# GNU gettext for Delphi, C++ Builder and Kylix 1.2 beta

**Lars B. Dybdahl**

**Peter Thornqvist**

**Jacquez Garcia Vasquez**

**Sandro Wendt**

**GNU gettext for Delphi, C++ Builder and Kylix 1.2 beta**
by Lars B. Dybdahl, Peter Thornqvist, Jacquez Garcia Vasquez, and Sandro Wendt

Version 1.2 Edition
Published 2003

# Table of Contents

# Preface

TODO: utf-8 appendix. license chapter. A chapter with special notes for C++ Builder, Kylix, Delphi etc. utf-8 appendix. More screenshots. Include unicode topics.

GNU gettext for Delphi, Kylix and C++ Builder, is a port of the general purpose translation tool named GNU gettext.

There are several po file editors out there. In this manual, poedit will be assumed, because it's the most widely used program for Delphi programmers.

Please note that this manual is for both Windows and Linux users.

# Chapter 1. Introduction

## How does GNU gettext work?

GNU gettext is based on the mo file format (explained later), and everything is based on a function named gettext(), that can look up the English text in such a file, and find the translation which is also in this file.

The simplest (but not easiest) way to put a translated caption on a label would therefore be:

```
Label1.Caption:=gettext('Enter username:');
```

If there is a translation, the translation will be assigned to the label. If there is no translation, or the translation cannot be read for some reason, the text inside the ( ) is used instead.

In order to create an mo file, you write a po file and compile it using the msgfmt compiler. A po file looks like this:

```
msgid ""
msgstr ""
"Project-Id-Version: Delphi 6 runtime translation\n"
"POT-Creation-Date: 2003-03-04 17:49\n"
"PO-Revision-Date: 2003-04-02 17:48+0100\n"
"Last-Translator: Lars B. Dybdahl <lars@dybdahl.dk>>\n"
"Language-Team: Dansk <da@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"License: Freeware\n"

#  This is a comment from the programmer to the translator
#. Programmer's name for it: SInvalidCreateWidget
#: Clx/QConsts.pas:22
msgid "Class %s could not create QT widget"
msgstr "Klassen % kunne ikke oprette en QT widget"

#. Programmer's name for it: STooManyMessageBoxButtons
#: Clx/QConsts.pas:23
msgid "Too many buttons specified for message box"
msgstr "For mange knapper angivet for meddelelsesvindue"
```

Here, `msgid` marks the English original text, and `msgstr` marks the Danish translation. You can write this file using a text editor like notepad, wordpad etc.

As you might already guess, this system can be extended in many ways. The most obvious is that you can have several mo and po files. If an application has more than one file for each language, these are named text domains. The name of a text domain is also the name of an mo file, without the extension. The default text domain is name "default" an must therefore be in the file "default.mo", a compilation of "default.po".

## Creating po files

Instead of typing the whole po file yourself, you can create it by scanning the source code. There are several ways of doing that:

- There is a command line tool named `dxgettext` that can scan Delphi pas, dfm, xfm, rc files
- The `dxgettext` commandline tool is also able to extract all the resourcestrings from an executable file on Windows - like .exe files, .dll files etc.

- The `xgettext` commandline tool can scan C and C++ files.
- On Windows, you can right-click a file folder in Windows Explorer and choose "Extract strings" in the popup menu. From the window that then pops up, you can scan all types of files that can be scanned with the command line tools mentioned above.

These scanners are very advanced. They don't pick up every string they find in a .pas file - only the strings that are surrounded by calls to gettext() and similar functions. This means that in this example, the string will not be extracted:

```
a:='Hello, World';
a:=gettext(a);
```

Actually, the system will give a warning that `a` is unknown. But in this example, it will get extracted:

```
a:=gettext('Hello, World');
```

This means that each single string has to be put in between the ( ) of the function call. In order to save you some typing, an alias for gettext() has been made, and is named _(). The above example can there also be written this way:

```
a:=_('Hello, World');
```

In order to have access to this function, you must include gnugettext.pas in your Delphi or C++ Builder project. This file is part of the installation.

## More gettext functions

Since you can have several mo files, there must be ways to choose between them. The default setting is to use the "default" domain, which means the file default.mo, if present. You can change the default text domain using the `textdomain()` procedure. You can also ask for a single translation from another textdomain like this:

```
a:=dgettext('languagenames','Danish');
```

Here, the string "Danish" will be looked up in languagenames.mo, and if it exists, the translation will be assigned to `a`. If no translation exists, or if the languagenames.mo file doesn't exist for the current language, the string "Danish" will be assigned to `a`.

If you want to use a special language, you can switch the default language using the UseLanguage() procedure, which takes the two-letter language code as parameter:

```
UseLanguage ('da'); // selects Danish
```

You can also specify a language and a country using the two-letter language and the two letter country code like this:

```
UseLanguage ('en_US');  // selects American English
```

The default is to use the language setting in Windows or Linux.

## Resourcestrings

If you have tried the Integrated Translation Environment of Borland Delphi, you will know about the `resourcestring` keyword. It works like this:

```
resourcestring
  msg='Hello, World';
```

```
begin
  ShowMessage (msg);
end.
```

Delphi automatically replaces the reference to msg with a call to the system function `LoadResString()`, which retrieves the string "Hello, World" from the resource part of the executable. The Delphi ITE achieves its translation mechanism by redirecting these fetches to other files.

By including gnugettext.pas in your project, these fetches are replaced with another function, which translates all the strings. The default is, that resourcestrings are translated using the default textdomain, i.e. default.mo. In case you want the system to search other mo files, if the translation isn't found in default.mo just make some calls to the `AddDomainForResourceString`. It is very common to have the Delphi runtime translation in a file named delphi.mo, and this line in the .dpr file to make sure that the Delphi runtime is translated:

```
AddDomainForResourceString ('delphi');
```

Using `resourcestring` is not recommended, because Delphi only handles ansistring this way - not Unicode. But it works.

## Forms

It would not be practical to use _() function calls to assign all strings used in a form. If a string has 200 translatable strings, you would have to type 200 lines of code that assigns strings. Therefore, a procedure named `TranslateComponent` was created. This procedure translates all string properties of a specified component, and all components owned by this component. You can use this to translate an entire form by putting this line into the OnCreate event of a form:

```
procedure TMyForm.FormCreate (Sender: TObject);
begin
  TranslateComponent (self);
end;
```

This single function call replaces lots of _() function calls, which is nice. And since the strings are put into dfm files, which are already extracted, everything works perfectly, except that some string properties should not be translated.

A default setting is that the .Name property of TComponents are not translated. That wouldn't make much sense, and would break your program. You can specify additional string properties, that should not be translated using the TP_GlobalIgnoreClass and TP_GlobalIgnoreClassProperty procedures:

```
TP_GlobalIgnoreClass(TParam);
TP_GlobalIgnoreClassProperty(TField,'FieldName');
```

In this example, no TParams will be translated at all, and the Fieldname property of TField objects will not be translated either. These are global settings and should therefore be placed at a global place in your program. It is recommended to put these lines into your .dpr file, see the Section called *Sample.dpr* in Appendix B>. You can also specify a custom handler for a class like this:

```
TP_GlobalHandleClass (TSpecialClass,Myhandler);
```

You can read more about typical ignores in Appendix H>.

# Chapter 2. Action

## Localize your first application

Assuming that you have created a simple application in Delphi, this section will show you how to localize it.

First, you must add `gnugettext.pas` to your project. It is recommended to copy this file to your project directory, making it part of your application.

In order to translate your forms and datamodules, you must add `gnugettext` to the uses-clause of all units that have a form or datamodule, and put this line into the OnCreate event of your forms and datamodules:

```
TranslateComponent (self);
```

In this line, `self` refers to the form or datamodule to which this event belongs. The `TranslateComponent` procedure then translates all string properties of all components on the form or datamodule. Please note that all subcomponents are translated - for a TDataset component this includes the TField subcomponents etc.

Unfortunately, some string properties will raise an Exception or create unwanted side effects if they are translated. For instance, if you translate the IndexFieldName property of a TClientDataset to something that is not the name of an index, it will raise an Exception. In order to instruct the `TranslateComponent` procedure that it should not translate certain string properties, you should add procedure calls like the ones specified in Appendix H> just before the `TranslateComponent(self)` call in your main form or in the .dpr file. You can see the source code for a small, localized application in Appendix B>

Now, you must ensure that all strings inside your source code are translated properly. If you have a line like this:

```
ShowMessage ('Hello, World');
```

Then you must add `_()` around the string, like this:

```
ShowMessage (_('Hello, World'));
```

Now your program is internationalized, but it hasn't been localized, yet. This means that your program can run in another language, if a translation would be present, but we didn't make a translation, yet. In order to make a translation, we have to get a list of messages first.

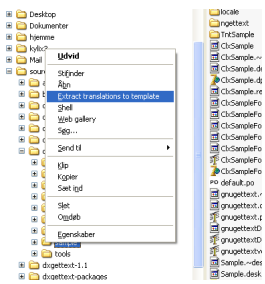



**Figure 2-1. Explorer integration of string extraction**

In order to get this list, click with the right mouse button on the file folder that contains your Delphi source code, and choose "Extract translations to template", just as you can see it in Figure 2-1>.

This will create a file named default.po with all the texts to be translated. Because it has no translations in it, it is named a "translation template". You should download poEdit[1] and use this program to translate the messages in the po file.

When you have translated the po file, click with your right mouse button on the filename in the Windows Explorer, and choose "Compile to mo file". This will compile the po file and generate the binary mo file needed by your application. If your application is located in `c:\my\program\path\myprogram.exe`, you must put the default.mo file in `c:\my\program\path\locale\##\LC_MESSAGES\default.mo`. In this path, ## represents the two-letter language code that you can find in the Section called *ISO 639 language codes* in Appendix F>.

That should be it - your application now uses the translation when the Windows settings corresponds to the two-letter language code that you chose.

## New program versions, old translation files

After you have made your first translation, you will find that your old translation file needs to be updated for this new version. This is the procedure:

Extract strings from the new source code. The po file that was generated by this, is called the "template", because it determines which messages that the final translation must contain. You can then update the old translation file to the new translation template using the merge functionality. Linux users have to use the `msgmerge` program. Windows users both have a command line tool, but can also click with the right mouse button on the old translation file in Windows Explorer and choose "Merge template". Here, you can pick the template file and do the merge.

The result is a file that contains the messages of the template, with the automated comments from the template and the translations from the translation files.

Since the string extraction tools always extract in the same order, and since the merging will preserve the order from the template, po files are very suitable to be stored in source code version control systems like CVS, FreeVCS, SourceSafe, PVCS, TeamSource etc. Just make sure that you merge with the template before putting a new copy into the version control system.

## Create a single language application with localized runtime library

A typical problem with Delphi is to create a program in a non-English language, because the runtime library is in English. You can easily put a chinese caption on a button, but you cannot easily make the "Division by zero" message turn chinese, because that message is hidden deeply in the Delphi runtime library. With GNU gettext for Delphi, you can easily solve this problem:

1. Install GNU gettext for Delphi
2. Add `gnugettext.pas` to your project.
3. Get the `default.po` file for the Delphi runtime libraries and translate it to your language. Maybe there is already a translation for your language online[2].
4. Compile the default.po file to a default.mo file by clicking on it with the right mouse button and choose "compile to .mo file".
5. Create a `locale\LL\LC_MESSAGES` subdirectory to where your exe file is, where LL is the two letter ISO 639 language code (see the Section called *ISO 639 language codes* in Appendix F>) of the language that your program uses (da for danish, de for german etc.) and put your `default.mo` file there.

6. Give your translations to us, so that other people may use your translation.

In order to have one .exe file that contains everything, including the translations, you can now click with the right mouse button on the executable, and choose "Embed translations". This will append the `default.mo` file, that is in the `locale\LL\LC_MESSAGES` subdirectory into the .exe file. Please note that you have to do this every time that you have compiled and generated a new .exe file.

## Uses clauses

The order, with which Delphi executes the initialization sections of your units depend on the order that the units are included in your application. A typical application has a .dpr file that looks like this:

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Here, the initialization section of the `Forms` unit will be executed before the initialization section of the unit named `Unit1`. All units that are used by the `Forms` unit will also have their initialization section executed before the section of `Unit1`.

Not all units have an initialization section, but the `gnugettext.pas` file does. It detects the language, starts the resourcestring translation etc. Therefore, all resourcestrings, that are fetched before this intitialization section has been run, will not be translated, but all that are fetched afterwards, will. If you include the `DBClient` unit, and this unit fails to initialize because there is no `MIDAS.DLL`, then it will show an error message in English if gnugettext was later in the uses list, but it will show it in the local language if gnugettext was first.

A translated application's .dpr file could look like this:

```
program Project1;

uses
  gnugettext in 'gnugettext.pas',
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  // Add extra domain for runtime library translations
  AddDomainForResourceString ('delphi');

  // Force program to use Danish instead of the current Windows settings
  UseLanguage ('da');

  // Put ignores on the properties that cannot be translated
  TP_GlobalIgnoreClassProperty (TMyComponent1,'property1');
  TP_GlobalIgnoreClassProperty (TMyComponent2,'property2');
  TP_GlobalIgnoreClassProperty (TMyComponent3,'property3');
  TP_GlobalIgnoreClassProperty (TMyComponent4,'property4');
```

```
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

This works very, very well for most situations, but if you want translations to start as early as possible, your .dpr file should look like this:

```
program Project1;

uses
  gnugettext in 'gnugettext.pas',
  gginitializer in 'gginitializer.pas',
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  // Put ignores on the properties that cannot be translated
  // These just have to be placed before the first call
  // to TranslateComponents()
  TP_GlobalIgnoreClassProperty (TMyComponent1,'property1');
  TP_GlobalIgnoreClassProperty (TMyComponent2,'property2');
  TP_GlobalIgnoreClassProperty (TMyComponent3,'property3');
  TP_GlobalIgnoreClassProperty (TMyComponent4,'property4');

  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

In this example, gginitializer sets all the necessary settings in `gnugettext.pas`. It could look like this:

```
unit gginitializer;

interface

implementation

uses
  gnugettext;

initialization
  // Add extra domain for runtime library translations
  AddDomainForResourceString ('delphi');

  // Force program to use Danish instead of the current Windows settings
  UseLanguage ('da');

  // Do not put TP_GlobalIgnoreClass* statements here, because
  // that would require this unit to use other units than gnugettext,
  // and then all these units would have their initialization
  // section executed before this one.

end.
```

In this example, the initialization section of gginitializer will be run before the initialization sections of units like `Forms` and `Unit1` are run.

## Solving ambiguities

Sometimes it may happen that the same English message should be translated to two different messages in another language. A very wellknown ambiguity is the word "free", which can be like free beer or free speech. For instance, if you have two radiobutton groups describing a piece of software, and the first is about the software price and the second is about the software type, both may include the choice "free". The software price "free" would be translated to "gratis" in Danish, and the software type "free" would be translated to "fri" in Danish.

It is the programmer's responsibility to ensure, that one msgid only can result in one translation. Sometimes this fails - and then the translator has to report back that something has to be changed. There are several ways to solve this

### Change words

The first solution is to change the words. Instead of the price option "free" you could give the option "$0", or you could write "free software" in one of the choices. Do a brainstorm and pick the best.

### Adding spaces

You could add a space to one of the strings. This would give the translations "free"->"fri" and "free "->"gratis". Spaces are not visible in a radiogroup. If you use this technique inside the source code, you may want to remove the space before the space. In this case, you should provide a comment to the translator that the space needs to be preserved:

```
// Preserve the initial space in the translation.
dataset.FieldByName('Name').DisplayLabel:=copy(_(' Name'),2,maxint);
```

A good translator should always preserve leading and trailing spaces, but sometimes it is useful to give the translator a hint anyway.

### Using domains

In some occasions, the solution to ambiguities can be to use several text domains. In the case with two radio button groups, you would exclude one of them from `TranslateComponent()` with `TP_Ignore()`, and then translate it separately afterwards using `TranslateComponent(MyRadioButtonGroup,'separatedomain');`.

### Using trailing comments

Some people using trailing comments to include a comment within the msgid:

```
function stripafterdot(s:widestring):widestring;
var
  p:integer;
begin
  p:=pos('.',s);
  if p<>0 then
    s:=copy(s,1,p-1);
  Result:=s;
end;

myfield.DisplayLabel:=stripafterdot(_('Name.Displaylabel for the field named "name"'));
```

This solution requires the translator to know about this notation.

## Plural forms

The `ngettext()` function is a very powerful function for handling plural forms. In order to understand this function, you should first understand the `gettext()` function in the Section called *function gettext(msg:widestring):widestring;* in Appendix A>.

A well known problem is to specify the number of files in a list of filenames. With this function, it can be done like this:

```
LabelCount.Caption:=format(ngettext('%s file','%s files',filelist.Count),[filelist.Cour
```

If no translations are available, the `ngettext()` function will return '%s file' if filelist.Count=1, and it will return '%s files' otherwise. The `format()` function will then put the actual number of files in place of the %s, and the result will be something like '0 files', '1 file', '2 files', '3 files' etc.

If you would want to translate this to french, the entry in the po file should look like this:

```
msgid "%s file"
msgid_plural "%s files"
msgstr[0] "%s fichier"
msgstr[1] "%s fichiers"
```

The idea with ngettext is, that it doesn't just translate "%s file" to "%s fichier", but it takes into account, that the french use numbers differently. The English say "0 files", but the french use singular to describe the value zero: "0 fichier". So in the above example, the french version would be: '0 fichier', '1 fichier', '2 fichiers', '3 fichiers'...

Some languages are even more complicated. In Polish, there are three plural forms, and the translation would look like this:

```
msgid "%s file"
msgid_plural "%s files"
msgstr[0] "%s plik"
msgstr[1] "%s pliki"
msgstr[2] "%s plikÃ³w"
```

When counting files, it will become: '0 plikÃ³w', '1 plik', '2 pliki', '3 pliki', '4 pliki', '5 plikÃ³w'. Confused? Don't be. Just use `ngettext()` wherever your text depends on a number, and the translator will provide the correct translations.

> **Not all tools handle msgid_plural well:** Please note, that not all tools handle msgid_plural well. This includes poEdit and KBabel. If a po file contains msgid_plural translations, you should use a text editor to edit it/translate. A good text editor for po files is UniRed[3].

> **How does it work?:** The ngettext and dngettext functions use gettext(singular+#0+plural) to get a #0-separated list of plural forms.

Because some tools don't handle msgid_plural forms well, you should put all plural forms translations into a separate po file. You can do this using `dngettext()`, which is equivalent to `ngettext()` except that it takes a text domain name as first parameter:

```
LabelCount.Caption:=format(dngettext('plurals','%s file','%s files',filelist.Count),[fi
```

In this case, the source code string extraction will put the translation into a file named `plurals.po`, and the `dngettext()` function will retrieve the translations from `plurals.mo`. You can then ask the translator to use notepad to translate the

plurals.po file. Notepad is not always very handy, but it's surely compatible with the msgid_plural notation.

