Page and line numbers refer to IEEE Std 1003.1-2017

On P55 after L1672 (after 3.137), add a new definition:

3.*w* Dot-Po File

See [xref to 3.*y* Portable Messages Object Source File].

On page 69 after line 2021 (after 3.226), add a new definition:

3.*x* Messages Object

A file containing message identifiers and translations in an unspecified format. Used by the *gettext* family of functions and the *gettext* and *ngettext* utilities for internationalization and localization of programs and scripts. Messages objects have the filename suffix **.mo**, and can be created by the *msgfmt* utility.

See also [xref to 3.*z* Text Domain].

On P79 after L2265 (after 3.282), add a new definition:

3.*y* Portable Messages Object Source File (or Dot-Po File)

A text file containing messages and directives. A portable messages object source file can be compiled into a messages object by the *msgfmt* utility.

Note: By convention, portable messages object source files have filenames ending with the **.po** suffix. Utility descriptions in this standard frequently use dot-po file as a shorthand for portable messages object source file (even though the **.po** suffix need not be included in the filename). Template portable messages object source files can be created from C-language source files by the *xgettext* utility.

On Page 98 after line 2728 (after 3.402), add a new definition:

3.*z* Text Domain

A named collection of messages objects (one messages object per supported language) for internationalization and localization purposes. A text domain is often named after the application or library that provides the collection, but may have a more general name if it is intended to be shared by multiple applications or libraries.

Note: The use of text domains is defined in detail in the descriptions of the *bindtextdomain*() and *gettext* family of functions in the System Interfaces volume of POSIX.1-202x.

[Editorial: After the above definitions have been added, update the definitions section numbers to again be sequential and unique.]

After page 135 line 3954 add:

3. Some functions, such as *catopen*() and those related to text domains, may reference various environment variables and a locale category of a specific locale to access files they need to

use.

In section 8.2, page 174, after line 5696, add:

LANGUAGE

The *LANGUAGE* environment variable shall be examined to determine the messages object to be used for the *gettext* family of functions or the *gettext* and *ngettext* utilities[XSI] if *NLSPATH* is not set or the evaluation of *NLSPATH* did not lead to a suitable messages object being found[/XSI]. The value of *LANGUAGE* shall be a list of locale names separated by a <colon> (':') character. If *LANGUAGE* is set to a non-empty string, each locale name shall be tried in the specified order and if a messages object is found, it shall be used for translation. If a locale name has the format *language***[**_*territory***][**.*codeset***][**@*modifier***]**, additional searches of locale names without .*codeset* (if present), without _*territory* (if present), and without @*modifier* (if present) may be performed; if .*codeset* is not present, additional searches of locale names with an added .*codeset* may be performed. If locale names contain a <slash> ('/') character, or consist entirely of a dot (".") or dot dot ("..") character sequence, or are empty the behavior is implementation defined and they may be ignored for security reasons.

The locale names in *LANGUAGE* shall override the locale name associated with the "active category" of the current locale or, in the case of functions with an *_l* suffix, the provided locale object, and the language-specific part of the default search path for messages objects, unless the locale name that would be overridden is C or POSIX. For the *dcgettext*(), *dcgettext_l*(), *dcngettext*(), and *dcngettext_l*() functions, the active category is specified by the *category* argument; for all other *gettext* family functions and for the *gettext* and *ngettext* utilities, the active category is *LC_MESSAGES*.

For example, if:

- The *LC_MESSAGES* environment variable is "de_DE" (and *LC_ALL* is unset) and setlocale(LC_ALL, "") has been used to set the current locale
- The *LANGUAGE* environment variable is "fr_FR:it"
- Messages objects are by default searched for in **/gettextlib** then the following pathnames are tried in this order by *gettext* family functions that have neither a *category* argument nor an *_l* suffix until a valid messages object is found:
- **/gettextlib/fr_FR/LC_MESSAGES/***textdomain***.mo**
- (Optionally) **/gettextlib/fr/LC_MESSAGES/***textdomain***.mo**
- (Optionally) the above two pathnames with added codeset elements
- **/gettextlib/it/LC_MESSAGES/***textdomain***.mo**
- (Optionally) the above pathname with added codeset elements
- **/gettextlib/de_DE/LC_MESSAGES/***textdomain***.mo**

In section 8.2, page 175, change lines 5733-5763 (NLSPATH) from:

This variable shall contain a sequence of templates that the *catopen*( ) function uses when attempting to locate message catalogs. Each template consists of an optional prefix, one or more conversion specifications, a pathname, and an optional suffix.

