed under the GNU FDL and the GNU GPL. This means that you can redistribute this manual under either of these two licenses, at your choice.
  This manual is covered by the GNU FDL. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License (FDL), either version 1.2 of the License, or (at your option) any later version published by the Free Software Foundation (FSF); with no Invariant Sections, with no Front-Cover Text, and with no Back-Cover Texts. A copy of the license is included in Section C.3 [GNU FDL], page 230.
  This manual is covered by the GNU GPL. You can redistribute it and/or modify it under the terms of the GNU General Public License (GPL), either version 2 of the License, or (at your option) any later version published by the Free Software Foundation (FSF). A copy of the license is included in Section C.1 [GNU GPL], page 215.

C.1 GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General

Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.
4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose

any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two

goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) yyyy  name of author

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘`show w`’ and ‘`show c`’; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

C.2 GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program

by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the *Lesser* General Public License because it does *Less* to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   a. The modified work must itself be a software library.
   b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
   c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
   d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

      (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply

to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
   b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
   c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
   d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
   e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

   For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components

(compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

   b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.
14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

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

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

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the library's name and an idea of what it does.
Copyright (C) year  name of author

This library is free software; you can redistribute it and/or modify it
under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation; either version 2.1 of the License, or (at
your option) any later version.

This library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301,
USA.
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the library
'Frob' (a library for tweaking knobs) written by James Random Hacker.

signature of Ty Coon, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

C.3 GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Program Index

Option Index

Variable Index

G

L

M

P

T

PO Mode Index

O

P

Q

R

S

T

U

V

W

X

Y

Autoconf Macro Index

General Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y