

The GNU C Library Reference Manual

The GNU C Library Reference Manual

Sandra Loosemore
with
Richard M. Stallman, Roland McGrath, Andrew Oram, and Ulrich Drepper

Edition 0.09 DRAFT

last updated 28 Aug 1999

for version 2.2 Beta

Copyright © 1993, '94, '95, '96, '97, '98 Free Software Foundation, Inc.

Published by the Free Software Foundation
59 Temple Place – Suite 330,
Boston, MA 02111-1307 USA
Printed copies are available for \$50 each.
ISBN 1-882114-53-1

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

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the section entitled “GNU Library General Public License” is included exactly as in the original, and provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that the text of the translation of the section entitled “GNU Library General Public License” must be approved for accuracy by the Foundation.

We see how the catalog object is opened and the returned descriptor used in the other function calls. It is not really necessary to check for failure of any of the functions since even in these situations the functions will behave reasonable. They simply will be return a translation.

What remains unspecified here are the constants `SetMainSet` and `SetMainHello`. These are the symbolic names describing the message. To get the actual definitions which match the information in the catalog file we have to create the message catalog source file and process it using the `gencat` program.

```
$ Messages for the famous greeting program.
$quote "
```

```
$set Main
Hello "Hallo, Welt!\n"
```

Now we can start building the program (assume the message catalog source file is named `'hello.msg'` and the program source file `'hello.c'`):

```
% gencat -H msgnrs.h -o hello.cat hello.msg
% cat msgnrs.h
#define MainSet 0x1      /* hello.msg:4 */
#define MainHello 0x1    /* hello.msg:5 */
% gcc -o hello hello.c -I.
% cp hello.cat /usr/share/locale/de/LC_MESSAGES
% echo $LC_ALL
de
% ./hello
Hallo, Welt!
%
```

The call of the `gencat` program creates the missing header file `'msgnrs.h'` as well as the message catalog binary. The former is used in the compilation of `'hello.c'` while the later is placed in a directory in which the `catopen` function will try to locate it. Please check the `LC_ALL` environment variable and the default path for `catopen` presented in the description above.

8.2 The Uniform approach to Message Translation

Sun Microsystems tried to standardize a different approach to message translation in the Uniform group. There never was a real standard defined but still the interface was used in Sun's operation systems. Since this approach fits better in the development process of free software it is also used throughout the GNU project and the GNU `'gettext'` package provides support for this outside the GNU C Library.

The code of the `'libintl'` from GNU `'gettext'` is the same as the code in the GNU C Library. So the documentation in the GNU `'gettext'` manual is also valid for the functionality here. The following text will describe the library functions in detail. But the numerous helper programs are not described in this manual. Instead people should read the GNU

‘`gettext`’ manual (see [section “GNU gettext utilities” in *Native Language Support Library and Tools*](#)). We will only give a short overview.

Though the `catgets` functions are available by default on more systems the `gettext` interface is at least as portable as the former. The GNU ‘`gettext`’ package can be used wherever the functions are not available.

8.2.1 The `gettext` family of functions

The paradigms underlying the `gettext` approach to message translations is different from that of the `catgets` functions the basic functionality is equivalent. There are functions of the following categories:

8.2.1.1 What has to be done to translate a message?

The `gettext` functions have a very simple interface. The most basic function just takes the string which shall be translated as the argument and it returns the translation. This is fundamentally different from the `catgets` approach where an extra key is necessary and the original string is only used for the error case.

If the string which has to be translated is the only argument this of course means the string itself is the key. I.e., the translation will be selected based on the original string. The message catalogs must therefore contain the original strings plus one translation for any such string. The task of the `gettext` function is it to compare the argument string with the available strings in the catalog and return the appropriate translation. Of course this process is optimized so that this process is not more expensive than an access using an atomic key like in `catgets`.

The `gettext` approach has some advantages but also some disadvantages. Please see the GNU ‘`gettext`’ manual for a detailed discussion of the pros and cons.

All the definitions and declarations for `gettext` can be found in the ‘`libintl.h`’ header file. On systems where these functions are not part of the C library they can be found in a separate library named ‘`libintl.a`’ (or accordingly different for shared libraries).

<code>char * gettext (const char *msgid)</code>	Function
--	----------

The `gettext` function searches the currently selected message catalogs for a string which is equal to `msgid`. If there is such a string available it is returned. Otherwise the argument string `msgid` is returned.

Please note that although the return value is `char *` the returned string must not be changed. This broken type results from the history of the function and does not reflect the way the function should be used.

Please note that above we wrote “message catalogs” (plural). This is a specialty of the GNU implementation of these functions and we will say more about this when we talk about the ways message catalogs are selected (see [Section 8.2.1.2 \[How to determine which catalog to be used\]](#), page 191).