For example:

```
    NLSPATH="/system/nlslib/%N.cat"
```

defines that *catopen*( ) should look for all message catalogs in the directory **/system/nlslib**, where the catalog name should be constructed from the *name* parameter passed to *catopen*( ) (%N), with the suffix **.cat**.

Conversion specifications consist of a `'%'` symbol, followed by a single-letter keyword. The following keywords are currently defined:

`%N` The value of the *name* parameter passed to *catopen*( ).
`%L` The value of the *LC_MESSAGES* category.
`%l` The language element from the *LC_MESSAGES* category.
`%t` The territory element from the *LC_MESSAGES* category.
`%c` The codeset element from the *LC_MESSAGES* category.
`%%` A single `'%'` character.

An empty string is substituted if the specified value is not currently defined. The separators <underscore> (`'_'`) and <period> (`'.'`) are not included in the `%t` and `%c` conversion specifications.

Templates defined in *NLSPATH* are separated by <colon> characters (`':'`). A leading or two adjacent <colon> characters (`"::"`) is equivalent to specifying %N. For example:

```
NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

indicates to *catopen*( ) that it should look for the requested message catalog in *name*, *name***.cat**, and **/nlslib/***category***/***name***.cat**, where *category* is the value of the *LC_MESSAGES* category of the current locale.

Users should not set the *NLSPATH* variable unless they have a specific reason to override the default system path. Setting *NLSPATH* to override the default system path produces undefined results in the standard utilities and in applications with appropriate privileges.

to:

[XSI]This variable shall contain a sequence of templates to be used by *catopen*() when locating message catalogs, and by the *gettext* family of functions when locating messages objects. Each template consists of an optional prefix, one or more conversion specifications, and an optional suffix.

The conversion specification descriptions below refer to a "currently active text domain". The currently active text domain is, in decreasing order of precedence:

- the *domain* parameter of the *gettext* family of functions or the *gettext* and *ngettext* utilities
- the text domain bound by the last call to *textdomain()* when using a *gettext* family function, or the *TEXTDOMAIN* environment variable when using the *gettext* and *ngettext* utilities
- the default text domain

Conversion specifications consist of a `'%'` symbol, followed by a single-letter keyword. The following conversion specifications are currently defined:

%N The value of the *name* parameter passed to *catopen*( ) or the currently active text domain of the *gettext* family of functions and the *gettext* and *ngettext* utilities (see above).
%L The locale name given by the value of the active category (see *LANGUAGE* above) in either the current locale or, in the case of functions with an *_l* suffix, the provided locale object.
%l The language element of the locale name that would result from a %L conversion.
%t The territory element of the locale name that would result from a %L conversion.
%c The codeset element of the locale name that would result from a %L conversion.
%% A single `'%'` character.

An empty string shall be substituted if the specified value is not currently defined. The separators <underscore> (`'_'`) and <period> (`'.'`) shall not be included in the `%t` and `%c` conversion specifications.

Templates defined in *NLSPATH* are separated by <colon> characters (`':'`). A leading, trailing, or two adjacent <colon> characters (`"::"`) shall be equivalent to specifying %N.

Since <colon> is a separator in this context, directory names that might be used in NLSPATH should not include a <colon> character.

Example 1, for an application that uses *catopen*( ) but does not use the *gettext* family of functions:

NLSPATH="/system/nlslib/%N.cat"

indicates that *catopen*( ) should look for all message catalogs in the directory **/system/nlslib**, where the catalog name should be constructed from the *name* argument (replacing %N) passed to *catopen*( ), with the suffix **.cat**.

Example 2, for an application that uses the *gettext* family of functions but does not use *catopen*( ):

NLSPATH="/usr/lib/locale/fr/LC_MESSAGES/%N.mo"

indicates that the *gettext* family of functions (and the *gettext* and *ngettext* utilities) should look for all messages objects in the directory **/usr/lib/locale/fr/LC_MESSAGES**, where the messages object's name should be constructed from the currently active text domain (replacing %N), with the suffix **.mo**.

Example 3, for an application that uses *catopen*( ) but does not use the *gettext* family of functions:

```
NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

indicates that *catopen*( ) should look for the requested message catalog in *name*, *name***.cat**, and **/nlslib/***localename***/***name***.cat**, where *localename* is the locale name given by the value of the *LC_MESSAGES* category in the current locale.