## Database applications

### Preventing unwanted translations

When you create a database application, there will be a lot of component properties that you don't want to be translated when using Translatecomponent(). Typically, the field names, table names and even database names of a database have meaningful translations, and the translation file from the translator may include translations for field names. Also, SQL statements should not be translated. They will get extracted, but if the translator modifies them, it would most likely break your program if this translation would be applied.

Therefore, it is very important, that you consult Appendix H> to see what ignores you should add to your application. Make sure that these ignores are executed before the first call to TranslateComponent(). The list in that appendix may not be complete - especially not if you use database components that are not mentioned. Therefore you must have a look at your components and make sure, that all properties, that should not be translated, have a corresponding TP_GlobalIgnoreClassProperty() call.

### DisplayLabel property explained

Delphi is very good for creating a single-language database application, fast, if the application only needs one language. A typical single-language application uses field names that are easily understood, like "Name", "Address" etc. Let's assume that you use a query component with an SQL statement like this:

```
select * from customer order by Name
```

In this case, the name of the Name field of the customer table will propagate through all components, will become the column heading of a TDBGrid etc. Your grid column for names will read "Name". This is desired in an English language application, but absolutely not in all other languages.

The solution is to modify the .DisplayLabel property of the TField components, that your dataset possesses. Every TDataset descendant has a .Fields[] property that refers to the fields in the dataset, and all these field components are descendants of TField, which has the TField.DisplayLabel property.

Delphi provides two ways to modify this DisplayLabel property: At runtime and at design time, which leads to two different ways of handling the localization of it.

### Setting displaylabel at runtime

The runtime assignment solution is to assign a displaylabel at runtime like this:

```
procedure TForm1.Query1AfterOpen(DataSet: TDataSet);
begin
  Dataset.FieldByName('Name').DisplayLabel:=_('Name');
end;
```

This way, the field names are always the same in the database and the entire application, but the user will see a localized name. This works for all dataset type components, including table components, TClientDataset etc.

### Display label at design time

Creating design-time DisplayLabel properties goes like this:

- Double-click the dataset component. This brings up the field list window.
- Click with your right mouse button on the field list window and choose "Add all fields". This requires your dataset to be able to actually fetch data from the database at design time, but will add all the fields of the dataset to the field list window.
- Make sure that the form (or datamodule) that the dataset component is on, is translated using `TranslateComponent()` before the dataset is opened. This means that the dataset needs to be closed by default, and that you have to open the dataset at runtime using `.Open` or `.Active:=True`.

By doing this, the field names will be present in the DFM files, and will therefore be extracted for the translator to be translated. Now, `TranslateComponent()` will translate the `TField.DisplayLabel` values:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  TranslateComponent (self);
  Query.Open;
end;
```

### Multiple field name translations

Sometimes it is very handy to have field names translated to something else than what you put into the `.DisplayLabel` property. For instance, you may want to have a short version for exported ASCII files, another version for exporting XML files, one version for reports, and again another version for the `.DisplayLabel` property.

This is easily done by using several po files (which is the same as multiple text domains). The part of your program that writes a column header to an ascii file might look like this:

```
write (mytextfile,dgettext('fieldnames',myfield.FieldName));
```

This will find the field name in the `fieldnames.mo` file and output the translation. The `fieldnames.po` template can be written by hand (using notepad), or sometimes be extracted from a datamodule. Often, the number of field names is so low that the quickest solution is to write the po file by hand.

## Translation repositories

It is often very useful to create one po file that has all the translations from all other po files included. For instance, if you are the producer of software for chemical laboratories, and you have 5 different products, there will probably be a lot of common translations for the different products, and it would save you a lot of work if you don't have to translate it all again for each new product.

In order to merge several po files, you can use the `msgcat` tool:

```
C:\translations\da>msgcat -o result.po -t utf-8 --use-first delphi7.po kylix3.po

C:\translations\da>
```

This set of parameters will put the result into `result.po`, use utf-8 encoding, and only take the first translation from each po file.

## Notes

1. http://poedit.sf.net/
2. http://dybdahl.dk/dxgettext/translations/
3. http://unired.sf.net/

# Chapter 3. Project management

## Introduction

Localizing an application is not a simple task. In Germany, the first floor is the first one above ground. In USA, it's at ground level. How do you put that into a database? If you make an integer field named `floor` that contains `0` for ground level and `1` for the first floor above ground, it is very easy to understand in Germany, but you'll have to modify the user interface quite a bit to make the same data accessible by Americans in a way that they understand easily.

The GNU gettext system only helps you with translating messages, but this can go wrong, too. For instance, the Math unit in Delphi has functionality to convert between different units. The names of these units can be translated, but if you translate two unit names to the same thing, all programs that use the Math unit will generate an exception during initialization. Actually, this already happened. The first Danish translation of the Borland Delphi 7 runtime library had "Hektar" as the translation for both "SquareHectometers" and "Hectares". It's the same thing, and the correct word is "Hektar", so why not use it? Now that the program doesn't work, whose fault is it?

The answer is very simple: It's the programmer's fault. He didn't live up to the responsibility of ensuring that the translator couldn't do anything to break the program. It might take unnecessary extra programming to ensure this, but it can be very, very difficult to debug the program if this is not ensured.

Let's take another example: The translator translates the application, but the users cannot find out how to operate the program. The English version works well and users know how to operate the English version. Whose fault is this?

Here, the answer is more complex: it might be the translator that made a bad translation, but it could also be the programmer that designed a user interface that is hard to translate. Often, user interfaces use concepts that can be described with one word in the programmer's own language, but if it takes 10 words to describe the same concept in another language, people that only know that other language might not understand the concept. For instance, zip-files could be represented by a zipper in English (Windows XP does that), but in other languages thsi relationship makes absolutely no sense. In Danish, zip-files are named "zip-filer", and zipper is "lynlaas". There is absolutely no relationship between "zip-filer" and "lynlaas", and whoever invented that icon definitely didn't think of internationalization.

The leader of an internationalized project has to:

- Ensure that all programmers understand the concept of internationalization.
- Ensure that translators can give feedback to the programmers in order to ensure that everything is localizable.
- Ensure that there is a release procedure with beta-testing for each language. Having debugged the program in one language doesn't mean that it is bugfree in other languages, too.
- Ensure that the beta-testers know, that they must give feedback on the translation, too.

## Coordinating translations

Besides having programmers and translators, it is very important that somebody is appointed to manage the localization process. This person must ensure correct archiving of translations and other po files, and ensure that the translations are tested. Typically, this assignment is given to a programmer with localization knowledge.

## The translator

It can be difficult to find a good translator. A good translator understands the application, the local market and is good at finding translations for concepts that maybe doesn't make a lot of sense in the native language. It is also important that multiple applications are translated the same way.

There are many small companies out there that have specialized in translating software. Use one of those - it pays off. One of the good things about GNU gettext for Delphi, Kylix and C++ Builder, is that the file format is standardized and known by most translation companies.

# Chapter 4. Experienced programmers' topics

## Determinism and responsibility

It is very important for the software development process, that the programmer decides, what gets translated, and what doesn't. Also, it is important for the programmer to ensure, that the translator cannot break the application, no matter how bad the translations are made.

GNU gettext is based on function calls (gettext, dgettext etc.), comments to the translator and a proper localization release procedure.

The system doesn't translate anything that isn't put through the `dgettext()` function in a way or the other. The programmer fully controls what gets translated, and what doesn't, and has the responsibility to ensure, that everything that gets translated, can be translated to virtually anything without breaking the program. For instance, if the caption of a label on a form is made translatable by the programmer, the translator can only change the look of the label. But if the component name of a label would be translatable, the translator could break the program by translating the name of a label to something that is already the name of another component.

In order to get a program translated, the programmer must provide information to the translator about what gets translated and where to find it. He also needs to ensure, that the texts are unambigious, that one msgid cannot be translated into two different things in another language. Since it may not always be clear to the translator, in which context a message is used, the programmer can provide comments to the translator, even access to the source code locations, if the translator is able to read source code. The comments should also make the translator able to run the program with his/her translations, and find the place where a particular text can be found inside the program.

The translator is often also a beta-tester for a specific language. The translator is often the only one that is able to control the program while running in the other language, and is often also the only one internally in the organization that is capable to see if labels are put correctly, translations are made correctly etc. GNU gettext helps out here by making the translator able to apply his or her translation without involving the programmer.

## Text domain management

Different text domains are basically different po files. Usually, one application uses one text domain, but often, several applications share one text domain. Sometimes it is necessary to use an extra text domain for a special purpose.

As a project leader, it is important to know, that you can always easily split a project into two parts. The key is to create two template files instead of one. Just merge the old translation file with the new templates, and you have two new, smaller translation files.

It is a bit trickier to assemble two smaller subprojects into one big translation project with just one file. There might be messages, that were translated differently in the two projects, and this has to be taken care of. Several tools can assemble two po files, and these include `msgcat` and `msghack`. See Appendix D> for more information.

## The better alternative to resourcestring

Instead of using resourcestrings, there is a better alternative:

```
ShowMessage (_('Action aborted'));
```

The _() is a pascal function inside gnugettext.pas, which translates the text. It returns a widestring, unlike resourcestring, which is limited to ansistring. You can use _() together with ansistrings as you like, because Delphi automatically converts widestrings to strings when needed. Another benefit of this is that you can write comments, that the translator can use to make better translations:

```
// Message shown to the user after the user clicks Abort button
ShowMessage (_('Action aborted'));
```

You can also write the comment in the same line:

```
ShowMessage (_('Action aborted')); // Message to user when clicking Abort button
```

But only the // style comment is supported - you cannot use { } or (* *) comments for this purpose.

Good comments normally lead to good translations. If the translator has a copy of the source code, poedit and kbabel can both show the location in the source code to the translator. This makes sense with _(), because the translator might get a good idea, what this is about, even if the translator isn't a programmer.

In other words, there are many reasons to use _() instead of resourcestrings. If you create a new application, don't even think about resourcestrings - just go directly for the _() solution.

## Debugging

You will typically experience two types of errors:

- Something is not translated
- An error occurs when using `TranslateComponent()`

The first item often happens because the translation files (.mo files) are not present for the current language, not placed where they are expected to be etc. The second occurs because a property of some component should not be translated. When possible, you should get an Exception that is easy to understand, but sometimes you don't. This section is about finding the error anyway.

The `gnugettext.pas` file has a logging system for debugging built-in. You activate it by defining the conditional define `DXGETTEXTDEBUG`. You can also find the first occurence of this string in `gnugettext.pas` - here you will find the following code:

```
// DEBUGGING stuff. If the conditional define DXGETTEXTDEBUG is defined, it is activated.
{ $define DXGETTEXTDEBUG}
{$ifdef DXGETTEXTDEBUG}
const
  DebugLogFilename='c:\dxgettext-log.txt';
{$endif}
```

One way to activate the debugging log is to change

```
{ $define DXGETTEXTDEBUG}
```

to

```
{$define DXGETTEXTDEBUG}
```

by removing the space. As you can see, you can also here modify the location where the log-file is written to.

The log-file contains a lot of information. At the beginning, you will find very useful information about what settings the system uses:

```
Debug log started 21-08-2003 10:32:08

UseLanguage(”); called
LANG env variable is ”.
Found Windows language code to be ’da_DK’.
Language code that will be set is ’da_DK’.
Plural form for the language was not found. English plurality system assumed.

Text domain "default" is now located at "C:\source\sf\dxgettext-devel\dxgettext\sample\
Changed text domain to "default"
Globally, the NAME property of class TComponent is being ignored.
Globally, the PROPNAME property of class TCollection is being ignored.
Extra domain for resourcestring: delphi
Globally, class TFont is being ignored.
```

In this example, we can see that the program was running on a Danish language Windows, which uses the same plurality system as English. It also tells us where it will look for .mo files, and what ignore settings were specified. When `TranslateComponent()` is used, it looks like this:

```
============================================================================
TranslateComponent() was called for a component with name FormMain.
A retranslator was created.
----------------------------------------------------------------------------
-
TranslateProperties() was called for an object of class TFormMain with do-
main "".
Reading .mo data from file ’C:\source\sf\dxgettext-devel\dxgettext\sample\locale\da_DK\
Found in .mo (default): ""->"Project-Id-Version: PACKAGE VERSIONPOT-Creation-
Date: 2003-02-16 21:36PO-Revision-Date: 2003-02-17 23:01+0100Last-Translator: Lars B. D
dahl <lars@dybdahl.dk>Language-Team:  <>MIME-Version: 1.0Content-Type: text/plain; cha
8Content-Transfer-Encoding: 8bit"
GetTranslationProperty(CONTENT-TYPE: ) returns ’text/plain; charset=UTF-
8’.
Found in .mo (default): "GNU gettext sample application"->"GNU gettext eksempel"
Found in .mo (default): "Click me"->"Klik mig"
Found in .mo (default): "Click me"->"Klik mig"
----------------------------------------------------------------------------
-
This is the first time, that this component has been translated. A re-
translator component has been created for this component.
============================================================================
```

The log simply specifies exactly the filename from which translations are fetched, and it also specifies exactly, which strings are translated to what using which text domain. The "retranslator component" is a component that is added to the form to make it remember the untranslated properties, in case you want to do a language switch at runtime.

The first time that the application wants to translate an "OK" button, it looks like this:

```
Loaded resourcestring: OK
Translation not found in .mo file (default) : "OK"
Reading .mo data from file ’C:\source\sf\dxgettext-devel\dxgettext\sample\locale\da_DK\
Found in .mo (delphi): ""->"Project-Id-Version: Delphi 7 RTLPOT-Creation-
Date: 2003-03-02 18:54PO-Revision-Date: 2003-03-03 00:31+0100Last-Translator: Lars B. D
dahl >lars@dybdahl.dk<Language-Team: Dansk >da@li.org<MIME-Version: 1.0Content-
Type: text/plain; charset=UTF-8Content-Transfer-Encoding: 8bit"
GetTranslationProperty(CONTENT-TYPE: ) returns ’text/plain; charset=UTF-
8’.
Found in .mo (delphi): "OK"->"O.k."
```

Here you can see, that it first searches `default.mo` for a translation, but doesn't find one. Because it's a resourcestring translation, and because we specified the "delphi" textdomain to be searched for resourcestrings, it decides to try out delphi.mo. This file has not been opened, yet, and therefore the file is opened at this point, and the full filename is written to the log file. The first action when opening a new .mo file is to check wether the Content-Type is set to use utf-8. Once it found out that this is the case, it looks up "OK" in the delphi.mo file, and finds the Danish "O.k." translation.

Log-files can be huge if your program runs for a long time. If that happens, load the logfile into an editor that is capable of handling huge files, and search for the keywords that you saw in this section.

If your program breaks because a string property is translated that shouldn't, try to search for the Exception error message in the log file (error messages are translated, too, and are also mentioned in the log file). The last translation mentioned before that error mesage is probably the one that made your program fail.

When you have identified a property that should not be translated, you can either specify it globally that it should not be translated, by calling `TP_GlobalIgnoreClassProperty()`, or you can disable it only for the next call to `TranslateComponent()` using `TP_Ignore`.

## Directives

It is possible to control the string extraction from Pascal source code using directives, very much like the compiler directives. Currently, the directives only support extraction of string constants defined using the `const` keyword, into a specified text domain:

```
{gnugetext: scan-all [text-domain='domain name'] }
.
. in this section the string constants will be extracted
.
{gnugettext: reset }
```

`{gnugettext: scan-all }` is named the 'start directive' in this document. `{gnugettext: reset }` is named the 'end directive' in this document.

If a start directive exists, the end one must exist in the same file. The directives are local to a file. Several start directive can exists before the end one and is used to change the target domain. This directive actually works for constants only! Initialized variables are not taken in care. The domain name, if present, must be enclosed by single quote. If the domain name include a single quote (but remember that the domain name will becomes a file name), it must be doubled. There are no check on the domain name. It is assumed that you know what you type!