The `gettext` function does not modify the value of the global `errno` variable. This is necessary to make it possible to write something like

```
printf (gettext ("Operation failed: %m\n"));
```

Here the *errno* value is used in the `printf` function while processing the `%m` format element and if the `gettext` function would change this value (it is called before `printf` is called) we would get a wrong message.

So there is no easy way to detect a missing message catalog beside comparing the argument string with the result. But it is normally the task of the user to react on missing catalogs. The program cannot guess when a message catalog is really necessary since for a user who speaks the language the program was developed in does not need any translation.

The remaining two functions to access the message catalog add some functionality to select a message catalog which is not the default one. This is important if parts of the program are developed independently. Every part can have its own message catalog and all of them can be used at the same time. The C library itself is an example: internally it uses the `gettext` functions but since it must not depend on a currently selected default message catalog it must specify all ambiguous information.

char * dgettext (const char *domainname, const char *msgid) Function

The `dgettext` functions acts just like the `gettext` function. It only takes an additional first argument *domainname* which guides the selection of the message catalogs which are searched for the translation. If the *domainname* parameter is the null pointer the `dgettext` function is exactly equivalent to `gettext` since the default value for the domain name is used.

As for `gettext` the return value type is `char *` which is an anachronism. The returned string must never be modified.

char * dcgettext (const char *domainname, const char *msgid, int category) Function

The `dcgettext` adds another argument to those which `dgettext` takes. This argument *category* specifies the last piece of information needed to localize the message catalog. I.e., the domain name and the locale category exactly specify which message catalog has to be used (relative to a given directory, see below).

The `dgettext` function can be expressed in terms of `dcgettext` by using

```
dcgettext (domain, string, LC_MESSAGES)
```

instead of

```
dgettext (domain, string)
```

This also shows which values are expected for the third parameter. One has to use the available selectors for the categories available in `'locale.h'`. Normally the available values are `LC_CTYPE`, `LC_COLLATE`, `LC_MESSAGES`, `LC_MONETARY`, `LC_NUMERIC`, and `LC_TIME`. Please note that `LC_ALL` must not be used and even though the names might suggest this, there is no relation to the environments variables of this name.

The `dcgettext` function is only implemented for compatibility with other systems which have `gettext` functions. There is not really any situation where it is necessary (or useful) to use a different value but `LC_MESSAGES` in for the *category* parameter. We are dealing with messages here and any other choice can only be irritating.

As for `gettext` the return value type is `char *` which is an anachronism. The returned string must never be modified.

When using the three functions above in a program it is a frequent case that the *msgid* argument is a constant string. So it is worth to optimize this case. Thinking shortly about this one will realize that as long as no new message catalog is loaded the translation of a message will not change. This optimization is actually implemented by the `gettext`, `dgettext` and `dcgettext` functions.

8.2.1.2 How to determine which catalog to be used

The functions to retrieve the translations for a given message have a remarkable simple interface. But to provide the user of the program still the opportunity to select exactly the translation s/he wants and also to provide the programmer the possibility to influence the way to locate the search for catalogs files there is a quite complicated underlying mechanism which controls all this. The code is complicated the use is easy.

Basically we have two different tasks to perform which can also be performed by the `catgets` functions:

1. Locate the set of message catalogs. There are a number of files for different languages and which all belong to the package. Usually they are all stored in the filesystem below a certain directory.

There can be arbitrary many packages installed and they can follow different guidelines for the placement of their files.

2. Relative to the location specified by the package the actual translation files must be searched, based on the wishes of the user. I.e., for each language the user selects the program should be able to locate the appropriate file.

This is the functionality required by the specifications for `gettext` and this is also what the `catgets` functions are able to do. But there are some problems unresolved:

- The language to be used can be specified in several different ways. There is no generally accepted standard for this and the user always expects the program understand what s/he means. E.g., to select the German translation one could write `de`, `german`, or `deutsch` and the program should always react the same.
- Sometimes the specification of the user is too detailed. If s/he, e.g., specifies `de_DE.ISO-8859-1` which means German, spoken in Germany, coded using the ISO 8859-1 character set there is the possibility that a message catalog matching this exactly is not available. But there could be a catalog matching `de` and if the character set used on the machine is always ISO 8859-1 there is no reason why this later message catalog should not be used. (We call this *message inheritance*.)
- If a catalog for a wanted language is not available it is not always the second best choice to fall back on the language of the developer and simply not translate any message. Instead a user might be better able to read the messages in another language and so the user of the program should be able to define an precedence order of languages.