Example 4, for an application that uses the *gettext* family of functions but does not use *catopen*( ):

NLSPATH="/usr/lib/locale/%L/%N.mo:/usr/lib/locale/fr/%N.mo"

indicates that the *gettext* family of functions (and the *gettext* and *ngettext* utilities) should look for all messages objects first in **/usr/lib/locale/***localename***/***textdomain*.mo, and if not found there, then try in **/usr/lib/locale/fr/***textdomain***.mo**, where *localename* is the locale name given by the value of the active category in the current locale or provided locale object.

Example 5, for an application that uses *catopen*( ) and the *gettext* family of functions:

NLSPATH="/usr/lib/locale/%L/%N.mo:/system/nlslib/%L/%N.cat"

indicates that the *gettext* family of functions (and the *gettext* and *ngettext* utilities) should look for all messages objects in **/usr/lib/locale/***localename***/***textdomain***.mo**, where *localename* is the locale name given by the value of the active category in the current locale or provided locale object. Also, *catopen*( ) should look for all message catalogs in the directory **/system/nlslib/***localename***/***name***.cat**, (assuming that **/usr/lib/locale/***localename***/***name***.mo** is not a message catalog). In this scenario, *catopen*() ignores all files that are not valid message catalogs while traversing *NLSPATH*. Furthermore, the *gettext* family of functions and the *gettext* and *ngettext* utilities ignore all files that are not valid messages objects found while traversing *NLSPATH*.

Users should not set the *NLSPATH* variable unless they have a specific reason to override the default system path. Setting *NLSPATH* to override the default system path may produce undefined results in the standard utilities other than *gettext* and *ngettext*, and in applications with appropriate privileges.

Specifying a relative pathname in the *NLSPATH* environment variable should be avoided without a specific reason, including the use of a leading or two adjacent <colon> characters, since it may result in messages objects being searched for in a directory relative to the current working directory of the calling process; if the process calls the *chdir*() function, the directory searched for may also be changed.[/XSI]

In section 8.2, page 176, add after line 5763:

TEXTDOMAIN
Specify the text domain name that the *gettext* and *ngettext* utilities use during the search for messages objects. This is identical to the messages object filename without the **.mo** suffix.

TEXTDOMAINDIR
Specify the pathname to the root directory of the messages object hierarchy the *gettext* and *ngettext* utilities use during the search for messages objects. If present, it shall replace the default root directory pathname. [XSI]*NLSPATH* has precedence over *TEXTDOMAINDIR*.[/XSI]

For catopen(), on P649 L22307 append to the paragraph:

When searching *NLSPATH*, *catopen*() shall ignore any files it finds that are not valid message catalog files.

For catopen(), change on P649, L22332:

[ENOENT] The message catalog does not exist or the *name* argument points to an empty string.

to:

[ENOENT] The *name* argument contains a '/' and does not name an existing message catalog, the *name* argument does not contain a '/' and searching [XSI]*NLSPATH* (if set) and then [/XSI]the implementation-defined default path for a message catalog with that name failed, one or more files exist but all are of an invalid format, or the *name* argument points to an empty string.