```
{gnugettext: scan-all text-domain='toto' }
Const
  a = 'toto';
{gnugettext: scan-all text-domain='titi' }
  b = 'titi';
{gnugettext: reset }
```

**Example::** In this example, the msgid 'toto' will be put into `toto.po` and 'titi' will be put into the file named `titi.po`.

It is the plan to extend the directive system in the future to disable/enable string extraction etc.

# Chapter 5. Advanced topics

## Migrating from the ITE to dxgettext

This chapter was written by Peter Thornqvist.

### Introduction

Although I've been somewhat involved in the development of dxgettext - mainly donating a couple of tools to convert translations from ITE to dxgettext - I had not myself been ready to take the final step in moving any larger application over. It was both a matter of lack of time but also a hesitation about the usefulness of the po file format and the possible loss of information.

Well, one day the ITE gave up on me (for the umpteenth time) and I just couldn't get it to compile my project anymore. It complained about "ancestor not found". Checking out the files from VCS and even going back in history didn't solve it. I was stranded with a non-compiling project and I had no solution how to get it to work again.

It was time for me to get down and dirty with dxgettext and this is the report on how I did it, what I had to change and how it all worked out in the end.

### The project

The application I am going to migrate is called EQ Plan and is a graphical planning tool mainly for manufacturing companies (see http://www.timemetrics.se/ for details and trial downloads). As far as localization is concerned, this is a mid-sized application consisting of about 45 forms, 5 or so frames (I avoided frames since I knew the ITE didn't like them) and about 10-15 additional files. Since I was using the ITE, all strings (except property values) were declared as resourcestrings and most of them was located in a single unit. Additionally, the application uses a wide variety of components and controls; some third-party components from JVCL and KWizard and a couple of custom made components (like the Gantt-chart and a VS.Net style treeview). The application is not Unicode enabled, limiting it's usability to western Windows versions. All string in the program are in English and we have translated it to Danish and Swedish. Translation to Norwegian and German is planned but not started (as of this writing), making this a good time to do the change.

### Planning

I didn't do much planning because the task was pretty straightforward but I at least made the following to-do list:

- Isolate any remaining strings used in the program and convert them to resourcestring. Since I already used resourcestrings and the program isn't Unicode anyway, I decided to stick with resourcestrings, although I probably would have used the _() syntax if this had been a new program (see the dxgettext documentation for an explanation of "_()")
- Extract any existing ITE translations from the dfn and rc files in the language subfolders.
- Since all forms already inherited from a common ancestor, I decided to use this ancestor to implement as much as possible of the basic translation functionality and add any special handling in each derived form or utility unit as needed.
- After successfully moving the translation, find all components and/or units that have untranslated strings even when running a localized version of the program.

Add special handling of the components as necessary and add any missing strings as resourcestrings.

- Since gnugettext doesn't localize component bounds (top, left, width, height), go through all forms and check that texts and label alignments and the like looks good in all languages.
- Create installation packages for the language file(s).
- Implement a language switch functionality inside the program so the end-user easily can change the used language.
- Test the new translations on as many systems as possible, i.e. different Windows versions as well as different language versions of Windows.

### Tools I needed (and used)

There are several tools I need to be able to migrate my application. Specifically, I used the following:

- dxgettext (or actually the shell integration) to extract strings and properties from the application sources to create the template po file
- dfntopo (include din dxgettext distribution) to extract the translations from the dfn and rc files and update the language specific po file.
- poEdit (http://poEdit.sourceforge.net/) to view the po files, provide additional and compile the mo file (the binary translation file)

### Doing the conversion

I started with extracting the po template from my source folder: I right-clicked on the source folder and clicked on "Extract translations to template". A file named default.po was created in the selected folder and it contained all the original strings found by dxgettext. Since there were a lot of strings, there was no way I could determine how successful dxgettext had been. I just had to trust it for the time being. One thing that I didn't do at this stage (because I wasn't aware of it) but that I recommend you to do is this: Run the msgmkignore.exe on the default.po file. This will create a new file 8typicaly named ignore.po) with all the strings that probably shouldn't be in default.po. This is strings like numbers, component names, font names etc. It does an amazingly good job too: all the strings it found in my sources were such that I did actually want to remove them. To actually remove the strings from default.po, I ran the command line tool mskremove.exe. Now the (first) template was ready for use.

Since I now had a template, I needed to retrieve as many of the available translations as possible from the ITE. I certainly didn't want to translate all this again. I made this simple for myself. I copied the dfntopo.exe program along with the default.po file into each of my ITE language folders (where the dfn files are located) and ran dfntopo.exe from a command-prompt. I used the -f option to force dfntopo to add all new translations that it couldn't find in the po file. Although this will probably reinsert the items we just removed with msgremove, I wanted to make sure that all the translations I had, made it over into the po file. I can always run msgremove again at a later time.

So, I now have a default.po in Danish and one in Swedish. I also have the template default.po file (the one without any translations) in the original source folder. This means that I am now actually ready to test the translations with gnugettext and this after only about an hours work! Of course, I still have to enable my application to read the translations and this is a three step process:

- Create the correct subfolder structure for my application folder and put the translated default.po files there.
- Open each po file with poEdit and save it to create the mo file.
- Add gnugettext.pas to my application so it can read the mo file (almost) automatically

The subfolder structure on my machine looked like this after I've added the Swedish and Danish folders:


Each of the translated default.po files are located in the language specific LC_MESSAGES folder. As you can see, the folder structure is somewhat convoluted and uses some predefined language identifiers (these names are actually standard ISO639 identifiers, sometimes combined with ISO3166 country abbreviations) to enable gnugettext to automatically select the correct language file (this is based on the users current locale). I suppose one could modify gnugettext.pas to, as an example, place all the language files in one folder and call them se.po, da.po etc instead but I decided to use the default settings. If the po/mo files are correctly placed and it still doesn't work at run-time, at least I know that it's something else that isn't working.

Now I could open and save the po files with poEdit. This creates an mo file - a compiled binary version of the po file - in the same folder as the po file and it's actually this file that gnugettext needs. The po file is only used to allow humans to read and edit the content. Now I was ready to edit my project to read the translations.

I had some errors when I opened the po files with poEdit: TABS (#9) had been translated to \x9\ but poEdit expected \x9 and embedded double-quotes weren't properly escaped (should have been \" instead of just "). The quote escaping I fixed by modifying the source of dfntopo and the \x9\ problem I fixed manually by doing a search and replace on the po files.

### Common ancestor forms are good!

Since all my forms already had a common ancestor form (a good coding practice, by the way: you loose nothing and can gain a lot), I decided to add as much gnugettext functionality to that form. The other forms would then inherit the behavior and I would have to add the code everywhere. Additionally, any new forms I decide to add in the future will also have the functionality automatically. So I added the following overridden AfterConstruction code to my ancestor form:

```
var
  AlreadyDone:boolean = false;

procedure TfrmGnuGT.AfterConstruction;
begin
  inherited;
  if not AlreadyDone then
```

```
  // this should only be done once for the whole app
  begin
    TP_GlobalIgnoreClassProperty(TAction,'Category');
    TP_GlobalIgnoreClassProperty(TControl,'ImeName');
    TP_GlobalIgnoreClassProperty(TControl,'HelpKeyword');
    TP_GlobalIgnoreClass(TMonthCalendar);
    TP_GlobalIgnoreClass(TFont);
    // TP_GlobalIgnoreClass(TStatusBar);
    // TP_GlobalIgnoreClass(TWebBrowser);
    // TP_GlobalIgnoreClass(TNoteBook);
    // TP_GlobalHandleClass(TCustomTreeView,HandleTreeView);
    // TP_GlobalHandleClass(TKWizardCustomPage,HandleWizardPage);
    AlreadyDone := true;
  end;
  TranslateComponent(self,'default');
end;
```

The AlreadyDone variable is needed because this code will be called for each form in the application and gnugettext raises an exception if the TP_GlobalXXXX functions are called more than once for the same class. Personally, I think this is unnecessary. There is no risks involved adding these call many times as the class or property is to be ignored anyway. But since repeated calls to TP_GlobalHandleClass class *should* generate an exception in my opinion, I would still need the AlreadyDone variable, so this is no big issue with me.

The TranslateComponent call is the one doing all the magic: this function iterates all the components on the form and all it's subcomponents, hunting out published properties that can be translated. If a property is found, it searches the mo file for a translation and uses RTTI to change the property value (if it isn't read-only). Actually, there is another piece of hidden magic working for us as well: remember that I use resourcestrings? Well, these are also handled automatically thanks to a dynamic replacement of the LoadResString function in gnugettext.pas. This replacement function calls into gnugettext instead of into the resource DLL as the ITE does. Together, they cover almost everything that can be translated in an application (unless you are using Unicode in which case resourcestring translation won't work).

When I ran the program, a lot of the strings were translated but not all. Among other things, neither menus, treeviews nor all strings in the KWizard we translated. Additionally, some of the forms I opened generated Access Violations and strange errors. Something seemed to be amiss and how to fix that is our next priority.

### Handling components dxgettext doesn't handle

If you read the documentation for dxgettext (and you should!), you soon realize that it doesn't handle *everything* that you throw at it. It can handle published properties of the components passed into TranslateComponent or published properties of components owned by that component. If you create components dynamically at run-time (with Owner set to nil), these components won't be translated unless you call TranslateComponent explicitly. Additionally, public properties won't be translated (they don't have any RTTI). Some components, like treeviews and listviews, have item list properties that you need to handle manually by adding your own procedure to explicitly iterate over the list and use _() to translate them. For a treeview, here's the code I added to my base form:

```
type
  THackTreeView = class(TCustomTreView);

procedure TfrmGnuGT.HandleTreeView(Obj: TObject);
var N:TTreeNode;T:THackTreeView;
begin
  T := THackTreeView(Obj);
  N := T.Items.GetFirstNode;
```

```
  while N <> nil do
  begin
    N.Text := _(N.Text);
    N := N.GetNext;
  end;
end;
```

I also had to tell dxgettext that I want to handle treeview translations myself, so I added a call to TP_GlobalHandleClass in AfterConstruction:

```
TP_GlobalHandleClass(TCustomTreeView,HandleTreeView);
```

I also added similar handlers for listviews and the KWizard.

## Problems

When running the program, I noticed that some of my menu items weren't translated correctly. I use TActionLists exclusively and those items were correctly translated but the top level menu items (those without actions) weren't translated. I ran gnugettext in debug mode and found that it was caused by the menu component(s) having their AutoHotKeys property set to automatic. I changed this to manual and explicitly assigned hot keys (Alt+F, Alt+E etc) to all top level menu items. Since I didn't want to regenerate the po, I just added these new strings manually to the po file. Now these items were also translated correctly.

I also noticed that the color combobox from JVCL that I was using didn't translate it's text captions (the ColorNameMap property). After some searching, debugging and hair pulling I found a problem in the way the component retrieved the color names, fixed it (thankfully, I am a developer on JVCL so I can do these things!) and then it worked like a charm.

Running the program again everything seemed to work fine until I tried to open the report form: this form uses a TWebBrowser (these are HTML reports and the TWebBrowser is used to preview and print them) and I got a nasty Exception telling me that the StatusText property of the control couldn't be translated. The error message also suggested how I should fix the problem (very nice!). Since there is no visible UI elements in TWebBrowser (except for the main window, which contains nothing to translate) I elected to add a TP_GlobalIgnoreClass for it and the form then opened without problems.

Next, I started to work through all my menus, popups and forms one by one to see if everything was translated and worked as expected. I found that I had some comboboxes that lost their stored ItemIndex when they were translated (it was reset to -1) so I added some code to set these programmatically. Also, I found some items that hadn't been translated in the po and I fixed these with poEdit.

Everything seemed to work fine until I tried to open one of the forms: I got an unexpected AV. I tried to trace the AV but didn't get too far: the code in gnugettext is highly recursive making it problematic to find the spot where an error occurs. Since the form used an "interposer class" - a class declaration that overrides an existing class, I thought at first that this was the cause of the problem. I tried temporarily removing it but that didn't help. After some more debugging I finally figured out that the AV was caused by a TNotebook on the form and after adding it to the ignore list, the error disappeared. Had I read the documentation a little more carefully, I probably wouldn't have had this problem since it mentions notebooks as one component that causes problems.

The next weird error was with one of my toolbars: suddenly the rightmost buttons on the toolbar had switched event handlers! I couldn't believe this at first and had to check several times to make sure that I was seeing things right. It turned out that I had a button on the toolbar that was hidden at run-time and this seemed to cause gnugettext to somehow switch the buttons around, so what I thought was the fifth

button was actually the sixth according to gnugettext. Since I didn't really need the hidden button, I removed it and didn't investigate it further but it might be something to be vary of.

Next issue was the KWizard I was using: the button captions and everything on the pages was translated properly but the Header.Title.Text and Header.SubTitle.Text were not. I suspect this has to do with the fact that these properties use nested (TPersistent) classes and dxgettext doesn't handle that. I added a TP_GlobalHandleClass for TKWizardCustomPage to TfrmGnuGT and got it running fine.

One other oddity in the wizard form was a TStaticText that disappeared when translated. I checked the debug log from gnugettext and the string was found and translated but nothing showed up at run-time. At first I thought it had to do with the anchoring (it was anchored left, bottom) but that wasn't it. I finally figured out that it had to do with AutoSize being set to true and setting it to false fixed the problem. Apparently, the label was resized to zero width when the Caption changed but it was never resized according to the width of the new Caption.

## Conclusion

The whole process of moving an app from ITE to dxgettext can be broken down into the following steps:

- Get and install the latest version of dxgettext (http://dxgettext.sf.net/)
- Get and install latest version of poEdit (http://poedit.sf.net/)
- Add a form (let's call it TfrmGnuGT) to your project. Use this form as the ancestor for all other forms in the application. Override the AfterConstruction method and add calls to TP_GlobalIgnoreClass, TP_GlobalHandleClass,TP_GlobalIgnoreClassProperty and to TranslateComponent as necessary.
- Add gnugettext.pas to the new form's uses clause. Make sure gnugettext is in your path or copy it to your project folder.
- Go through all the forms in the project and change their inheritance so they now inherit from TfrmGnuGT. Add the TfrmGnuGT unit's name to the forms interface uses clause.
- In the Explorer, right-click your project folder and select "Extract translations to template". Your sources are parsed and all found strings are put into a file named deafult.po
- Double-click the default.po file to open with poEdit. Verify that everything looks OK. Close it again.
- Copy dfntopo.exe and the default.po into (one of) your ITE language subfolders. Open a command prompt in that folder and type dfntopo <ENTER> to see the command-line switches for the tool. Run the tool to extract translations from the dfn and rc files in the folder. The resulting default.po should now contain at least some translated strings: open with poEdit to verify.
- Repeat above step for each of your languages
- Create a subfolder structure below your projects output folder for each of your languages using this format: <root>\<langcode>\LC_MESSAGES\ and put each of the previously created default.po files into each folder.
- Manually change "\x9\" in the po file to "\x9" and make sure quote characters are properly escaped.
- Create your own translation handlers for classes like treeviews, listviews and any third-party that refuses to translate.

- Disable handling of some classes (like TFont, TWebBrowser and TNotebook), that obviously shouldn't be translated or that can cause problems with dxgettext.

I had initially planned to add dynamic switching functionality to the program but decided against it. Our users don't really need to switch languages at run-time since they once and for all deicde which language they want to use and stick with that.

All in all, the migration went better than I had thought. There was some problems but nothing unsolvable and it took about 4-5 hours to do it from start to finish. In conclusion, well worth the effort.

## Translation statistics

The `msgfmt` program is able to output some statistics about the contents of a po file:

```
C:\source\sf\DXGETT~1\TRANSL~1\de>msgfmt --statistics kylix3.po
1815 translated messages, 113 fuzzy translations, 53 untranslated mes-
sages.
```

Additionally, the `msgshowheader` tool is able to show the header of a compiled mo file:

```
C:\source\sf\DXGETT~1\TRANSL~1\de>msgshowheader kylix3.mo
Project-Id-Version: Kylix 3 German
POT-Creation-Date: 2003-03-02 18:54
PO-Revision-Date: 2003-07-02 17:20+0100
Last-Translator: Sandro Wendt <info@xan.de>
Language-Team: XAN <info@xan.de>
MIME-Version: 1.0
Content-Type: text/plain; charset=utf-8
Content-Transfer-Encoding: 8bit
```

In order to generate statistics about translations, we just need to be able to create a web page by iterating over all po files and parse the output from the above programs. There is a Python[5] script in the translations CVS module on SourceForge for this purpose.

For further information on this topic, ask in our forum[6].

## Multiple instances

This chapter will contain an explanation of the TGnuGettextInstance class.

## Multithreading issues

The procedures in `gnugettext.pas` are not multithreading safe by themselves. If you want to create a threadsafe application, you will need to create one TGnuGettextInstance object for each thread and make sure that each thread only uses its own object.

# Notes

1. http://www.timemetrics.se/
2. http://poEdit.sourceforge.net/
3. http://dxgettext.sf.net/
4. http://poedit.sf.net/
5. http://www.python.org/

6. http://groups.yahoo.com/group/dxgettext/

# Appendix A. API reference

## procedure AddDomainForResourceString (domain:string);

The initialization section of the gnugettext unit hooks into several Delphi runtime functions and replaces those functions. One of the functions that are replaced, is the function LoadResString, which retrieves resourcestring strings from the resource-part of the .exe file. The replacement first finds the string from the resource-part of the .exe file, and then translates it using the 'default' text domain. You can instruct the system to search for a translation in other text domains, too, if it isn't found in the default text domain, by using this procedure. Simply write this to make LoadStr search `delphi.mo`, too, if it isn't found in `default.mo`:

```
AddDomainForResourcestring ('delphi');
```

## procedure RemoveDomainForResourceString (domain:string);

If a text domain has been added using AddDomainForResourcestring(), you can remove it again using this function.

## function LoadResString(ResStringRec: PResStringRec): widestring;

This function is 100% identical to LoadResString.

## function LoadResStringW(ResStringRec: PResStringRec): widestring;

LoadResStringW is a replacement of the system unit function named LoadResString, which works exactly the same way, except that the string fetched is translated (if possible) and the translation is returned as a widestring.

## function LoadResStringA(ResStringRec: PResStringRec): ansistring;

This function is identical to LoadResStringW, except that the string is returned using ansistring. There should be no need for this function - it only exists because this is exactly the function that is used when the resourcestring keyword is used in Delphi.

## var ExecutableFilename:string;

Don't modify this variable. It contains the full path and filename of the .exe file, if `gnugettext.pas` is compiled into an .exe file, and it contains the full path and filename to the .dll file, if `gnugettext.pas` is compiled into a .dll file. This variable is used to find any embedded translations, if present.

## procedure HookIntoResourceStrings (enabled:boolean=true; SupportPackages:boolean=false);

This procedure lets you control, whether resourcestring retrieval should be translated automatically or not. The default is to have this enabled, but there may be situations, where you want to disable it.

Also, this procedure lets you hook deeper into the runtime library, which is needed when creating packages. Please note, that package support requires you to keep track

of designtime and runtime. You may only call this function with the second parameter set to True during runtime. Setting it to true during design time may make your package conflict with other packages, that also hook into the runtime libraries.

The problem with packages are, that calling a function from the runtime library doesn't call the functions directly - instead it calls into an address, where you will find a machine code jump to the real function, which can be shared between packages. When you load and unload packages inside the Delphi ITE, the packages are sharing runtime libraries. If one package would hook into the runtime library, and another package does the same, and the packages are unloaded FIFO style, the "unhooking" of the runtime library done by last package that is unloaded, will make your system unstable. It is very important to understand this when creating packages, because your system might look as if it just works, but it might not work with your customer unless you do it right.

## const DebugLogFilename='c:\dxgettext-log.txt';

This is the full path of the log output when doing debugging. See the Section called *Debugging* in Chapter 4> for more information.

## TExecutable

This class is not for end-user usage. Please ignore it.

## TGetPluralForm

This class is not for end-user usage. Please ignore it.

## TGnuGettextInstance class

The entire functionality of the `gnugettext.pas` unit is encapsulated in the `TGnuGettextInstance` class. This way you can instantiate multiple instances if you need an instance with different settings than the ones from the default instance. The default instance is put into a variable named `DefaultInstance`, and almost all non-class procedures and functions in the unit refer to this instance.

## procedure UseLanguage(LanguageCode: string);

When an application starts up, the initialization section of `gnugettext.pas` makes this call:

```
UseLanguage('');
```

This call examines the system language settings and sets the language values accordingly. On Windows, GetLocaleInfo() is used to determine the language settings, although on Windows 95, GetThreadLocale() is used because GetLocaleInfo() wasn't implemented in that version.

The OS language settings can always be overridden by setting the `LANG` environment variable like this (example is for Windows):

```
set LANG=de_DE
myapp
```

or on Linux:

```
LANG=de_DE && ./myapp
```

When a language has been set, UseLanguage will examine the first two letters and find out which plural forms that apply to this language. See the Section called *Plural forms* in Chapter 2> for more information on this topic.

All mo files that were open are closed. mo files are opened again when the text domains are accessed.

A language code usually consists of a two-letter lowercase language code, an underscore, and a two-letter uppercase country code. On Windows, this is not case sensitive, but on Linux, it is. German in Germany becomes `de_DE`, English in England becomes `en_GB` and flamish becomes `nl_BE`. If no translations can be found or a 5-character language code, the system automatically attempts to use only the first two digits. This means that if there is no `nl_BE/default.mo` file, `nl/default.mo` will be used instead, if present.

You can find the two-letter ISO 639 language codes in the Section called *ISO 639 language codes* in Appendix F> and the two-letter ISO 3166 country codes in the Section called *ISO 3166 country codes* in Appendix F>.

## function _(msg:widestring):widestring;

The `_()` function is an alias to the gettext function. Please see the Section called *function gettext(msg:widestring):widestring;>* for further information.

## function GetCurrentLanguage:string;

This function returns the language code that was specified to the last call of `UseLanguage()`. If `UseLanguage()` has not been called, yet, or was called with an empty string as parameter, it returns the language code that was derived from the operating system language settings.

See the Section called *procedure UseLanguage(LanguageCode: string);>* for more information.

## function gettext(msg:widestring):widestring;

This function is the most important function of them all, and is the function that names GNU gettext. It takes a string as parameter and returns the translation of that string, if a translation can be found. It is used like this:

```
MessageDlg (gettext('Hello, World'),mtInformation,[mbOK],0);
```

Because the `_()` function is the same as `gettext()`, the line above is normally written like this:

```
MessageDlg (_('Hello, World'),mtInformation,[mbOK],0);
```

The string parameter should only contain ascii characters. If it contains non-ascii characters, the conversion from ansistring to widestring may be sensitive to current Windows locale.

The result value is a widestring. If you use this in context with the Delphi `string` type (`ansistring`), Delphi 6 and later will automatically convert the string types. The MessageDlg example above show this, the first parameter in `MessageDlg()` is a `string`, not a `widestring`. Delphi converts between `string` and `widestring` using Windows API calls, and therefore converts perfectly to and from multibyte character sets. On Linux, the behaviour is identical.

The `gettext()` functions looks up translations in the default.mo file (the default text domain) unless the `textdomain()` procedure has been called.

**Special case: The empty string:** Please note that you should never try to translate an empty string (`_('')`). The translation of the empty string is the message header, which contains information about the translator, the character set, modification date etc.

**Performance:** The `gettext()` function is very fast at looking up translations. It does a binary search on the translation data, which are sorted in binary, and the search is case sensitive and only a perfect match will give a translation. The translation files are memory mapped on Windows and read into memory on Linux, and therefore it doesn't take much time to find a translation. However, you should consider to store a translated value if you need it a lot of times, like when populating a big array.

## function dgettext(Domain: string; MsgId: widestring): widestring;

This function is identical to `gettext()`, except that it looks up translations in the text domain that is specified as the first parameter. You should always specify a string literal as first parameter - don't use variable names etc.:

```
LanguageList.Add(dgettext('languagenames','Danish'));
```

In this example, Danish is looked up in `languagenames.mo`. During extraction of strings from Delphi/ObjectPascal source code, 'Danish' will be put into `languagenames.po`.

**Special note for C/C++ programmers:** The string extraction tool that is used for C and C++ source code extracts differently than the extraction tool for ObjectPascal source code. In the example above, the string `'Danish'` will be put in the main default.po file, mixing everything up.

For further information on this subject, please see the Section called *function gettext(msg:widestring):widestring;*>.

## function ngettext(const singular,plural:widestring;Number:longint):widestring;

This function is explained in the Section called *Plural forms* in Chapter 2>.

## function dngettext(Domain,singular,plural:widestring;Number:longint):widestring;

This function is explained in the Section called *Plural forms* in Chapter 2>.

## function getcurrenttextdomain:string;

This function returns the current text domain. The default value for the current text domain is 'default', but can be changed using `textdomain()`. See the Section called *procedure textdomain(Domain:string);*> for further information.

## procedure textdomain(Domain:string);

This procedure changes the current textdomain, i.e. the text domain that is used by `gettext()`, `_()` and `ngettext()`. The default text domain, if `textdomain()` is not called, is 'default'.

Usually, libraries, modules and other pieces of software that are not developed together with the application that they are included in, should use another text domain. For instance, if the programmer of program A uses module B, then module B should use another text domain that 'default'.

## procedure bindtextdomain(Domain:string; Directory:string);

Each text domain can be located in a separate directory. The default is to search for all mo files as `applicationdir/locale/XX/LC_MESSAGES/domainname.mo`, where applicationdir is the directory where the .exe file or .dll file is, and XX is the language code. You can specify an alternative location for a domain here. Example:

```
bindtextdomain ('moduleB','c:\moduleB\locale\');
```

In this example, all translations done using the text domain 'moduleB' will be looked up in `c:\moduleB\locale\XX\LC_MESSAGES\moduleB.mo`. This can be very useful if you link a module into your program file, but the module translations are somewhere else on your harddisk.

## procedure bindtextdomainToFile (Domain,Filename:string);

If you want to use `gettext()` as a string translation table that is independent of languages, you might want to bind a domain to a specific file on the harddisk. For instance, you might want to translate the ISO language codes to English language names. This could be done like this:

```
function GetLanguageName (isocode:string):string;
var
  EnglishLanguageName:string;
begin
  bindtextdomainToFile ('isocodes','c:\isocodes.mo');
  EnglishLanguageName:=dgettext('isocodes',isocode);
  Result:=dgettext('languagenames',EnglishLanguageName);
end;
```

There are many ways this can be used. For instance, the `msgshowheader` command line tool uses `bindtextdomainToFile()` to fetch the header from a specific file that is given as parameter. It is done something like this:

```
bindtextdomainToFile ('parameterfile',paramstr(1));
writeln (dgettext('parameterfile',''));
```

The translation of the empty string is always the header.

## procedure GetListOfLanguages (domain:string; list:TStrings);

This procedures searches the directory structure and the embedded translations for any valid translation files for the specified domain. It uses the `bindtextdomain()` directory. The result is a list of language codes that it puts into the list parameter.

It is then up to the programmer to convert these language codes into language names. Example:

```
// Put the language codes into a listbox
```

```
ListBox.Items.Clear;
DefaultInstance.GetListOfLanguages ('default',ListBox.Items);

// Convert the language names to an English language name using isotolanguagenames.mo
DefaultInstance.BindtextdomainToFile ('isotolanguagenames',extractfilepath(paramstr(0))
DefaultInstance.TranslateProperties (ListBox,'isotolanguagenames');

// Translate the English language name to a localized language name
DefaultInstance.TranslateProperties (ListBox,'languagenames');
```

## function GetTranslationProperty (Propertyname:string):WideString;

The translation files contain a header as the translation of an empty string. A typical header looks like this:

```
msgid ""
msgstr ""
 "Project-Id-Version: Delphi 7 RTL\n"
 "POT-Creation-Date: 2003-03-02 18:54\n"
 "PO-Revision-Date: 2003-06-01 11:52--100\n"
 "Last-Translator: Lars B. Dybdahl <Lars@dybdahl.dk>\n"
 "Language-Team: Dansk <da@li.org>\n"
 "MIME-Version: 1.0\n"
 "Content-Type: text/plain; charset=UTF-8\n"
 "Content-Transfer-Encoding: 8bit\n"
 "License: You may use this file any way you want\n"
```

As you can see, all lines consist of a keyword, a colon, a space and a value. In order to retrieve these informations easily, you can use the `GetTranslationProperty()` function like this:

```
LabelTranslationLicense.Caption:=DefaultInstance.GetTranslationProperty('License');
```

## function GetTranslatorNameAndEmail:widestring;

These two lines do the same thing:

```
LabelTranslator.Caption:=DefaultInstance.GetTranslationProperty('Last-
Translator');
LabelTranslator.Caption:=DefaultInstance.GetTranslatorNameAndEmail;
```

In other words, this function is just an easy way to fetch the translator name, but not the only one. See the Section called *function GetTranslationProperty (Propertyname:string):WideString;>* for more information.

## procedure SaveUntranslatedMsgids(filename: string);

This function was originally meant to be used for debugging, but much better tools are available now. Please do not use this function - expect it to become deprecated in future versions.

## procedure TranslateProperties(AnObject:TObject; textdomain:string=”);

This procedure iterates through all properties of `AnObject` and all it's subcomponents, and translates all string and widestring properties using the specified text domain. If no text domain is specified, the current text domain is used.

It will skip all properties that were marked to be ignored by one of the TP_* procedures. If no translation is found for a string, it is not translated.

## procedure TranslateComponent(AnObject: TComponent; TextDomain:string='');

This procedure will basically do the same as `TranslateProperties()`, but it will add a subcomponent to `AnObject`, which remembers all untranslated strings. This makes it possible to retranslate `AnObject` to another language later.

The second time that `TranslateComponent()` is called for a specific object, the subcomponent (that remembers all untranslated strings) is detected, and a retranslation is done. The retranslation retranslates exactly those properties that were translated the first time - it does not recurse all subcomponents to redetect string properties. Therefore, you must be careful that all subcomponents that were present at the first translation, also are present at the retranslation.

## function TP_CreateRetranslator:TExecutable;

This function is only for internal use. Don't use it yourself.

## procedure TP_Ignore(AnObject:TObject; const name:string);

This procedure only affects the next execution of `TranslateProperties()` or `TranslateComponent()`. The first parameter must be the same as the first parameter in those other two procedures, and the name parameter specifies, which property you don't want to be translated. Several syntaxes are allowed:

```
 TP_Ignore (self,'ButtonOK.Caption');   // Ignores caption on ButtonOK
 TP_Ignore (self,'MyDBGrid');           // Ignores all properties on com-
ponent MyDBGrid
 TP_Ignore (self,'.Caption');           // Ignores self's caption
```

Since this procedure only has effect on the upcoming translation, it should always be used just before `TranslateProperties()` or `TranslateComponent()`. Example:

```
procedure TFormMain.FormCreate(Sender: TObject);
begin
  TP_Ignore (self, 'Listbox1.Items');
  TranslateComponent (self);
end;
```

In this example, that shows an implementation in the form's FormCreate event handler, all components on the form except the items in Listbox1 are translated.

## procedure TP_GlobalIgnoreClass (IgnClass:TClass);

This procedure puts a class on a global ignore list, so that all objects of this type or of types descending from this type won't be translated by `TranslateProperties()` or `TranslateComponent()`. For instance, it is not very useful to have TFont objects translated, and therefore this line would make a lot of sense in most applications:

```
TP_GlobalIgnoreClass (TFont);
```

A good place to put this line is in the .dpr file. See the Section called *Sample.dpr* in Appendix B> for further information.

## procedure TP_GlobalIgnoreClassProperty (IgnClass:TClass;propertyname:string);

This procedure puts one property of a class on a global ignore list, so that this property won't be translated by `TranslateProperties()` or `TranslateComponent()` for all objects of this type or of a type that descends from this type. For instance, it is not very useful to have TField.Fieldname translated, and therefore this line would make a lot of sense in most database applications:

```
TP_GlobalIgnoreClassProperty (TField,'Fieldname');
```

A good place to put this line is in the .dpr file. See the Section called *Sample.dpr* in Appendix B> for further information.

## procedure TP_GlobalHandleClass (HClass:TClass;Handler:TTranslator);

This procedure makes it possible implement another translation handling of a class than what you can control with `TranslateProperties()`, `TranslateComponent()` and the TP_* procedures. For instance, if you create a new class that does not use the `published` keyword, its properties cannot be iterated by `TranslateProperties()` and `TranslateComponent()`. Another example is, when you get 3rd party components, that fail when you translate it's properties, but if you do something special to them, they translate well.

Basically, if you cannot translate a component using `TranslateComponent`, but you can write a procedure yourself that can, then use `TP_GlobalHandleClass` to make `TranslateComponent` use your procedure.

# Appendix B. "Hello, World" source code

## Sample.dpr

```
program Sample;

uses
  gnugettext in 'gnugettext.pas',
  gginitializer in 'gginitializer.pas',
  Forms,
  Graphics,
  SampleForm in 'SampleForm.pas' {FormMain};

{$R *.res}

begin
  // This is the list of ignores for this project. The list of
  // ignores has to come before the first call to TranslateComponent().
  TP_GlobalIgnoreClass(TFont);

  Application.Initialize;
  Application.CreateForm(TFormMain, FormMain);
  Application.Run;
end.
```

## gginitializer.pas

```
unit gginitializer;

interface

implementation

uses
  gnugettext;

initialization
  // Use delphi.mo for runtime library translations, if it is there
  // by putting this line into this separate unit, we can execute it
  // before the initialization sections of the other units are executed.
  AddDomainForResourceString('delphi');

end.
```

## SampleForm.pas

```
unit SampleForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TFormMain = class(TForm)
    ButtonTestGettext: TButton;
    ButtonTestResourcestring: TButton;
    procedure ButtonTestGettextClick(Sender: TObject);
    procedure ButtonTestResourcestringClick(Sender: TObject);
```

```
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormMain: TFormMain;

implementation

uses
  gnugettext;

{$R *.dfm}

procedure TFormMain.FormCreate(Sender: TObject);
begin
  TranslateComponent (self);
end;

resourcestring
  MessageToUser='Thank you for clicking this button';

procedure TFormMain.ButtonTestResourcestringClick(Sender: TObject);
begin
  // This is a demonstration of automatic resourcestring translation
  MessageDlg (MessageToUser,mtInformation,[mbOK],0);
end;

procedure TFormMain.ButtonTestGettextClick(Sender: TObject);
begin
  // This is a demo of the _() syntax
  MessageDlg (_('Thank you for clicking this button'),mtInformation,[mbOK],0);
end;

end.
```

## SampleForm.dfm

```
object FormMain: TFormMain
  Width = 354
  Height = 179
  Caption = 'GNU gettext sample application'
  OnCreate = FormCreate
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  object ButtonTestGettext: TButton
    Left = 64
    Top = 48
    Width = 75
    Height = 25
    Caption = 'Click me'
    OnClick = ButtonTestGettextClick
  end
  object ButtonTestResourcestring: TButton
    Left = 144
    Top = 48
    Width = 75
    Height = 25
```

```
    Caption = 'Click me'
    OnClick = ButtonTestResourcestringClick
  end
end
```

# Appendix C. Dxgettext command-line tools reference

## assemble

This tool embeds your translations in your executable. When your programs work, including translations, type this:

```
assemble --dxgettext applicationfilename.exe
```

This will find all translation files (mo files) in the locale subdirectory and append them to the exe file. It also works with dll files. This system should coexist nicely with other tools that append something to the .exe file, as long as the other tool knows how to coexist with other tools. If you are using another tool to append something, that does not coexist with other tools, use that tool before you embed translations.

## dfntopo

This tool converts your Delphi ITE translations to PO files. This is the help screen that appears when you run the program without parameters:

```
DFNToPO: extracts strings from DFN and RC files and insert any trans-
lations into a PO file

This program is subject to the Mozilla Public License
Version 1.1 (the "License"); you may not use this program except
in compliance with the License. You may obtain a copy of the License at
http://www.mozilla.org/MPL/MPL-1.1.html

Portions created by Peter Thornqvist are
Copyright (c) 2003 by Peter Thornqvist; all rights reserved.

Usage:
DFNToPO [options]
where [options] can be any of the following:
-s :  searches in sub-folders also (defaults to FALSE)
-f :  force creation of DFN/RC items not found in po (defaults to FALSE)
-m :  merge TStrings items into single item delimited by \n (defaults to FALSE)
-p<POFile>  : full path and filename of the po file (defaults to "de-
fault.po" in current dir).
            NOTE: Filenames with spaces must be enclosed in quotes.
-d<DFNPath> : full path to the dfn files. Do NOT include a filemask (de-
faults to current dir).
            NOTE: Paths with spaces must be enclosed in quotes.
```

See the Section called *Migrating from the ITE to dxgettext* in Chapter 5> for further information.

## dxgettext

This will extract all texts from the specified files and save them in a file named default.po:

```
dxgettext usage:
dxgettext *.pas *.dfm *.dpr -r
dxgettext -b c:\source\myprogram --delphi
```

The following file formats are supported:

- ObjectPascal source code: *.pas, *.inc and *.dpr.

- DFM/XFM files: *.dfm and *.xfm.
- C/C++ source code: *.c and *.cpp. These are extracted using the xgettext command line tool (see the Section called *xgettext* in Appendix D>).
- RC files: *.rc.
- Executables: *.exe, *.dll and *.bpl. (only on Windows)

This is the help text that appears when you run the program without parameters:

```
dxgettext 1.1.1
dxgettext usage:
  dxgettext *.pas *.dfm *.dpr -r
  dxgettext -b c:\source\myprogram --delphi

This will extract all texts from the specified files and save them
in a file named default.po.

Options:
  --delphi              Adds the wildcards: *.pas *.inc *.rc *.dpr *.xfm *.dfm
  --kylix               Adds the wildcards: *.pas *.inc *.rc *.dpr *.xfm
  -r                    Recurse subdirectories
  -b dir                Use directory as base directory for filenames.
                        You can specify several base dirs, that will all be scanned.
  -o:msgid              Order by msgid
  -o:occ                Order by occurence (default)
  -o dir                Output directory for .po files
  -q                    Quiet: Reduces output to absolute minimum.
  --codepage nnn        Assume the specified codepage. Default is CP_ACP.
  --nowc                Assume wildcards to be part of filenames
  --ignore-constreplace Suppresses warnings about CRLF

If a filename is preceded with @, it is assumed to contain a list of
filenames or file masks.
```

## dxgreg

This Windows only tool can do three different things:

- Starting it with no parameters will make it register the shell extensions with your Windows. This is normally done during installation and requires administrative rights with your computer. If you lose the desktop integrations for some reason, just run this program and your Windows Explorer will be fully integrated with GNU gettext for Delphi, C++ Builder and Kylix again.
- If you start it with the `--reset-user` parameter, it will delete all user-specific file extension associations done to the file types used by this system. This only makes sense on Windows 2000, XP and later. It does not require administrative rights.
- If you start it with the `--uninstall` parameter, it will delete all shell integration features that make use of executable files in the directory where `dxgreg.exe` was started. This is usually done when uninstalling this system.

## ixtopo

This tool is useful for converting Delphi ITE translations to GNU gettext. This is the text that appears when you run the program without parameters:

```
IXTOPO 1.0: extracts strings from an ITE xml file and inserts any trans-
lations into a PO file.

This program is subject to the Mozilla Public License
```

```
Version 1.1 (the "License"); you may not use this program except
in compliance with the License. You may obtain a copy of the License at
http://www.mozilla.org/MPL/MPL-1.1.html

Portions created by Peter Thornqvist are
Copyright (c) 2003 by Peter Thornqvist; all rights reserved.