We can divide the configuration actions in two parts: the one is performed by the programmer, the other by the user. We will start with the functions the programmer can use since the user configuration will be based on this.

As the functions described in the last sections already mention separate sets of messages can be selected by a *domain name*. This is a simple string which should be unique for each program part with uses a separate domain. It is possible to use in one program arbitrary many domains at the same time. E.g., the GNU C Library itself uses a domain named `libc` while the program using the C Library could use a domain named `foo`. The important point is that at any time exactly one domain is active. This is controlled with the following function.

char * `textdomain` (const char **domainname*) Function

The `textdomain` function sets the default domain, which is used in all future `gettext` calls, to *domainname*. Please note that `dgettext` and `dcgettext` calls are not influenced if the *domainname* parameter of these functions is not the null pointer.

Before the first call to `textdomain` the default domain is `messages`. This is the name specified in the specification of the `gettext` API. This name is as good as any other name. No program should ever really use a domain with this name since this can only lead to problems.

The function returns the value which is from now on taken as the default domain. If the system went out of memory the returned value is `NULL` and the global variable `errno` is set to `ENOMEM`. Despite the return value type being `char *` the return string must not be changed. It is allocated internally by the `textdomain` function.

If the *domainname* parameter is the null pointer no new default domain is set. Instead the currently selected default domain is returned.

If the *domainname* parameter is the empty string the default domain is reset to its initial value, the domain with the name `messages`. This possibility is questionable to use since the domain `messages` really never should be used.

char * `bindtextdomain` (const char **domainname*, const char **dirname*) Function

The `bindtextdomain` function can be used to specify the directory which contains the message catalogs for domain *domainname* for the different languages. To be correct, this is the directory where the hierarchy of directories is expected. Details are explained below.

For the programmer it is important to note that the translations which come with the program have to be placed in a directory hierarchy starting at, say, `‘/foo/bar’`. Then the program should make a `bindtextdomain` call to bind the domain for the current program to this directory. So it is made sure the catalogs are found. A correctly running program does not depend on the user setting an environment variable.

The `bindtextdomain` function can be used several times and if the *domainname* argument is different the previously bound domains will not be overwritten.

If the program which wish to use `bindtextdomain` at some point of time use the `chdir` function to change the current working directory it is important that the *dirname* strings ought to be an absolute pathname. Otherwise the addressed directory might vary with the time.

If the *dirname* parameter is the null pointer `bindtextdomain` returns the currently selected directory for the domain with the name *domainname*.

The `bindtextdomain` function returns a pointer to a string containing the name of the selected directory name. The string is allocated internally in the function and must not be changed by the user. If the system went out of core during the execution of `bindtextdomain` the return value is `NULL` and the global variable `errno` is set accordingly.

8.2.1.3 Additional functions for more complicated situations

The functions of the `gettext` family described so far (and all the `catgets` functions as well) have one problem in the real world which have been neglected completely in all existing approaches. What is meant here is the handling of plural forms.

Looking through Unix source code before the time anybody thought about internationalization (and, sadly, even afterwards) one can often find code similar to the following:

```
printf ("%d file%s deleted", n, n == 1 ? "" : "s");
```

After the first complaints from people internationalizing the code people either completely avoided formulations like this or used strings like `"file(s)"`. Both look unnatural and should be avoided. First tries to solve the problem correctly looked like this:

```
if (n == 1)
    printf ("%d file deleted", n);
else
    printf ("%d files deleted", n);
```

But this does not solve the problem. It helps languages where the plural form of a noun is not simply constructed by adding an 's' but that is all. Once again people fell into the trap of believing the rules their language is using are universal. But the handling of plural forms differs widely between the language families. There are two things we can differ between (and even inside language families);

- The form how plural forms are build differs. This is a problem with language which have many irregularities. German, for instance, is a drastic case. Though English and German are part of the same language family (Germanic), the almost regular forming of plural noun forms (appending an 's') is hardly found in German.
- The number of plural forms differ. This is somewhat surprising for those who only have experiences with Romanic and Germanic languages since here the number is the same (there are two).

But other language families have only one form or many forms. More information on this in an extra section.

The consequence of this is that application writers should not try to solve the problem in their code. This would be localization since it is only usable for certain, hardcoded language environments. Instead the extended `gettext` interface should be used.

These extra functions are taking instead of the one key string two strings and an numerical argument. The idea behind this is that using the numerical argument and the first string as a key, the implementation can select using rules specified by the translator the right plural form. The two string arguments then will be used to provide a return value in case no message catalog is found (similar to the normal `gettext` behavior). In this case the rules for Germanic language is used and it is assumed that the first string argument is the singular form, the second the plural form.