Add the following new header before <limits.h>:

NAME

libintl.h — international messaging

SYNOPSIS

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

DESCRIPTION

The **<libint.h>** header may define the macro TEXTDOMAINMAX. If defined, it shall have the same value as {TEXTDOMAIN_MAX} in **<limits.h>**.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
char *bindtextdomain(const char *, const char *);

char *bind_textdomain_codeset(const char *, const char *);

char *dcgettext(const char *, const char *, int);

char *dcgettext_l(const char *, const char *,
          int, locale_t);

char *dcngettext(const char *, const char *,
          const char *, unsigned long int, int);

char *dcngettext_l(const char *, const char *,
          const char *, unsigned long int, int, locale_t);

char *dgettext(const char *, const char *);

char *dgettext_l(const char *, const char *, locale_t);

char *dngettext(const char *, const char *,
          const char *, unsigned long int);

char *dngettext_l(const char *, const char *,
          const char *, unsigned long int, locale_t);

char *gettext(const char *);

char *gettext_l(const char *, locale_t);

char *ngettext(const char *, const char *,
          unsigned long int);

char *ngettext_l(const char *, const char *,
          unsigned long int, locale_t);

char *textdomain(const char *);
```

APPLICATION USAGE

None.

RATIONALE

Some historical implementations defined TEXTDOMAINMAX in this header. This standard instead defines {TEXTDOMAIN_MAX} in **<limits.h>**. This was done to allow the maximum length of a text domain name to vary depending on the filesystem type used to store message catalogs. Implementations are allowed to continue to define TEXTDOMAINMAX in this header as an extension to the standard (see XSH 2.2.2 on page NNN).

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *gettext*, *bindtextdomain*()

Add to <limits.h> on page 274 after line 9185:

{TEXTDOMAIN_MAX}
Maximum length of a text domain name, not including the terminating null byte.
Minimum Acceptable Value: {_POSIX_NAME_MAX} - 3
<XSI>Minimum Acceptable Value: {_XOPEN_NAME_MAX} - 3</XSI>

Add to <unistd.h>, page 444 after line 15233 (i.e. in alphabetic order)

_PC_TEXTDOMAIN_MAX

Add to the table on page 475, at line 16324 (between grp.h and limits.h):

<libintl.h>			TEXTDOMAINMAX

Add to fpathconf() table on page 902 line 30512:

{TEXTDOMAIN_MAX}	_PC_TEXTDOMAIN_MAX	3,4

Add the following sections where appropriate:

NAME

gettext, gettext_l, dgettext, dgettext_l, dcgettext, dcgettext_l, ngettext, ngettext_l, dngettext, dngettext_l, dcngettext, dcngettext_l — message handling functions

SYNOPSIS

```
#include <libintl.h>

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

char *dgettext_l(const char *domainname, const char *msgid,
          locale_t locale);

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

char *dcgettext_l(const char *domainname, const char *msgid,
          int category, locale_t locale);

char *dngettext(const char *domainname, const char *msgid,
          const char *msgid_plural, unsigned long int n);

char *dngettext_l(const char *domainname, const char *msgid,
          const char *msgid_plural, unsigned long int n, locale_t
locale);

char *dcngettext(const char *domainname, const char *msgid,
          const char *msgid_plural, unsigned long int n, int category);

char *dcngettext_l(const char *domainname, const char *msgid,
          const char *msgid_plural, unsigned long int n, int category,
          locale_t locale);

char *gettext(const char *msgid);

char *gettext_l(const char *msgid, locale_t locale);

char *ngettext(const char *msgid, const char *msgid_plural,
          unsigned long int n);