Usage:

ixtopo <xmlfile> <pofile> <locale> [-f]

where:

<xmlfile>    is the xml file to read from (REQUIRED)
<pofile>     is the po file to write to (REQUIRED)
<locale>     is the locale to extract and insert into the po file (REQUIRED)
-f           forces the creation of new entries in the po file if not found (OP-
TIONAL, defaults to FALSE)

NOTE:
Since locale names in XML files contains spaces, you must
put quotes around the locale, i.e use "US en" to extract
the American english translations
```

See the Section called *Migrating from the ITE to dxgettext* in Chapter 5> for further information.

## msgimport

This tool makes it possible to create a po file from an ascii tabulated text file. It does not support multiline msgstr values. In order to use it, it must conform to these rules:

- Be an utf-8 encoded text file
- It may not contain any header lines etc.
- It must contain exactly two columns, separated by a tabulator (ascii 9), where the first column contains the msgid, and the second column contains the msgstr.

If you have something in tabular form that you want to convert to a po file, do it like this:

- Load the tabulated data into a spreadsheet. If you don't have a spreadsheet on your computer, get it at openoffice.org[1].
- Delete all columns except the two that contain msgid and msgstr. Make sure the first column is msgstr.
- Delete all rows that do not contains data to be converted.
- Save the file as a text-file, tabulator separated with no text delimiters.
- Load the text file into an utf-8 capable text editor like Unired[2].
- Save the text file as an utf-8 text file with "no BOM".
- Run

  `msgimport textfile.txt -o output.po`

  and you got a valid .po file.

## msgmergePOT

This tool can merge two templates to become a translation file. It looks at the comments in each template to find out, which msgids belong together.

This is the help screen that appears when you run the tool with no parameters:

```
msgmergePOT 1.1.1

msgmergePOT usage:
  msgmergePOT english.po otherlanguage.po destination.po

This will create a po file from two identical po extracts that
only differ by the language, for instance the German and English
runtime source code from Borland.
```

## msgmkignore

This tool tries to guess, which msgids should not be translated, and writes those to a separate file. The result of this can be used with msgremove to remove unnecessary msgids from a template before sending it to the translator.

msgmkignore uses several methods to find out, which messages aren't translatable. These include finding messages without a single letter in it and finding messages with letters and digits but no spaces.

This is the help screen that appears when you run the program with no parameters:

```
msgmkignore 1.1.1

Usage:
  msgmkignore default.po -o ignore.po

This will extract texts from default.po that this program
thinks should not be translated by a translator
```

## msgremove

msgremove can remove all messages from a PO file that are contained in a second PO file. With this, you can remove a lot of unnecessary entries from your automatically generated template before sending it to the translator. This is the text that appears when you run the program without any parameters:

```
msgremove 1.1.1

msgremove usage:
  msgremove default.po -i ignore.po -o output.po

This will generate the file output.po as a copy of default.po,
but without all the MsgIds that are listed in ignore.po.
```

## msgshowheader

This tool takes an MO file as parameter and simply outputs the header to standard out. This is the text that appears when you run the program without any parameters:

```
msgshowheader 1.1.1

msgshowheader usage:
  msgshowheader translation.mo
```

```
This will output the header of the translation to stdout, i.e. _("").
```

### msgsplitTStrings

This tool is only intended to be used when upgrading from versions of dxgettext that are older than version 1.0. It takes all multiline messages and adds each line without the linebreak as a new message.

### msgstripCR

This tool is only intended to be used when upgrading from versions of dxgettext that are older than version 1.0. It simply removes all '\r' escape sequences from messages.

## Notes

1. http://www.openoffice.org/
2. http://unired.sf.net/

# Appendix D. GNU Command-line tools reference

## msgattrib

Changes the fuzzy/obsolete/translated status of messages in a PO file.

```
NAME
       msgattrib - attribute matching and manipulation on message catalog

SYNOPSIS
       msgattrib [OPTION] [INPUTFILE]

DESCRIPTION
       Filters the  messages  of  a  translation  catalog  according to their
       attributes, and manipulates the attributes.

       Mandatory arguments to long options are mandatory  for short  options
       too.

   Input file location:
       INPUTFILE
       input PO file

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

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

   Output file location:
       -o, --output-file=FILE
       write output to specified file

       The  results are written to standard output if no output file is speci-

       fied or if it is -.

   Message selection:
       --translated
       keep translated, remove untranslated messages

       --untranslated
       keep untranslated, remove translated messages

       --no-fuzzy
       remove 'fuzzy' marked messages

       --only-fuzzy
       keep 'fuzzy' marked messages

       --no-obsolete
       remove obsolete #~ messages

       --only-obsolete
       keep obsolete #~ messages

   Attribute manipulation:
       --set-fuzzy
       set all messages 'fuzzy'

       --clear-fuzzy
       set all messages non-'fuzzy'

       --set-obsolete
       set all messages obsolete
```

```
       --clear-obsolete
       set all messages non-obsolete

       --fuzzy
       synonym for --only-fuzzy --clear-fuzzy

       --obsolete
       synonym for --only-obsolete --clear-obsolete

   Output details:
       -e, --no-escape
       do not use C escapes in output (default)

       -E, --escape
       use C escapes in output, no extended chars

       --force-po
       write PO file even if empty

       -i, --indent
       write the .po file using indented style

       --no-location
       do not write '#: filename:line' lines

       -n, --add-location
       generate '#: filename:line' lines (default)

       --strict
       write out strict Uniforum conforming .po file

       -w, --width=NUMBER
       set output page width

       --no-wrap
       do not break long message lines, longer  than  the  output  page
       width, into several lines

       -s, --sort-output
       generate sorted output

       -F, --sort-by-file
       sort output by file location

   Informative output:
       -h, --help
       display this help and exit

       -V, --version
       output version information and exit

AUTHOR
       Written by Bruno Haible.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>

COPYRIGHT
       Copyright (C) 2001-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO warranty; not even for MERCHANTABILITY or FITNESS FOR  A  PARTICULAR
       PURPOSE.

SEE ALSO
       The full documentation for msgattrib is maintained as a Texinfo manual.
       If the info and msgattrib programs are properly installed at your site,
```

```
       the command

       info msgattrib

       should give you access to the complete manual.
```

## msgcat

This tool merges several PO files. It can also sort the files and change the output format.

```
NAME
       msgcat - combines several message catalogs

SYNOPSIS
       msgcat [OPTION] [INPUTFILE]...

DESCRIPTION
       Concatenates  and  merges  the specified PO files.  Find messages which
       are common to two or more of the specified  PO  files. By  us-
ing  the
       --more-than  option,  greater  commonality may be requested be-
fore mes-
       sages are printed.  Conversely, the --less-than option may be  used  to
       specify  less commonality  before  messages  are printed  (i.e.
       --less-than=2 will only print the unique messages).  Translations, com-

       ments  and  extract  comments will  be  cumulated,  except  that  if
       --use-first is specified, they will be taken from the first PO file  to
       define them.  File positions from all PO files will be cumulated.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

   Input file location:
       INPUTFILE ...
       input files

       -f, --files-from=FILE
       get list of input files from FILE

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

       If input file is -, standard input is read.

   Output file location:
       -o, --output-file=FILE
       write output to specified file

       The results are written to standard output if no output file is speci-

       fied or if it is -.

   Message selection:
       -<, --less-than=NUMBER
       print messages with less than this many definitions, defaults to
       infinite if not set

       ->, --more-than=NUMBER
       print messages with more than this many definitions, defaults to
       0 if not set

       -u, --unique
```

```
            shorthand  for --less-than=2, requests that only unique messages
            be printed

        Output details:
            -t, --to-code=NAME
            encoding for output

            --use-first
            use first available translation for each message,  don't  merge
            several translations

            -e, --no-escape
            do not use C escapes in output (default)

            -E, --escape
            use C escapes in output, no extended chars

            --force-po
            write PO file even if empty

            -i, --indent
            write the .po file using indented style

            --no-location
            do not write '#: filename:line' lines

            -n, --add-location
            generate '#: filename:line' lines (default)

            --strict
            write out strict Uniforum conforming .po file

            -w, --width=NUMBER
            set output page width

            --no-wrap
            do  not  break  long  message lines, longer than the output page
            width, into several lines

            -s, --sort-output
            generate sorted output

            -F, --sort-by-file
            sort output by file location

        Informative output:
            -h, --help
            display this help and exit

            -V, --version
            output version information and exit

    AUTHOR
           Written by Bruno Haible.

    REPORTING BUGS
           Report bugs to <bug-gnu-gettext@gnu.org>.

    COPYRIGHT
           Copyright (C) 2001-2002 Free Software Foundation, Inc.
           This is free software; see the source for copying conditions.  There is
           NO  warranty;  not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
           PURPOSE.

    SEE ALSO
           The full documentation for msgcat is maintained as  a  Texinfo  manual.
```

```
       If  the info  and msgcat programs are properly installed at your site,
       the command

       info msgcat

       should give you access to the complete manual.
```

## msgcmp

This compares two PO files, and is often used to check that everything has been translated.

```
NAME
       msgcmp - compare message catalog and template

SYNOPSIS
       msgcmp [OPTION] def.po ref.pot

DESCRIPTION
       Compare two  Uniforum  style  .po files to check that both con-
tain the
       same set of msgid strings.  The def.po file is an existing PO file with
       the  translations.   The ref.pot file is the last created PO file, or a
       PO Template file (generally created by xgettext).  This is  use-
ful  for
       checking  that  you have translated each and every message in your pro-

       gram.  Where an exact match cannot be found, fuzzy matching is used  to
       produce better diagnostics.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

   Input file location:
       def.po translations

       ref.pot
       references to the sources

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

   Operation modifiers:
       -m, --multi-domain
       apply ref.pot to each of the domains in def.po

   Informative output:
       -h, --help
       display this help and exit

       -V, --version
       output version information and exit

AUTHOR
       Written by Peter Miller.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.

COPYRIGHT
       Copyright (C) 1995-1998, 2000-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO  warranty;  not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
       PURPOSE.
```

```
SEE ALSO
       The full documentation for msgcmp is maintained as  a  Texinfo  manual.
       If  the info  and msgcmp programs are properly installed at your site,
       the command

       info msgcmp

       should give you access to the complete manual.
```

## msgcomm

This tool lists all messages that are common for two or more PO files.

```
NAME
       msgcomm - match two message catalogs

SYNOPSIS
       msgcomm [OPTION] [INPUTFILE]...

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

       Mandatory arguments to long options are mandatory  for short  options
       too.

   Input file location:
       INPUTFILE ...
       input files

       -f, --files-from=FILE
       get list of input files from FILE

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

       If input file is -, standard input is read.

   Output file location:
       -o, --output-file=FILE
       write output to specified file

       The  results are written to standard output if no output file is speci-

       fied or if it is -.

   Message selection:
       -<, --less-than=NUMBER
       print messages with less than this many definitions, defaults to
       infinite if not set

       ->, --more-than=NUMBER
       print messages with more than this many definitions, defaults to
       1 if not set
```

```
       -u, --unique
       shorthand for --less-than=2, requests that only unique  messages
       be printed

   Output details:
       -e, --no-escape
       do not use C escapes in output (default)

       -E, --escape
       use C escapes in output, no extended chars

       --force-po
       write PO file even if empty

       -i, --indent
       write the .po file using indented style

       --no-location
       do not write '#: filename:line' lines

       -n, --add-location
       generate '#: filename:line' lines (default)

       --strict
       write out strict Uniforum conforming .po file

       -w, --width=NUMBER
       set output page width

       --no-wrap
       do  not  break  long  message lines, longer than the output page
       width, into several lines

       -s, --sort-output
       generate sorted output

       -F, --sort-by-file
       sort output by file location

       --omit-header
       don't write header with 'msgid ""' entry

   Informative output:
       -h, --help
       display this help and exit

       -V, --version
       output version information and exit

AUTHOR
       Written by Peter Miller.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.

COPYRIGHT
       Copyright (C) 1995-1998, 2000-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO  warranty;  not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
       PURPOSE.

SEE ALSO
       The full documentation for msgcomm is maintained as a  Texinfo  manual.
       If  the info and msgcomm programs are properly installed at your site,
       the command
```

```
       info msgcomm

       should give you access to the complete manual.
```

## msgen

Very simple program that creates an English translation file, where all translations are a copy of the original, but marked as fuzzy. A translator can then run through all messages and correct any typing errors or improve any words, if needed.

```
NAME
       msgen - create English message catalog

SYNOPSIS
       msgen [OPTION] INPUTFILE

DESCRIPTION
       Creates an  English  translation  catalog.  The input file is the last
       created English PO file, or a PO Template file  (generally  cre-
ated  by
       xgettext).   Untranslated  entries  are assigned a translation that is
       identical to the msgid, and are marked fuzzy.

       Mandatory arguments to long options are mandatory  for short  options
       too.

   Input file location:
       INPUTFILE
       input PO or POT file

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

       If input file is -, standard input is read.

   Output file location:
       -o, --output-file=FILE
       write output to specified file

       The  results are written to standard output if no output file is speci-

       fied or if it is -.

   Output details:
       -e, --no-escape
       do not use C escapes in output (default)

       -E, --escape
       use C escapes in output, no extended chars

       --force-po
       write PO file even if empty

       -i, --indent
       indented output style

       --no-location
       suppress '#: filename:line' lines

       --add-location
       preserve '#: filename:line' lines (default)

       --strict
```

```
       strict Uniforum output style

       -w, --width=NUMBER
       set output page width

       --no-wrap
       do not break long message lines, longer  than  the  output  page
       width, into several lines

       -s, --sort-output
       generate sorted output

       -F, --sort-by-file
       sort output by file location

   Informative output:
       -h, --help
       display this help and exit

       -V, --version
       output version information and exit

AUTHOR
       Written by Bruno Haible.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.

COPYRIGHT
       Copyright (C) 2001-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO warranty; not even for MERCHANTABILITY or FITNESS FOR  A  PARTICULAR
       PURPOSE.

SEE ALSO
       The full documentation for msgen is maintained as a Texinfo man-
ual.  If
       the info and msgen programs are properly installed at  your  site,  the
       command

       info msgen

       should give you access to the complete manual.
```

## msgexec

Runs an external command line tool once for each message in a PO file.

```
NAME
       msgexec - process translations of message catalog

SYNOPSIS
       msgexec [OPTION] COMMAND [COMMAND-OPTION]

DESCRIPTION
       Applies a  command  to all translations of a translation cata-
log.  The
       COMMAND can be any program  that  reads a  translation from  standard
       input. It  is invoked once for each translation.  Its output becomes
       msgexec's output.  msgexec's return code is  the  maximum  re-
turn  code
       across all invocations.

       A  special builtin command called '0' outputs the translation, followed
```

```
       by a null byte. The output of "msgexec 0" is  suitable as  in-
put  for
       "xargs -0".

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

   Input file location:
       -i, --input=INPUTFILE
       input PO file

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

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

   Informative output:
       -h, --help
       display this help and exit

       -V, --version
       output version information and exit

AUTHOR
       Written by Bruno Haible.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.

COPYRIGHT
       Copyright (C) 2001-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO  warranty;  not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
       PURPOSE.

SEE ALSO
       The full documentation for msgexec is maintained as a  Texinfo  manual.
       If  the info and msgexec programs are properly installed at your site,
       the command

       info msgexec

       should give you access to the complete manual.
```

## msgfilter

Runs an external command line tool once for each message, but uses it to make changes to each translation. The external tool takes the translation as input on stdin, and writes the new translation out on stdout.

```
NAME
       msgfilter - edit translations of message catalog

SYNOPSIS
       msgfilter [OPTION] FILTER [FILTER-OPTION]

DESCRIPTION
       Applies a filter to all translations of a translation catalog.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

   Input file location:
       -i, --input=INPUTFILE
```

```
    input PO file

    -D, --directory=DIRECTORY
    add DIRECTORY to list for input files search

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

Output file location:
    -o, --output-file=FILE
    write output to specified file

    The results are written to standard output if no output file is speci-

    fied or if it is -.

    The  FILTER  can  be any program that reads a translation from standard
    input and writes a modified translation to standard output.

Useful FILTER-OPTIONs when the FILTER is 'sed':
    -e, --expression=SCRIPT
    add SCRIPT to the commands to be executed

    -f, --file=SCRIPTFILE
    add the contents of SCRIPTFILE to the commands to be executed

    -n, --quiet, --silent
    suppress automatic printing of pattern space

Output details:
    --no-escape
    do not use C escapes in output (default)

    -E, --escape
    use C escapes in output, no extended chars

    --force-po
    write PO file even if empty

    --indent
    indented output style

    --keep-header
    keep header entry unmodified, don't filter it

    --no-location
    suppress '#: filename:line' lines

    --add-location
    preserve '#: filename:line' lines (default)

    --strict
    strict Uniforum output style

    -w, --width=NUMBER
    set output page width

    --no-wrap
    do not break long message lines, longer  than  the  output  page
    width, into several lines

    -s, --sort-output
    generate sorted output

    -F, --sort-by-file
    sort output by file location
```

```
   Informative output:
       -h, --help
       display this help and exit

       -V, --version
       output version information and exit

AUTHOR
       Written by Bruno Haible.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.

COPYRIGHT
       Copyright (C) 2001-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO warranty; not even for MERCHANTABILITY or FITNESS FOR  A  PARTICULAR
       PURPOSE.

SEE ALSO
       The full documentation for msgfilter is maintained as a Texinfo manual.
       If the info and msgfilter programs are properly installed at your site,
       the command

       info msgfilter

       should give you access to the complete manual.
```

## msgfmt

Compiles a PO file to an MO file.

```
NAME
       msgfmt - compile message catalog to binary format

SYNOPSIS
       msgfmt [OPTION] filename.po ...

DESCRIPTION
       Generate binary message catalog from textual translation description.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

   Input file location:
       filename.po ...
       input files

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

       If input file is -, standard input is read.

   Operation mode:
       -j, --java
       Java mode: generate a Java ResourceBundle class

       --java2
       like --java, and assume Java2 (JDK 1.2 or higher)

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

   Output file location:
       -o, --output-file=FILE
```

```
    write output to specified file

    --strict
    enable strict Uniforum mode

    If output file is -, output is written to standard output.

Output file location in Java mode:
    -r, --resource=RESOURCE
    resource name

    -l, --locale=LOCALE
    locale name, either language or language_COUNTRY

    -d DIRECTORY
    base directory of classes directory hierarchy

    The class name is determined  by  appending  the  locale  name  to  the
    resource  name, separated with an underscore.  The -d option is manda-

    tory.  The class is written under the specified directory.

Output file location in Tcl mode:
    -l, --locale=LOCALE
    locale name, either language or language_COUNTRY

    -d DIRECTORY
    base directory of .msg message catalogs

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

Input file interpretation:
    -c, --check
    perform  all the    checks implied   by   --check-format,
    --check-header, --check-domain

    --check-format
    check language dependent format strings

    --check-header
    verify presence and contents of the header entry

    --check-domain
    check for conflicts between domain  directives  and  the --out-

    put-file option

    -C, --check-compatibility
    check that GNU msgfmt behaves like X/Open msgfmt

    --check-accelerators[=CHAR]
    check presence of keyboard accelerators for menu items

    -f, --use-fuzzy
    use fuzzy entries in output

Output details:
    -a, --alignment=NUMBER
    align strings to NUMBER bytes (default: 1)

    --no-hash
    binary file will not include the hash table

Informative output:
    -h, --help
```

```
       display this help and exit

       -V, --version
       output version information and exit

       --statistics
       print statistics about translations

       -v, --verbose
       increase verbosity level

AUTHOR
       Written by Ulrich Drepper.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.

COPYRIGHT
       Copyright (C) 1995-1998, 2000-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO warranty; not even for MERCHANTABILITY or FITNESS FOR  A  PARTICULAR
       PURPOSE.

SEE ALSO
       The  full  documentation  for msgfmt is maintained as a Texinfo manual.
       If the info and msgfmt programs are properly installed  at  your  site,
       the command

       info msgfmt

       should give you access to the complete manual.
```

## msggrep

Extracts all messages of a translation catalog that match a given pattern or belong to some given source files.