char *ngettext_l(const char *msgid, const char *msgid_plural,
          unsigned long int n, locale_t locale);
```

DESCRIPTION

The *gettext*() function shall:

- attempt to locate a suitable messages object (described in detail below) for the *LC_MESSAGES* category in the current locale, and for the current text domain (see [xref to *textdomain*()]), containing the string identified by *msgid*,
- retrieve the string identified by *msgid* from the messages object,
- convert the string to the output codeset if necessary (described in detail below), and
- return the result.

If the locale name in effect is "POSIX" or "C" (i.e. the name associated with the *LC_MESSAGES* locale category in the current locale), or if no suitable messages object exists, or if no string identified by *msgid* exists in the messages object, or if an error occurs, *msgid* shall be returned.

The *dgettext*() function shall be equivalent to *gettext*(), except *domainname* shall be used instead of the current text domain to locate the messages object.

The *dcgettext*() function shall be equivalent to *dgettext*(), except the locale category identified by *category* shall be used instead of *LC_MESSAGES*.

The *ngettext*() function shall be equivalent to *gettext*(), except:

- the string to retrieve shall be identified by a combination of *msgid* and *n* (see [xref to XCU *msgfmt*]), and
- if the locale name in effect is "POSIX" or "C", or if no suitable messages object exists, or if no string identified by the combination of *msgid* and *n* exists in the messages object, or if an error occurs, the return value shall be *msgid* if *n* is 1, otherwise *msgid_plural*.

The *dngettext*() function shall be equivalent to *ngettext*(), except *domainname* shall be used instead of the current text domain to locate the messages object.

The *dcngettext*() function shall be equivalent to *dngettext*(), except the locale category identified by *category* shall be used instead of *LC_MESSAGES*.

The *_*l*() functions shall be equivalent to their counterparts without the *_l* suffix, except *locale* shall be used instead of the current locale. If *locale* is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle, the behavior is undefined.

The *msgid* and *msgid_plural* arguments shall be strings. If either *msgid* or *msgid_plural* is an empty string, or contains characters not in the portable character set, the results are unspecified. If the argument *category* is *LC_ALL,* the results are unspecified.

The location of the messages object shall be determined according to the following criteria, stopping when the first messages object is found:

1. [XSI]If the *NLSPATH* environment variable is set to a non-empty string, an *NLSPATH* search shall be performed as described in [xref to XBD 8.2]. If *NLSPATH* identifies more than one template to use, each template in turn shall be used until a valid messages object is found.[/XSI]
2. If the *LANGUAGE* environment variable is set to a non-empty string, a *LANGUAGE* search shall be performed as described below. If *LANGUAGE* identifies more than one directory to search, each directory shall be searched until a valid messages object is found.
3. A single-locale search shall be performed as described below.

For [XSI]the *NLSPATH* search and[/XSI] the single-locale search, the single locale name used to locate the messages object shall be the locale name associated with the selected locale category from the current locale, or the provided locale object if calling one of the **_l*() functions; additional searches of locale names without *.codeset* (if present), without *_territory* (if present), and without *@modifier* (if present) may be performed.

For the *LANGUAGE* search, the value of the *LANGUAGE* environment variable shall be a list of one or more locale names separated by a colon (':') character. Each locale name shall be tried in the specified order. If a messages object for the locale does not exist, or cannot be opened, or is unsuitable for implementation-defined reasons (such as security), the next locale name (if any) shall be tried. If:

- a messages object for the locale can be opened but cannot be processed without error, or
- the messages object does not contain a string identified by *msgid*, or *msgid* and *n* for the *ngettext* functions,

it is unspecified whether the next locale name (if any) is tried. In all other cases, the messages object for the locale shall be used.

For each locale name in *LANGUAGE*, or if *LANGUAGE* is not set or is empty, or no suitable messages object is found in processing *LANGUAGE*, the pathname used to locate the messages object shall be *dirname*/*localename*/*categoryname*/*textdomainname***.mo**, where:

- The *dirname* part is the *dirname* argument of the most recent successful call to *bindtextdomain*() that had *textdomainname* as the *domainname* argument; any trailing <slash> characters in *dirname* shall be discarded. If a successful call to *bindtextdomain*() has not been made for *textdomainname*, an implementation-defined default directory shall be used.
- For the *LANGUAGE* search, the *localename* part is each locale name from *LANGUAGE* in turn; if a locale name has the format *language***[***_territory***][***.codeset***][***@modifier***]**, additional searches of locale names without *.codeset* (if present), without *_territory* (if present), and without *@modifier* (if present) may be performed; if *.codeset* is not present, additional searches of locale names with an added *.codeset* may be performed. For the single-locale search, the *localename* part is the name of the current locale, or the locale specified in an **_l*() function call, for the category named by *categoryname.* Spellings of *codeset* names are not standardized, and implementations may attempt to use different commonly known spellings, for example "utf8*"* and "UTF-8*".*
- The *categoryname* part is the string "LC_MESSAGES" if *gettext*(), *dgettext*(), *ngettext*(), or *dngettext*() is called, or the locale category name corresponding to the *category* argument to

dcgettext() or *dcngettext*(). Likewise for the *_*l*() variants of all these functions.

- For *gettext*(), *gettext_l*(), *ngettext*() and *ngettext_l*(), the *textdomainname* part is the text domain set by the last successful call to *textdomain*(). For *dgettext*(), *dcgettext*(), *dngettext*(), *dcngettext*(), and the *_*l*() variants of these functions, *textdomainname* is the text domain specified by the *domainname* argument. The *domainname* argument shall be equivalent in syntax and meaning to the *domainname* argument to *textdomain*(), except that the selection of the text domain shall affect only the *dgettext*(), *dcgettext*(), *dngettext*(), and *dcngettext*() function calls and their *_*l*() variants. If the *domainname* argument is a null pointer, the text domain set by the last successful call to *textdomain*() shall be used. For all of these functions, if a successful call to *textdomain*() has not been made the default text domain "messages" shall be used.

Resolution of the messages object pathname shall be performed the first time one of the *gettext* family of functions is called for a given combination of *dirname*, *localename*, *categoryname*, and *textdomainname*. It is unspecified whether the pathname is re-resolved if the combination has been used before in a call to one of the *gettext* family of functions. If *bindtextdomain*() performs pathname resolution of its *dirname* argument, only the part of the messages object pathname after *dirname* shall be resolved by the *gettext* family of functions.

When the *gettext* family of functions return a message string that was found in a messages object, they shall convert the codeset of the message string to the output codeset if a codeset is specified in the messages object (see [xref to XCU msgfmt]) and the output codeset is not the same as that codeset. If a successful call to *bind_textdomain_codeset*() has been made with the text domain of the messages object as the *domainname* argument and a non-null *codeset* argument, the output codeset shall be the *codeset* argument from the most recent such call. Otherwise, the output codeset shall be the codeset of characters in the current locale, or the provided locale object if calling one of the *_*l*() functions, as specified by the *LC_CTYPE* category of the locale. The conversion shall be performed as if by a call to *iconv*() using a conversion descriptor returned by *iconv_open*(<*output codeset*>, <*messages object codeset*>), except that if the return value of *iconv*() would be greater than zero, the non-identical conversions performed by the *gettext* family of functions need not be the same as those that such an *iconv*() call would perform. If an error prevents the codeset conversion from being performed, the *gettext* family of functions shall behave as if no message string was found in the messages object. If at least one non-identical conversion is performed that results in a fallback character (one that does not provide any information about the character it was converted from, for example, a <question-mark> or ``replacement-character"), the *gettext* family of functions may behave as if no message string was found in the messages object.

RETURN VALUE

The *gettext*(), *gettext_l*(), *dgettext*(), *dgettext_l*(), *dcgettext*(), and *dcgettext_l*() functions shall return the message string described in DESCRIPTION if successful. Otherwise, they shall return *msgid*.

The *ngettext*(), *ngettext_l*(), *dngettext*(), *dngettext_l*(), *dcngettext*(), and *dcngettext_l*() functions shall return the message string described in DESCRIPTION if successful. Otherwise, *msgid* shall be returned if *n* is equal to 1, or *msgid_plural* if *n* is not equal to 1.

The application shall ensure that it does not modify the returned string. A subsequent call to a *gettext* family function shall not overwrite or invalidate the returned string. The returned string may be invalidated by a subsequent call to *bind_textdomain_codeset*(), *bindtextdomain*(), *setlocale*(), or *textdomain*() in the same process, except for calls that only query values. The returned string shall not be invalidated by a subsequent call to *uselocale*().

ERRORS

The *gettext* family of functions shall not modify *errno*. If an error occurs these functions shall return a string as described in RETURN VALUE.

EXAMPLES

The example code below assumes the following:

- The implementation-defined default directory is **/system/gettextlib**
- The following locales are available on the target system: en_US, en_GB, de_DE. The codeset used for all of these locales is UTF-8.
- The en_AU locale is not available on the target system.
- The target system supports conversion from ISO/IEC 8859-1 to UTF-8.
- The codeset used for the POSIX locale is ASCII.
- The target system does not support conversion from ISO/IEC 8859-1 to ASCII.

Furthermore, the following **.mo** files (and only the following **.mo** files) are installed:

- **/system/gettextlib/en_US/LC_MESSAGES/mail.mo** and
- **/messagecatalogs/example/en_US/LC_MESSAGES/mail.mo**

These are compiled from a portable messages object source file (dot-po file) with the following ISO/IEC 8859-1 encoded contents (see the EXTENDED DESCRIPTION of the *msgfmt* utility for a description of the dot-po file format):