```
NAME
       msggrep - pattern matching on message catalog

SYNOPSIS
       msggrep [OPTION] [INPUTFILE]

DESCRIPTION
       Extracts  all messages of a translation catalog that match a given pat-

       tern or belong to some given source files.

       Mandatory arguments to long options are mandatory  for short  options
       too.

   Input file location:
       INPUTFILE
       input PO file

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

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

   Output file location:
       -o, --output-file=FILE
       write output to specified file
```

```
       The  results are written to standard output if no output file is speci-

       fied or if it is -.

   Message selection:
       [-N SOURCEFILE]... [-M DOMAINNAME]...   [-K  MSGID-PATTERN]  [-
T
       MSGSTR-PATTERN] [-C COMMENT-PATTERN]

       A  message  is  selected  if  it comes from one of the specified source
       files, or if it comes from one of the specified domains, or  if -
K  is
       given  and its key (msgid or msgid_plural) matches MSGID-PATTERN, or if
       -T is given and its translation (msgstr) matches MSGSTR-PATTERN, or  if
       -C is given and the translator's comment matches COMMENT-PATTERN.

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

       rion.

   MSGID-PATTERN or MSGSTR-PATTERN syntax:
       [-E | -F] [-e PATTERN | -f FILE]...

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

       -N, --location=SOURCEFILE
       select messages extracted from SOURCEFILE

       -M, --domain=DOMAINNAME
       select messages belonging to domain DOMAINNAME

       -K, --msgid
       start of patterns for the msgid

       -T, --msgstr
       start of patterns for the msgstr

       -E, --extended-regexp
       PATTERN is an extended regular expression

       -F, --fixed-strings
       PATTERN is a set of newline-separated strings

       -e, --regexp=PATTERN
       use PATTERN as a regular expression

       -f, --file=FILE
       obtain PATTERN from FILE

       -i, --ignore-case
       ignore case distinctions

   Output details:
       --no-escape
       do not use C escapes in output (default)

       --escape
       use C escapes in output, no extended chars

       --force-po
       write PO file even if empty

       --indent
       indented output style
```

```
       --no-location
       suppress '#: filename:line' lines

       --add-location
       preserve '#: filename:line' lines (default)

       --strict
       strict Uniforum output style

       -w, --width=NUMBER
       set output page width

       --no-wrap
       do not break long message lines, longer  than  the  output  page
       width, into several lines

       --sort-output
       generate sorted output

       --sort-by-file
       sort output by file location

   Informative output:
       -h, --help
       display this help and exit

       -V, --version
       output version information and exit

AUTHOR
       Written by Bruno Haible.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.

COPYRIGHT
       Copyright (C) 2001-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO warranty; not even for MERCHANTABILITY or FITNESS FOR  A  PARTICULAR
       PURPOSE.

SEE ALSO
       The  full  documentation for msggrep is maintained as a Texinfo manual.
       If the info and msggrep programs are properly installed at  your  site,
       the command

       info msggrep

       should give you access to the complete manual.
```

## msghack

This very special tool can swap msgid and msgstr in a PO file, remove all translations in order to convert a translation file into a template file, and can also append the contents of one PO file to another.

It is part of Red Hat Linux 9, but hasn't been seen compiled for Windows yet. It also doesn't have a man-page on Red Hat Linux, but here you can see the help when running `msghack --help`:

```
Usage:  /usr/bin/msghack  [OPTION] file.po [ref.po]
This program can be used to alter .po files in ways no sane mind would think about.
      -o               result will be written to FILE
```

```
      --invert          invert a po file by switching msgid and msgstr
      --master          join any number of files in a master-formatted catalog
      --empty           empty the contents of the .po file, creating a .pot
      --append          append entries from ref.po that don't exist in file.po

Note: It is just a replacement of msghack for backward support.
```

## msginit

```
NAME
       msginit - initialize a message catalog

SYNOPSIS
       msginit [OPTION]

DESCRIPTION
       Creates a  new PO file, initializing the meta information with values
       from the user's environment.

       Mandatory arguments to long options are mandatory  for short  options
       too.

   Input file location:
       -i, --input=INPUTFILE
       input POT file

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

   Output file location:
       -o, --output-file=FILE
       write output to specified PO file

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

   Output details:
       -l, --locale=LL_CC
       set target locale

       --no-translator
       assume the PO file is automatically generated

       -w, --width=NUMBER
       set output page width

       --no-wrap
       do not break long message lines, longer  than  the  output  page
       width, into several lines

   Informative output:
       -h, --help
       display this help and exit

       -V, --version
       output version information and exit

AUTHOR
       Written by Bruno Haible.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.
```

```
COPYRIGHT
       Copyright (C) 2001-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO warranty; not even for MERCHANTABILITY or FITNESS FOR  A  PARTICULAR
       PURPOSE.

SEE ALSO
       The  full  documentation for msginit is maintained as a Texinfo manual.
       If the info and msginit programs are properly installed at  your  site,
       the command

       info msginit

       should give you access to the complete manual.
```

## msgmerge

This tool can merge two PO files, where it takes the msgids from one file and the translations from another file. It is normally used to update a translation that was made for one version of a program, to be useful for the next version of the program, by merging the translation with a template that was extracted from the new source code.

```
NAME
       msgmerge - merge message catalog and template

SYNOPSIS
       msgmerge [OPTION] def.po ref.pot

DESCRIPTION
       Merges  two  Uniforum  style .po files together.  The def.po file is an
       existing PO file with translations which will  be  taken  over  to  the
       newly  created  file as long as they still match; comments will be pre-

       served, but extracted comments and file positions  will be  discarded.
       The  ref.pot  file  is  the last created PO file with up-to-date source
       references but old translations, or a PO Template file (gener-
ally  cre-
       ated  by  xgettext);  any  translations or comments in the file will be
       discarded, however dot comments and file positions will be  preserved.
       Where an exact match cannot be found, fuzzy matching is used to produce
       better results.

       Mandatory arguments to long options are mandatory  for short  options
       too.

   Input file location:
       def.po translations referring to old sources

       ref.pot
       references to new sources

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

       -C, --compendium=FILE
       additional  library  of  message translations, may be specified
       more than once

   Operation mode:
       -U, --update
       update def.po, do nothing if def.po already up to date

   Output file location:
```

```
       -o, --output-file=FILE
       write output to specified file

       The results are written to standard output if no output file is speci-

       fied or if it is -.

       Output  file  location  in  update  mode: The result is written back to
       def.po.

       --backup=CONTROL
       make a backup of def.po

       --suffix=SUFFIX
       override the usual backup suffix

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

       none, off
       never make backups (even if --backup is given)

       numbered, t
       make numbered backups

       existing, nil
       numbered if numbered backups exist, simple otherwise

       simple, never
       always make simple backups

       The backup suffix  is  ‘~’,  unless  set  with  --suffix  or  the  SIM-

       PLE_BACKUP_SUFFIX environment variable.

   Operation modifiers:
       -m, --multi-domain
       apply ref.pot to each of the domains in def.po

   Output details:
       -e, --no-escape
       do not use C escapes in output (default)

       -E, --escape
       use C escapes in output, no extended chars

       --force-po
       write PO file even if empty

       -i, --indent
       indented output style

       --no-location
       suppress ’#: filename:line’ lines

       --add-location
       preserve ’#: filename:line’ lines (default)

       --strict
       strict Uniforum output style

       -w, --width=NUMBER
       set output page width

       --no-wrap
```

```
       do  not  break  long  message lines, longer than the output page
       width, into several lines

       -s, --sort-output
       generate sorted output

       -F, --sort-by-file
       sort output by file location

   Informative output:
       -h, --help
       display this help and exit

       -V, --version
       output version information and exit

       -v, --verbose
       increase verbosity level

       -q, --quiet, --silent
       suppress progress indicators

AUTHOR
       Written by Peter Miller.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.

COPYRIGHT
       Copyright (C) 1995-1998, 2000-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO  warranty;  not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
       PURPOSE.

SEE ALSO
       The full documentation for msgmerge is maintained as a Texinfo  manual.
       If  the info and msgmerge programs are properly installed at your site,
       the command

       info msgmerge

       should give you access to the complete manual.
```

## msgunfmt

This tool decompiles an MO file to a PO file.

```
NAME
       msgunfmt - uncompile message catalog from binary format

SYNOPSIS
       msgunfmt [OPTION] [FILE]...

DESCRIPTION
       Convert binary message catalog to Uniforum style .po file.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

   Operation mode:
       -j, --java
       Java mode: input is a Java ResourceBundle class

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

```
Input file location:
    FILE ...
    input .mo files

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

Input file location in Java mode:
    -r, --resource=RESOURCE
    resource name

    -l, --locale=LOCALE
    locale name, either language or language_COUNTRY

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

Input file location in Tcl mode:
    -l, --locale=LOCALE
    locale name, either language or language_COUNTRY

    -d DIRECTORY
    base directory of .msg message catalogs

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

Output file location:
    -o, --output-file=FILE
    write output to specified file

    The  results are written to standard output if no output file is speci-

    fied or if it is -.

Output details:
    -e, --no-escape
    do not use C escapes in output (default)

    -E, --escape
    use C escapes in output, no extended chars

    --force-po
    write PO file even if empty

    -i, --indent
    write indented output style

    --strict
    write strict uniforum style

    -w, --width=NUMBER
    set output page width

    --no-wrap
    do not break long message lines, longer  than  the  output  page
    width, into several lines

    -s, --sort-output
    generate sorted output

Informative output:
    -h, --help
    display this help and exit
```

```
       -V, --version
       output version information and exit

       -v, --verbose
       increase verbosity level

AUTHOR
       Written by Ulrich Drepper.

REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.

COPYRIGHT
       Copyright (C) 1995-1998, 2000-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO warranty; not even for MERCHANTABILITY or FITNESS FOR  A  PARTICULAR
       PURPOSE.

SEE ALSO
       The  full documentation for msgunfmt is maintained as a Texinfo manual.
       If the info and msgunfmt programs are properly installed at your  site,
       the command

       info msgunfmt

       should give you access to the complete manual.
```

## msguniq

In case you did something to your PO files, that makes them contain duplicate messages, you can use msguniq to unify those duplicates.

```
NAME
       msguniq - unify duplicate translations in message catalog

SYNOPSIS
       msguniq [OPTION] [INPUTFILE]

DESCRIPTION
       Unifies duplicate translations in a translation catalog.  Finds dupli-

       cate translations of the same message ID.  Such duplicates are  invalid
       input  for other programs like msgfmt, msgmerge or msgcat.  By default,
       duplicates are merged together. When using the --repeated option, only
       duplicates  are output, and all other messages are discarded.  Comments
       and extracted comments will be cumulated, except that if --use-
first is
       specified,  they  will be taken from the first translation.  File posi-

       tions will be cumulated.  When using the  --unique  option,  duplicates
       are discarded.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

   Input file location:
       INPUTFILE
       input PO file

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

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

```
   Output file location:
       -o, --output-file=FILE
       write output to specified file

       The results are written to standard output if no output file is speci-

       fied or if it is -.

   Message selection:
       -d, --repeated
       print only duplicates

       -u, --unique
       print only unique messages, discard duplicates

   Output details:
       -t, --to-code=NAME
       encoding for output

       --use-first
       use  first  available  translation for each message, don't merge
       several translations

       -e, --no-escape
       do not use C escapes in output (default)

       -E, --escape
       use C escapes in output, no extended chars

       --force-po
       write PO file even if empty

       -i, --indent
       write the .po file using indented style

       --no-location
       do not write '#: filename:line' lines

       -n, --add-location
       generate '#: filename:line' lines (default)

       --strict
       write out strict Uniforum conforming .po file

       -w, --width=NUMBER
       set output page width

       --no-wrap
       do not break long message lines, longer  than  the  output  page
       width, into several lines

       -s, --sort-output
       generate sorted output

       -F, --sort-by-file
       sort output by file location

   Informative output:
       -h, --help
       display this help and exit

       -V, --version
       output version information and exit

AUTHOR
       Written by Bruno Haible.
```

```
REPORTING BUGS
       Report bugs to <bug-gnu-gettext@gnu.org>.

COPYRIGHT
       Copyright (C) 2001-2002 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO warranty; not even for MERCHANTABILITY or FITNESS FOR  A  PARTICULAR
       PURPOSE.

SEE ALSO
       The  full  documentation for msguniq is maintained as a Texinfo manual.
       If the info and msguniq programs are properly installed at  your  site,
       the command

       info msguniq

       should give you access to the complete manual.
```

## xgettext

This tool is the GNU equivalent to the dxgettext command line tool. It does not work with Delphi, but it works with C and C++.

```
NAME
       xgettext - extract gettext strings from source

SYNOPSIS
       xgettext [OPTION] [INPUTFILE]...

DESCRIPTION
       Extract translatable strings from given input files.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.  Similarly for optional arguments.

   Input file location:
       INPUTFILE ...
       input files

       -f, --files-from=FILE
       get list of input files from FILE

       -D, --directory=DIRECTORY
       add DIRECTORY to list for input files search

       If input file is -, standard input is read.

   Output file location:
       -d, --default-domain=NAME
       use NAME.po for output (instead of messages.po)

       -o, --output=FILE
       write output to specified file

       -p, --output-dir=DIR
       output files will be placed in directory DIR

       If output file is -, output is written to standard output.

   Choice of input file language:
       -L, --language=NAME
       recognise  the  specified  language  (C, C++,  ObjectiveC,  PO,
       Python,  Lisp,  EmacsLisp,  librep,  Java,  awk, YCP, Tcl, RST,
```

```
    Glade)

    -C, --c++
    shorthand for --language=C++

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

Operation mode:
    -j, --join-existing
    join messages with existing file

    -x, --exclude-file=FILE.po
    entries from FILE.po are not extracted

    -c, --add-comments[=TAG]
    place  comment block with TAG (or those preceding keyword lines)
    in output file

Language=C/C++ specific options:
    -a, --extract-all
    extract all strings

    -k, --keyword[=WORD]
    additional keyword to be looked for (without WORD means  not  to
    use default keywords)

    -T, --trigraphs
    understand ANSI C trigraphs for input

    --debug
    more detailed formatstring recognition result

Output details:
    -e, --no-escape
    do not use C escapes in output (default)

    -E, --escape
    use C escapes in output, no extended chars

    --force-po
    write PO file even if empty

    -i, --indent
    write the .po file using indented style

    --no-location
    do not write '#: filename:line' lines

    -n, --add-location
    generate '#: filename:line' lines (default)

    --strict
    write out strict Uniforum conforming .po file

    -w, --width=NUMBER
    set output page width

    --no-wrap
    do  not  break  long  message lines, longer than the output page
    width, into several lines

    -s, --sort-output
    generate sorted output

    -F, --sort-by-file
```

```
              sort output by file location

              --omit-header
              don't write header with 'msgid ""' entry

              --copyright-holder=STRING
              set copyright holder in output

              --foreign-user
              omit FSF copyright in output for foreign user

              -m, --msgstr-prefix[=STRING]
              use STRING or "" as prefix for msgstr entries

              -M, --msgstr-suffix[=STRING]
              use STRING or "" as suffix for msgstr entries

         Informative output:
              -h, --help
              display this help and exit

              -V, --version
              output version information and exit

       AUTHOR
              Written by Ulrich Drepper.

       REPORTING BUGS
              Report bugs to <bug-gnu-gettext@gnu.org>.

       COPYRIGHT
              Copyright (C) 1995-1998, 2000-2002 Free Software Foundation, Inc.
              This is free software; see the source for copying conditions.  There is
              NO  warranty;  not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
              PURPOSE.

       SEE ALSO
              The full documentation for xgettext is maintained as a Texinfo  manual.
              If  the info and xgettext programs are properly installed at your site,
              the command

              info xgettext

              should give you access to the complete manual.
```

# Appendix E. GUI tools reference

## PO files

In Windows Explorer, you can click with your right mouse button on po files and choose "Compile to mo file" and "Merge template". If you have installed a po file editor, like poEdit[1], you can also just open the po file.

## MO files

In Windows Explorer, you can click with your right mouse button on po files and choose "Decompile to po file".

## Executables (DLL, EXE files)

In Windows Explorer, you can click with your right mouse button on po files and choose "Extract Strings" and "Embed translations". The first will extract all strings from the resource part of the file into a po file, and the second will append all translation files (mo files) from the locale subdirectory to the executable file.

## File folders

In Windows Explorer, you can click with your right mouse button on a file folder and choose "Extract translations to template". This will enable you to scan a lot of source code files for strings to translate.

## Notes

1. http://poedit.sf.net/

# Appendix F. Standards

## ISO 639 language codes

```
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
bo Tibetan
br Breton
bs Bosnian
ca Catalan
ce Chechen
ch Chamorro
co Corsican
cr Cree
cs Czech
cv Chuvash
cy Welsh
da Danish
de German
dv Divehi
dz Dzongkha
ee Ewe
el Greek
en English
en_US American English
en_GB British English
en_AU Australian English
eo Esperanto
es Spanish
et Estonian
eu Basque
fa Persian
ff Fulah
fi Finnish
fj Fijian
fo Faroese
fr French
fr_BE Walloon
fy Frisian
ga Irish
gd Gaelic
gl Gallegan
gn Guarani
gu Gujarati
gv Manx
ha Hausa
he Hebrew
```

```
hi Hindi
ho Hiri Motu
hr Croatian
ht Haitian
hu Hungarian
hy Armenian
hz Herero
ia Interlingua
id Indonesian
ie Interlingue
ig Igbo
ii Sichuan Yi
ik Inupiaq
io Ido
is Icelandic
it Italian
iu Inuktitut
ja Japanese
jv Javanese
ka Georgian
kg Kongo
ki Kikuyu
kj Kuanyama
kk Kazakh
kl Greenlandic
km Khmer
kn Kannada
ko Korean
kr Kanuri
ks Kashmiri
ku Kurdish
kw Cornish
kv Komi
ky Kirghiz
la Latin
lb Luxembourgish
lg Ganda
li Limburgan
ln Lingala
lo Lao
lt Lithuanian
lu Luba-Katanga
lv Latvian
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 Bokmaal
nd Ndebele, North
ne Nepali
ng Ndonga
nl Dutch
nl_BE Flemish
nn Norwegian Nynorsk
no Norwegian
nr Ndebele, South
nv Navajo
ny Chichewa
```

```
oc Occitan
oj Ojibwa
om Oromo
or Oriya
os Ossetian
pa Panjabi
pi Pali
pl Polish
ps Pushto
pt Portuguese
qu Quechua
rm Raeto-Romance
rn Rundi
ro Romanian
ru Russian
rw Kinyarwanda
sa Sanskrit
sc Sardinian
sd Sindhi
se Northern Sami
sg Sango
si Sinhalese
sk Slovak
sl Slovenian
sm Samoan
sn Shona
so Somali
sq Albanian
sr Serbian
ss Swati
st Sotho, Southern
su Sundanese
sv Swedish
sw Swahili
ta Tamil
te Telugu
tg Tajik
th Thai
ti Tigrinya
tk Turkmen
tl Tagalog
tn Tswana
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 Volapuk
wa Walloon
wo Wolof
xh Xhosa
yi Yiddish
yo Yoruba
za Zhuang
zh Chinese
zu Zulu
```

## ISO 3166 country codes

**Country:** AFGHANISTAN
**Code:** AF

**Country:** ALBANIA
**Code:** AL

**Country:** ALGERIA
**Code:** DZ

**Country:** AMERICAN SAMOA
**Code:** AS

**Country:** ANDORRA
**Code:** AD

**Country:** ANGOLA
**Code:** AO

**Country:** ANGUILLA
**Code:** AI

**Country:** ANTARCTICA
**Code:** AQ

**Country:** ANTIGUA AND BARBUDA
**Code:** AG

**Country:** ARGENTINA
**Code:** AR

**Country:** ARMENIA
**Code:** AM

**Country:** ARUBA
**Code:** AW

**Country:** AUSTRALIA
**Code:** AU

**Country:** AUSTRIA
**Code:** AT

**Country:** AZERBAIJAN
**Code:** AZ

**Country:** BAHAMAS
**Code:** BS

**Country:** BAHRAIN
**Code:** BH

**Country:** BANGLADESH
**Code:** BD

**Country:** BARBADOS
**Code:** BB

**Country:** BELARUS
**Code:** BY

**Country:** BELGIUM
**Code:** BE

**Country:** BELIZE
**Code:** BZ

**Country:** BENIN
**Code:** BJ

**Country:** BERMUDA
**Code:** BM

**Country:** BHUTAN
**Code:** BT

**Country:** BOLIVIA
**Code:** BO

**Country:** BOSNIA AND HERZEGOVINA
**Code:** BA

**Country:** BOTSWANA
**Code:** BW

**Country:** BOUVET ISLAND
**Code:** BV

**Country:** BRAZIL
**Code:** BR

**Country:** BRITISH INDIAN OCEAN TERRITORY
**Code:** IO

**Country:** BRUNEI DARUSSALAM
**Code:** BN

**Country:** BULGARIA
**Code:** BG

**Country:** BURKINA FASO
**Code:** BF

**Country:** BURUNDI
**Code:** BI

**Country:** CAMBODIA
**Code:** KH

**Country:** CAMEROON
**Code:** CM

**Country:** CANADA
**Code:** CA

**Country:** CAPE VERDE
**Code:** CV

**Country:** CAYMAN ISLANDS
**Code:** KY

**Country:** CENTRAL AFRICAN REPUBLIC

**Code:** CF

**Country:** CHAD
**Code:** TD

**Country:** CHILE
**Code:** CL

**Country:** CHINA
**Code:** CN

**Country:** CHRISTMAS ISLAND
**Code:** CX

**Country:** COCOS (KEELING) ISLANDS
**Code:** CC

**Country:** COLOMBIA
**Code:** CO

**Country:** COMOROS
**Code:** KM

**Country:** CONGO
**Code:** CG

**Country:** CONGO, THE DEMOCRATIC REPUBLIC OF THE
**Code:** CD

**Country:** COOK ISLANDS
**Code:** CK

**Country:** COSTA RICA
**Code:** CR

**Country:** COTE D'IVOIRE
**Code:** CI

**Country:** CROATIA
**Code:** HR

**Country:** CUBA
**Code:** CU

**Country:** CYPRUS
**Code:** CY

**Country:** CZECH REPUBLIC
**Code:** CZ

**Country:** DENMARK
**Code:** DK

**Country:** DJIBOUTI
**Code:** DJ

**Country:** DOMINICA
**Code:** DM

**Country:** DOMINICAN REPUBLIC
**Code:** DO

**Country:** ECUADOR
**Code:** EC

**Country:** EGYPT
**Code:** EG

**Country:** EL SALVADOR
**Code:** SV

**Country:** EQUATORIAL GUINEA
**Code:** GQ

**Country:** ERITREA
**Code:** ER

**Country:** ESTONIA
**Code:** EE

**Country:** ETHIOPIA
**Code:** ET

**Country:** FALKLAND ISLANDS (MALVINAS)
**Code:** FK

**Country:** FAROE ISLANDS
**Code:** FO

**Country:** FIJI
**Code:** FJ

**Country:** FINLAND
**Code:** FI

**Country:** FRANCE
**Code:** FR

**Country:** FRENCH GUIANA
**Code:** GF

**Country:** FRENCH POLYNESIA
**Code:** PF

**Country:** FRENCH SOUTHERN TERRITORIES
**Code:** TF

**Country:** GABON
**Code:** GA

**Country:** GAMBIA
**Code:** GM

**Country:** GEORGIA
**Code:** GE

**Country:** GERMANY
**Code:** DE

**Country:** GHANA
**Code:** GH

**Country:** GIBRALTAR

**Code:** GI

**Country:** GREECE
**Code:** GR

**Country:** GREENLAND
**Code:** GL

**Country:** GRENADA
**Code:** GD

**Country:** GUADELOUPE
**Code:** GP

**Country:** GUAM
**Code:** GU

**Country:** GUATEMALA
**Code:** GT

**Country:** GUINEA
**Code:** GN

**Country:** GUINEA-BISSAU
**Code:** GW

**Country:** GUYANA
**Code:** GY

**Country:** HAITI
**Code:** HT

**Country:** HEARD ISLAND AND MCDONALD ISLANDS
**Code:** HM

**Country:** HOLY SEE (VATICAN CITY STATE)
**Code:** VA

**Country:** HONDURAS
**Code:** HN

**Country:** HONG KONG
**Code:** HK

**Country:** HUNGARY
**Code:** HU

**Country:** ICELAND
**Code:** IS

**Country:** INDIA
**Code:** IN

**Country:** INDONESIA
**Code:** ID

**Country:** IRAN, ISLAMIC REPUBLIC OF
**Code:** IR

**Country:** IRAQ
**Code:** IQ

**Country:** IRELAND
**Code:** IE

**Country:** ISRAEL
**Code:** IL

**Country:** ITALY
**Code:** IT

**Country:** JAMAICA
**Code:** JM

**Country:** JAPAN
**Code:** JP

**Country:** JORDAN
**Code:** JO

**Country:** KAZAKHSTAN
**Code:** KZ

**Country:** KENYA
**Code:** KE

**Country:** KIRIBATI
**Code:** KI

**Country:** KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF
**Code:** KP

**Country:** KOREA, REPUBLIC OF
**Code:** KR

**Country:** KUWAIT
**Code:** KW

**Country:** KYRGYZSTAN
**Code:** KG

**Country:** LAO PEOPLE'S DEMOCRATIC REPUBLIC
**Code:** LA

**Country:** LATVIA
**Code:** LV

**Country:** LEBANON
**Code:** LB

**Country:** LESOTHO
**Code:** LS

**Country:** LIBERIA
**Code:** LR

**Country:** LIBYAN ARAB JAMAHIRIYA
**Code:** LY

**Country:** LIECHTENSTEIN
**Code:** LI

**Country:** LITHUANIA

**Code:** LT

**Country:** LUXEMBOURG
**Code:** LU

**Country:** MACAO
**Code:** MO

**Country:** MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF
**Code:** MK

**Country:** MADAGASCAR
**Code:** MG

**Country:** MALAWI
**Code:** MW

**Country:** MALAYSIA
**Code:** MY

**Country:** MALDIVES
**Code:** MV

**Country:** MALI
**Code:** ML

**Country:** MALTA
**Code:** MT

**Country:** MARSHALL ISLANDS
**Code:** MH

**Country:** MARTINIQUE
**Code:** MQ

**Country:** MAURITANIA
**Code:** MR

**Country:** MAURITIUS
**Code:** MU

**Country:** MAYOTTE
**Code:** YT

**Country:** MEXICO
**Code:** MX

**Country:** MICRONESIA, FEDERATED STATES OF
**Code:** FM

**Country:** MOLDOVA, REPUBLIC OF
**Code:** MD

**Country:** MONACO
**Code:** MC

**Country:** MONGOLIA
**Code:** MN

**Country:** MONTSERRAT
**Code:** MS

**Country:** MOROCCO
**Code:** MA

**Country:** MOZAMBIQUE
**Code:** MZ

**Country:** MYANMAR
**Code:** MM

**Country:** NAMIBIA
**Code:** NA

**Country:** NAURU
**Code:** NR

**Country:** NEPAL
**Code:** NP

**Country:** NETHERLANDS
**Code:** NL

**Country:** NETHERLANDS ANTILLES
**Code:** AN

**Country:** NEW CALEDONIA
**Code:** NC

**Country:** NEW ZEALAND
**Code:** NZ

**Country:** NICARAGUA
**Code:** NI

**Country:** NIGER
**Code:** NE

**Country:** NIGERIA
**Code:** NG

**Country:** NIUE
**Code:** NU

**Country:** NORFOLK ISLAND
**Code:** NF

**Country:** NORTHERN MARIANA ISLANDS
**Code:** MP

**Country:** NORWAY
**Code:** NO

**Country:** OMAN
**Code:** OM

**Country:** PAKISTAN
**Code:** PK

**Country:** PALAU
**Code:** PW

**Country:** PALESTINIAN TERRITORY, OCCUPIED

**Code:** PS

**Country:** PANAMA
**Code:** PA

**Country:** PAPUA NEW GUINEA
**Code:** PG

**Country:** PARAGUAY
**Code:** PY

**Country:** PERU
**Code:** PE

**Country:** PHILIPPINES
**Code:** PH

**Country:** PITCAIRN
**Code:** PN

**Country:** POLAND
**Code:** PL

**Country:** PORTUGAL
**Code:** PT

**Country:** PUERTO RICO
**Code:** PR

**Country:** QATAR
**Code:** QA

**Country:** REUNION
**Code:** RE

**Country:** ROMANIA
**Code:** RO

**Country:** RUSSIAN FEDERATION
**Code:** RU

**Country:** RWANDA
**Code:** RW

**Country:** SAINT HELENA
**Code:** SH

**Country:** SAINT KITTS AND NEVIS
**Code:** KN

**Country:** SAINT LUCIA
**Code:** LC

**Country:** SAINT PIERRE AND MIQUELON
**Code:** PM

**Country:** SAINT VINCENT AND THE GRENADINES
**Code:** VC

**Country:** SAMOA
**Code:** WS

**Country:** SAN MARINO
**Code:** SM

**Country:** SAO TOME AND PRINCIPE
**Code:** ST

**Country:** SAUDI ARABIA
**Code:** SA

**Country:** SENEGAL
**Code:** SN

**Country:** SERBIA AND MONTENEGRO
**Code:** CS

**Country:** SEYCHELLES
**Code:** SC

**Country:** SIERRA LEONE
**Code:** SL

**Country:** SINGAPORE
**Code:** SG

**Country:** SLOVAKIA
**Code:** SK

**Country:** SLOVENIA
**Code:** SI

**Country:** SOLOMON ISLANDS
**Code:** SB

**Country:** SOMALIA
**Code:** SO

**Country:** SOUTH AFRICA
**Code:** ZA

**Country:** SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS
**Code:** GS

**Country:** SPAIN
**Code:** ES

**Country:** SRI LANKA
**Code:** LK

**Country:** SUDAN
**Code:** SD

**Country:** SURINAME
**Code:** SR

**Country:** SVALBARD AND JAN MAYEN
**Code:** SJ

**Country:** SWAZILAND
**Code:** SZ

**Country:** SWEDEN

**Code:** SE

**Country:** SWITZERLAND
**Code:** CH

**Country:** SYRIAN ARAB REPUBLIC
**Code:** SY

**Country:** TAIWAN, PROVINCE OF CHINA
**Code:** TW

**Country:** TAJIKISTAN
**Code:** TJ

**Country:** TANZANIA, UNITED REPUBLIC OF
**Code:** TZ

**Country:** THAILAND
**Code:** TH

**Country:** TIMOR-LESTE
**Code:** TL

**Country:** TOGO
**Code:** TG

**Country:** TOKELAU
**Code:** TK

**Country:** TONGA
**Code:** TO

**Country:** TRINIDAD AND TOBAGO
**Code:** TT

**Country:** TUNISIA
**Code:** TN

**Country:** TURKEY
**Code:** TR

**Country:** TURKMENISTAN
**Code:** TM

**Country:** TURKS AND CAICOS ISLANDS
**Code:** TC

**Country:** TUVALU
**Code:** TV

**Country:** UGANDA
**Code:** UG

**Country:** UKRAINE
**Code:** UA

**Country:** UNITED ARAB EMIRATES
**Code:** AE

**Country:** UNITED KINGDOM
**Code:** GB

**Country:** UNITED STATES
**Code:** US

**Country:** UNITED STATES MINOR OUTLYING ISLANDS
**Code:** UM

**Country:** URUGUAY
**Code:** UY

**Country:** UZBEKISTAN
**Code:** UZ

**Country:** VANUATU
**Code:** VU

**Country:** VENEZUELA
**Code:** VE

**Country:** VIET NAM
**Code:** VN

**Country:** VIRGIN ISLANDS, BRITISH
**Code:** VG

**Country:** VIRGIN ISLANDS, U.S.
**Code:** VI

**Country:** WALLIS AND FUTUNA
**Code:** WF

**Country:** WESTERN SAHARA
**Code:** EH

**Country:** YEMEN
**Code:** YE

**Country:** ZAMBIA
**Code:** ZM

**Country:** ZIMBABWE
**Code:** ZW

# Appendix G. File formats

## The format of GNU PO files

This section was last updated august 2nd, 2003 by copying the appropriate section from the GNU gettext manual.

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

```
white-space
#  translator-comments
#. automatic-comments
#: reference...
#, flag...
msgid "untranslated-string"
msgstr "translated-string"
```

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

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

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

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

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

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

**c-format:** These flags should not be added by a human. Instead only the `xgettext` program adds them. In an automated PO file processing system as proposed here the user

changes would be thrown away again as soon as the `xgettext` program generates a new template file.

In case the `c-format` flag is given for a string the `msgfmt` does some more tests to check to validity of the translation.

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

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

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

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

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

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

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

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

Outside strings, white lines and comments may be used freely. Comments start at the beginning of a line with # and extend until the end of the PO file line. Comments written by translators should have the initial # immediately followed by some white space. If the # is not immediately followed by white space, this comment is most likely generated and managed by specialized GNU tools, and might disappear or be replaced unexpectedly when the PO file is given to `msgmerge`.

## The format of GNU MO files

This section was last updated august 2nd, 2003 by copying the appropriate section from the GNU gettext manual.

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

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

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

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

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

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

As for the strings themselves, they follow the hash file, and each is terminated with a `NUL`, and this `NUL` is not counted in the length which appears in the string descriptor. The msgfmt

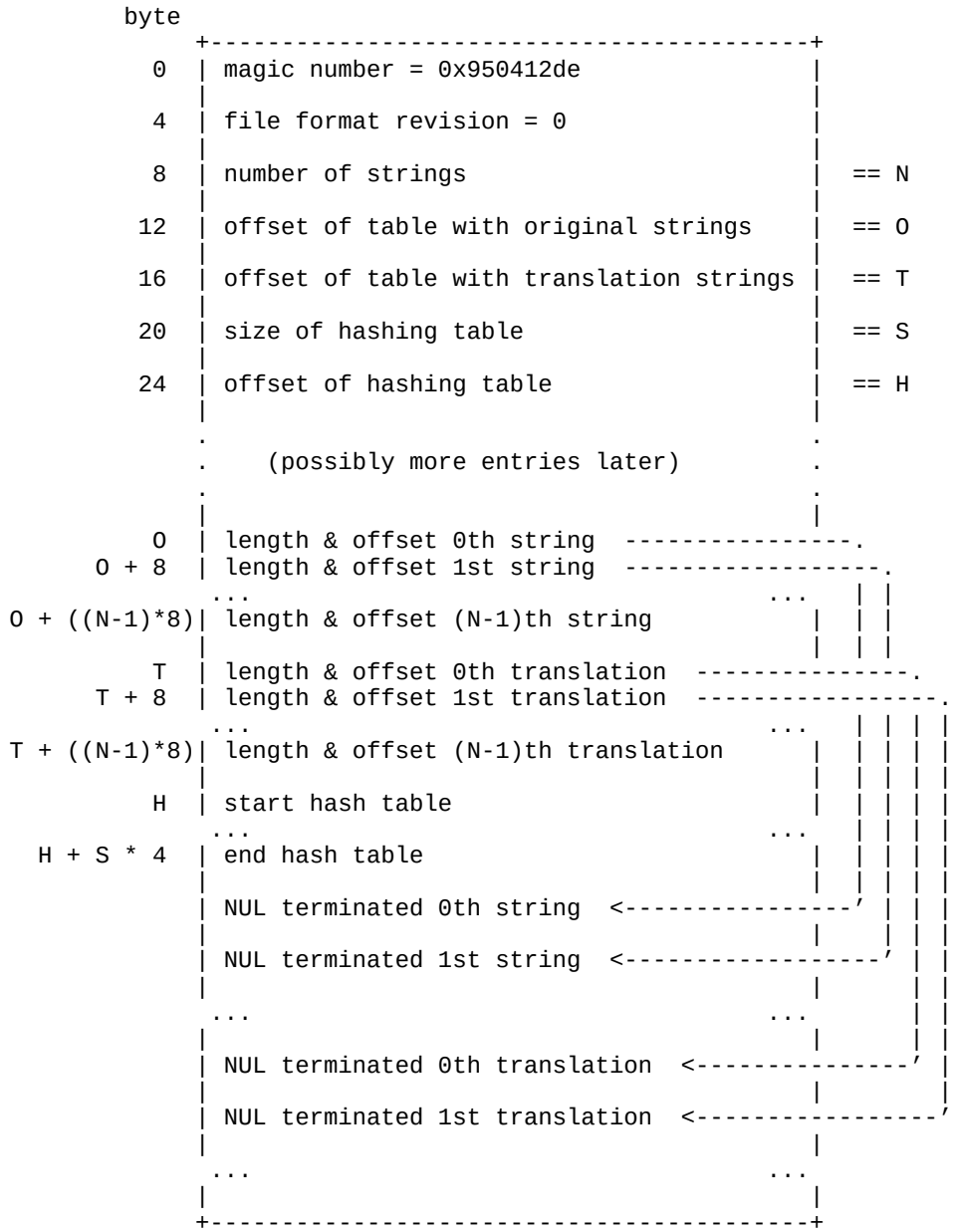
program has an option selecting the alignment for MO file strings. With this option, each string is separately aligned so it starts at an offset which is a multiple of the alignment value. On some RISC machines, a correct alignment will speed things up.

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

Nothing prevents a MO file from having embedded `NUL`s in strings. However, the program interface currently used already presumes that strings are `NUL` terminated, so embedded `NUL`s are somewhat useless. But the MO file format is general enough

so other interfaces would be later possible, if for example, we ever want to implement wide characters right in MO files, where `NUL` bytes may accidently appear. (No, we don't want to have wide characters in MO files. They would make the file unnecessarily large, and the `wchar_t` type being platform dependent, MO files would be platform dependent as well.)

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



# Appendix H. How to handle specific classes

This appendix contains documentation on how to handle various classes that do not translate easily using `TranslateComponent()` or `TranslateProperties()`. Most classes are handled easily by just putting an ignore on some of their properties, while other classes need more advanced handling.

Please note, that the TComponent.Name property is always ignored by default, and doesn't need to be specified.

## VCL, important ones

```
TP_GlobalIgnoreClassProperty(TAction,'Category');
TP_GlobalIgnoreClassProperty(TControl,'HelpKeyword');
TP_GlobalIgnoreClassProperty(TNotebook,'Pages');
```

## VCL, not so important

These are normally not needed.

```
TP_GlobalIgnoreClassProperty(TControl,'ImeName');
TP_GlobalIgnoreClass(TFont);
```

## Database (DB unit)

Field names and table names often tend to have names that are also used for other purposes elsewhere in the program. Therefore, it is very wise to add this somewhere in your program if you are using databases.

```
TP_GlobalIgnoreClassProperty(TField,'DefaultExpression');
TP_GlobalIgnoreClassProperty(TField,'FieldName');
TP_GlobalIgnoreClassProperty(TField,'KeyFields');
TP_GlobalIgnoreClassProperty(TField,'DisplayName');
TP_GlobalIgnoreClassProperty(TField,'LookupKeyFields');
TP_GlobalIgnoreClassProperty(TField,'LookupResultField');
TP_GlobalIgnoreClassProperty(TField,'Origin');
TP_GlobalIgnoreClass(TParam);
TP_GlobalIgnoreClassProperty(TFieldDef,'Name');
```

## MIDAS/Datasnap

```
TP_GlobalIgnoreClassProperty(TClientDataset,'CommandText');
TP_GlobalIgnoreClassProperty(TClientDataset,'Filename');
TP_GlobalIgnoreClassProperty(TClientDataset,'Filter');
TP_GlobalIgnoreClassProperty(TClientDataset,'IndexFieldnames');
TP_GlobalIgnoreClassProperty(TClientDataset,'IndexName');
TP_GlobalIgnoreClassProperty(TClientDataset,'MasterFields');
TP_GlobalIgnoreClassProperty(TClientDataset,'Params');
TP_GlobalIgnoreClassProperty(TClientDataset,'ProviderName');
```

## Database controls

```
TP_GlobalIgnoreClassProperty(TDBComboBox,'DataField');
TP_GlobalIgnoreClassProperty(TDBCheckBox,'DataField');
TP_GlobalIgnoreClassProperty(TDBEdit,'DataField');
TP_GlobalIgnoreClassProperty(TDBImage,'DataField');
TP_GlobalIgnoreClassProperty(TDBListBox,'DataField');
TP_GlobalIgnoreClassProperty(TDBLookupControl,'DataField');
TP_GlobalIgnoreClassProperty(TDBLookupControl,'KeyField');
TP_GlobalIgnoreClassProperty(TDBLookupControl,'ListField');
TP_GlobalIgnoreClassProperty(TDBMemo,'DataField');
TP_GlobalIgnoreClassProperty(TDBRadioGroup,'DataField');
TP_GlobalIgnoreClassProperty(TDBRichEdit,'DataField');
TP_GlobalIgnoreClassProperty(TDBText,'DataField');
```

## Interbase Express (IBX)

```
TP_GlobalIgnoreClass(TIBDatabase);
TP_GlobalIgnoreClass(TIBDatabase);
TP_GlobalIgnoreClass(TIBTransaction);
TP_GlobalIgnoreClassProperty(TIBSQL,'UniqueRelationName');
```

## Borland Database Engine (BDE)

```
TP_GlobalIgnoreClass(TSession);
TP_GlobalIgnoreClass(TDatabase);
```

## ADO components

```
TP_GlobalIgnoreClass (TADOConnection);
TP_GlobalIgnoreClassProperty(TADOQuery,'CommandText');
TP_GlobalIgnoreClassProperty(TADOQuery,'ConnectionString');
TP_GlobalIgnoreClassProperty(TADOQuery,'DatasetField');
TP_GlobalIgnoreClassProperty(TADOQuery,'Filter');
TP_GlobalIgnoreClassProperty(TADOQuery,'IndexFieldNames');
TP_GlobalIgnoreClassProperty(TADOQuery,'IndexName');
TP_GlobalIgnoreClassProperty(TADOQuery,'MasterFields');
TP_GlobalIgnoreClassProperty(TADOTable,'IndexFieldNames');
TP_GlobalIgnoreClassProperty(TADOTable,'IndexName');
TP_GlobalIgnoreClassProperty(TADOTable,'MasterFields');
TP_GlobalIgnoreClassProperty(TADOTable,'TableName');
TP_GlobalIgnoreClassProperty(TADODataset,'CommandText');
TP_GlobalIgnoreClassProperty(TADODataset,'ConnectionString');
TP_GlobalIgnoreClassProperty(TADODataset,'DatasetField');
TP_GlobalIgnoreClassProperty(TADODataset,'Filter');
TP_GlobalIgnoreClassProperty(TADODataset,'IndexFieldNames');
TP_GlobalIgnoreClassProperty(TADODataset,'IndexName');
TP_GlobalIgnoreClassProperty(TADODataset,'MasterFields');
```

## ActiveX stuff

```
TP_GlobalIgnoreClass (TWebBrowser);
```

## Turbopower Orpheus

```
TP_GlobalIgnoreClassProperty(TO32FlexEdit,'About');
TP_GlobalIgnoreClassProperty(TO32FlexEdit,'Validation');
TP_GlobalIgnoreClassProperty(TOvcTimeEdit,'About');
TP_GlobalIgnoreClassProperty(TOvcTimeEdit,'NowString');
```

## Turbopower Essentials

```
TP_GlobalIgnoreClassProperty(TEsDateEdit,'TodayString');
```

## TMS Software TAdvStringGrid

The TAdvStringGrid is not compatible with TranslateComponent(). You should therefore put it on ignore:

```
TP_GlobalIgnoreClass (TAdvStringGrid);
```

But there is a solution to get it translated. The following text was provided by Sandro Wendt:

As I needed to translate one of its descendants ( a TAdvColumnGrid ), I checked with Bruno and found that the reason for the exceptions "Controls ″ has no parent window" were probably the inplace editors of the grid. There are two you can get at through properties, but several you cannot as they are only contained in private member fields. However, using the components array of the grid, you can get at these as well.

This is the code I used:

```
var
  i: integer;
  FCompList: TObjectList;
begin
  FCompList := TObjectList.Create( false );
  try
    for i := 0 to FieldDefinition_grid.ComponentCount - 1 do begin
      if FieldDefinition_grid.Components[i] is TWinControl then begin
        if TWinControl( FieldDefinition_grid.Components[i] ).Parent = nil then begin
          TWinControl( FieldDefinition_grid.Components[i] ).Parent := FieldDefinition_g
          FCompList.Add( FieldDefinition_grid.Components[i] );
        end;
      end;
    end;
    TranslateComponent (self);
    for i := 0 to FCompList.Count - 1 do begin
      TWinControl( FCompList[i] ).Parent := nil;
    end;
  finally
    FCompList.Free;
  end;
end;
```

You need to set the Parent's back to nil because otherwise especially the RichEdit editors will display on top of the grid. Also, most inplace editors actually have a parent assigned, so you only want to nil the ones you gave a parent in order for the translation to succeed.

# Appendix I. Frequently Asked Questions

### 1. About this project

**1.1.** Is anybody using this?

Definitely. The concept, file formats and many of the tools are exactly the same as are used to localize most Linux, KDE, Gnome, Unix software. It is also emerging as a very common translation tool on Windows. Many thousands of programs have been localized using GNU gettext.

**1.2.** Is it productive?

Definitely. It goes a long way to reduce the amount of work that has to be done by the translator. Several tools, including KBabel, even provide automated raw translations based on online dictionaries, so that the translator's job is reduced to finding incorrect translations. You'll be amazed at how productive this translation environment is once you get started.

**1.3.** I use the Delphi Integrated Translation Environment (ITE) - should I switch?

A switch means that you have to change something, and all changes have a cost. But if you still do development on your application, or if you are starting up something completely new, the chance is very high that a switch pays off quickly in terms of Return of Investment. If you are part of a large programming group, and you use the Delphi ITE, try to ask your Boss how much money you spend on translators, and whether a 50% reduction on these costs for every future release would be a nice thing.

**1.4.** Why is this project called dxgettext?

The original GNU gettext software includes a tool named xgettext, that extracts strings from a lot of different file formats, including C and C++ source code. The main tool in this project is the one that extracts strings from Delphi source code, so that tool was named dxgettext.

**1.5.** What's the catch? (Why is it free?)

There is no catch. Developing a product is only a very small part of bringing a product to the market. If the authors of this translation system would try to earn money on it, there would a lot of work involved:

- Raising money to pay us. At least one guy needs to be fulltime on a commercial project
- Marketing - this costs money when doing it commercially. Nobody wants to pay for software they weren't told about.
- Documentation - the current documentation is not near anything that a commercial product requires. Much commercial software also sells better if you make printed documentation, which means that you actually have to make printed documentation to make your project succeed.
- CDs - customer's don't like receiving all software online, some simply want CDs. And they don't want CD-R's, they want real CDs. That costs money.
- Release process - commercial software requires you to test your software with all possible environments before releasing it. That's a lot of work.
- Software design - in order to make software sell easily, you need to design the software for it. Screenshots are everything, and GUIs are needed everywhere, and the GUIs need to use the latest GUI features from Microsoft. Sometimes a graphical designer is needed to make things look expensive.

- Administration - all the above has to be administrated. When money is involved, somebody has to be in charge etc.

What we do now is much easier:

- No money needed. We use our spare time to do it. If we don't have spare time, nothing is being done on the project.
- Marketing. When things are free, people talk about it. SourceForge tells about it. We can announce it in newsgroups. We are very well placed on Google, probably because of being on SourceForge. It doesn't cost money and doesn't take time.
- Documentation - well, documentation is always boring, but people can live with less when they don't pay for it. Hopefully we have enough documentation, otherwise let us know.
- Release process - the first releases were quite buggy because of lack of testing. But the users commented on it, and in cooperation the quality of the releases have improved a lot, and the last many releases should not have any serious bugs inside. This would never have worked this way commercially, but it does as long as it is free.
- Software design. Free software only has one goal: Being the best choice for those who use it. Since that's how we prefer software, too, there is no conflict between marketing and programmers on how the software should be.
- Administration - SourceForge and YahooGroups make much of our administration easier, and both would be unthinkable for commercial projects. We don't need to create a formal organization, because if somebody gets angry at the project he can just take all the source-code and start another project as a branch of this project. Of course we hope that this never happens, but this mechanism means that people actually cooperate.

If you want to know more about why Open Source software works, Eric S. Raymond has written a free book about it: The Cathedral and the Bazaar[1]

When Delphi developers cooperate on making Delphi better, we make Delphi a better choice, and thus make us Delphi developers a better choice than those programmers using other tools.

**1.6.** I have a question that is not on this list

Send an empty e-mail to dxgettext-subscribe@yahoogroups.com to join our mailing list, and then write your question to dxgettext@yahoogroups.com

You can also send an e-mail directly to the maintainer of this FAQ: Lars@dybdahl.dk

We're always glad to help you out, and always happy to get feedback from our users.

## 2. Considering to use this software

**2.1.** My program is not in English. What do I do?

Very easy, you do this:

1. Extract all strings from your program as it is now.
2. Translate your program to English.
3. Extract all strings from your program afterwards.

4. Use the msgmergePOT tool to create an English->YourLanguage translation file.

Now you have an English language program with a translation to the language that your program used before.

**2.2.** Can I use this to translate a German language program to English?

Yes, but it will only work on computers that use the same character set, i.e. iso-8859-1. You can convert your application to English with the method specified in the question above.

**2.3.** Does this support Unicode or widestrings?

Yes, this system supports widestrings all along, and actually does it much better than the Delphi ITE. But please note, that Delphi has built-in limitations. Delphi's VCL does not do Unicode, and resourcestring retrieval isn't Unicode either.

But if you get Unicode components and use the features of GNU gettext for Delphi, you can create an all-through Unicode program.

**2.4.** Does it work without Unicode or widestrings?

Yes. The gettext() function returns widestring, and Delphi will automatically convert the strings to the local 8-bit character set when you use gettext() where you don't use widestrings otherwise. For instance, if you assign:

```
MyButton.Caption:=gettext('New caption');
```

Then gettext will return a widestring and Delphi will convert that to the local 8-bit character set before it is assigned to the caption.

**2.5.** My program uses resourcestrings - can gettext handle this?

Yes. It will automatically extract resourcestrings into the .po files, and it will automatic translate the resourcestrings when they are used in your source code.

**3. Something is not being translated**

**3.1.** I want to know exactly why something isn't translated

As of version 1.1, there is a setting in gnugettext.pas, that will output a lot of details about the translation system into a logfile. Switch on this feature and you can see exactly how the translation system sees the world.

**3.2.** I use TObject derivatives that are not derived from TComponent, how do I translate them?

The "TranslateProperties()" function handles all kinds of objects well, even TCollections, as long as they are derived from TPersistent. This includes report components, network components etc.

**3.3.** My 'This is version '+Version+' of my program' is not extracted

Dxgettext has some logic, but it can't read Pascal code like the Delphi compiler does. Therefore, in the above example, Version must be defined in the same unit, otherwise dxgettext isn't able to extract it well.

**3.4.** Some Delphi things are not translated. How do I translate these?

Get a translation for the library you need to have translated. The homepage provides translations for several libraries and for the Delphi runtime library in several languages, but you can also do it yourself, if you have the source code.

If you get a delphi.mo file with the translations for the Delphi runtime library and the VCL, put it together with your own translation and call:

```
AddDomainForResourceString ('delphi');
```

Somewhere in your program before you start translating forms etc.

If you still need support on this subject, please join our mailing list and get instructions there.

**3.5.** How do I translate the Delphi runtime library messages?

See this link[2] for instructions.

**3.6.** My menu items are not translated. Why?

Please note that a TMainMenu has a property named AutoHotkeys. Set this to maManual, or it will automatically change the captions of the menu items, and thus make it impossible to translate the menus correctly.

**3.7.** I have some 3rd party components without source. How do I translate those?

All resourcestrings in the 3rd party components will be translated automatically, so if you can make a list of them and add put them in resourcestrings in your source code, gnugettext.pas will extract the texts and translate them properly.

But if you cannot make a list, or if the 3rd party component contains form resources (like those in a dfm file), you could compile your program with separate packages and include the translation for that package together with the package.

If your application uses the xyz package and is translated to German (language code "de"), you might end up with the following files:

```
appdir\application.exe                    (your program)
appdir\locale\de\LC_MESSAGES\default.mo  (the translation of your pro-
gram)
appdir\locale\de\LC_MESSAGES\delphi.mo   (the translation of the Del-
phi VCL)
winsysdir\xyz.bpl                         (the xyz package)
winsysdir\xyz.de                          (the translation of the xyz pack-
age)
```

**3.8.** Something in my forms isn't translated. What do I do?

The extraction tool doesn't extract all properties from forms in order to make the life for the translators much easier. But even though it extracts a lot of texts, not all text may be translated once you run the program. You can solve this by assigning the text in the Form's OnCreate event like this:

```
component.property:=gettext('This text wasn"t translated in the first place');
```

**4. Concepts**

**4.1.** How does gettext handle two different translations of the same English word?

Experience with thousands of programs shows, that this is extremely rare. When it does happen, it is usually an error. There are a few ways out, though, and the easiest one is to put that text into another domain. Another solution is to add a whitespace and then programmatically remove it again. This will make the text differ from the other English word.

**4.2.** How should I choose text domains?

Most applications only use one text domain, and this set of tools assumes that that text domain is named "default". Normally, only very large projects need multiple text domains, and since it is fairly easy to split a domain into two, you shouldn't worry until your project gets too big for one domain, and then you will probably know by the structure of your project, how to divide the domain into smaller domains. Many projects that have several executables, still only use one single domain for all the applications, because it reduces the amount of work that is needed by the translator.

**4.3.** Why are memos extracted line by line?

Delphi stores memos as a list of strings in dfm files, and does not put any information into the dfm files to tell that these strings come from a TStrings object. Therefore, it is not possible for the string extraction tools to see, that these strings should be assembled into one, big, multiline string in the po file. And since the po file doesn't contain the full memo text as one string, but as several lines, the `TranslateComponent()` procedure cannot do anything else than translate the memos line by line.

Another problem is that some programmers use the TMemo.Lines.Objects[] array to store objects. When you translate such a memo, it is important not to destroy these objects, which would be the case if a translation was assigned to the TMemo.Lines.Text property. By assigning translations to each TMemo.Lines.Strings[] index, this is avoided.

There is way to make it all work, though: Instead of putting the memo contents into the user interface at design time, you can assign the string at runtime like this:

```
MyMemo.Lines.Text:=
  _('This is a demo of my multiline memo translation, where '+
    'the entire memo is translated as one big message.'+sLineBreak+
    'This message even contains a programmed line break using '+
    'the sLineBreak constant, which is equivalent to #13#10 in '+
    'Delphi and #10 in Kylix.');
```

Here, the string extractor will take the entire string as one big message, the translator will translate it all as one big message, and at runtime it will be assigned as one big message.

## 5. Using it

**5.1.** I want to force my program to use another language than Windows settings, how?

Insert this line as the first in your .dpr main program block:

```
UseLanguage ('fr');
```

Here, 'fr' means french. Please note that you can not just switch to a language that uses characters that isn't supported in your Windows. Switching to greek, russian or chinese without the proper fonts etc. won't work.

There is also another possibility: You can set the environment variable "LANG" to the desired language code, e.g. "set LANG=fr". This will override the detection of the Windows language settings.

**5.2.** How do I switch language at runtime?

See the source of the TntSample application that is included in the download for Delphi.

**5.3.** I want to use language XXX but my controls do not support Unicode, what do I do?

The gettext functions return widestrings, which will automatically be converted to the local character set by Delphi when you assign it to Control properties of type string or ansistring. This way, the local 8-bit characterset will be support automatically.

But if you really should need Unicode support in your controls, look for TNT controls. They are free and do Unicode on Windows NT/2000/XP.

**6. Errors**

**6.1.** In Kylix and CLX apps, gettext('Test') cannot compile

In CLX, each form has a gettext() function that will be chosen instead of gnugettext.gettext(). Use _() instead:

```
a:=_('Test');
```

**6.2.** I got an Access Violation. Why?

One of the forms that you translate using TranslateComponent() probably has a component that doesn't like one of its properties translated or has been programmed very badly and cannot be analyzed by TranslateComponent(). The solution is to ignore this component. See the list at Appendix H> for more information, or contact our e-mail list[3] for further information. If you send an e-mail on that list you will most likely get your problem solved quickly.

## Notes

1. http://www.firstmonday.dk/issues/issue3_3/raymond/
2. http://dybdahl.dk/dxgettext/docs/howto-mono.php
3. http://groups.yahoo.com/group/dxgettext/

# Index