```
msgid ""
msgstr ""
"Content-Type: text/plain; charset=ISO_8859-1\n"
"Plural-Forms: nplurals=4; plural= n==1?0: (n>1 && n< 10)?1: (n==0)?
2:3;\n"

msgid "recipient"
msgid_plural "recipients"
msgstr[0] "1 recipient"
msgstr[1] "2 to 9 recipients"
```

```
msgstr[2] "no recipients"
msgstr[3] "more than 9 recipients"
```

/system/gettextlib/de_DE/LC_MESSAGES/mail.mo is compiled from a dot-po file with the following ISO/IEC 8859-1 encoded contents:

```
msgid ""
msgstr ""
"Content-Type: text/plain; charset=ISO_8859-1\n"
"Plural-Forms: nplurals=4; plural= n==1?0: (n>1 && n< 5)?1: (n==0)?
2:3;\n"

msgid "recipient"
msgid_plural "recipients"
msgstr[0] "1 Empfänger"
msgstr[1] "2 bis 4 Empfänger"
msgstr[2] "keine Empfänger"
msgstr[3] "mehr als 4 Empfänger"
```

/messagecatalogs/example/en_GB/LC_MESSAGES/mail.mo is compiled from a dot-po file with the following ISO/IEC 8859-1 encoded contents:

```
msgid ""
msgstr ""
"Content-Type: text/plain; charset=ISO_8859-1\n"
"Plural-Forms: nplurals=4; plural= n==1?0: (n>1 && n< 5)?1: (n==0)?
2:3;\n"

msgid "recipient"
msgid_plural "recipients"
msgstr[0] "1 recipient"
msgstr[1] "2 to 4 recipients"
msgstr[2] "no recipients"
msgstr[3] "5 or more recipients"
```

/messagecatalogs/example2/en_US/LC_MESSAGES/othermail.mo is not a suitable messages object file or is a suitable messages object file that does not contain the **msgid** "recipient"

The following example demonstrates the interactions between *bindtextdomain*(), *bind_textdomain_codeset*(), *textdomain*(), and the *gettext* family of functions.

```
unsigned long n_recipients;
// strdup() is used to prevent default_domain from being invalidated by a
future
// call to bindtextdomain()
```

```
const char *default_domain = strdup(bindtextdomain("mail", NULL));
setlocale(LC_MESSAGES, "POSIX");
setlocale(LC_CTYPE, "POSIX");

n_recipients = 1;
// The following outputs "recipient" with the same encoding as the
"recipient"
// argument to ngettext():
printf("%s\n", ngettext("recipient", "recipients", n_recipients));

n_recipients = 3;
// The following outputs "recipients" with the same encoding as the
"recipients"
// argument to ngettext():
printf("%s\n", ngettext("recipient", "recipients", n_recipients));

setlocale(LC_MESSAGES, "en_US");
setlocale(LC_CTYPE, "en_US");
textdomain("mail");

n_recipients = 1;
// The following outputs "1 recipient", encoded in UTF-8:
printf("%s\n", ngettext("recipient", "recipients", n_recipients));

n_recipients = 3;
// The following outputs "2 to 9 recipients", encoded in UTF-8:
printf("%s\n", ngettext("recipient", "recipients", n_recipients));

setlocale(LC_MESSAGES, "en_GB");
setlocale(LC_CTYPE, "en_GB");
bindtextdomain("mail", "/messagecatalogs/example/");

n_recipients = 3;
// The following outputs "2 to 4 recipients", encoded in UTF-8:
printf("%s\n", ngettext("recipient", "recipients", n_recipients));

setlocale(LC_MESSAGES, "en_US");
setlocale(LC_CTYPE, "en_US");
textdomain("othermail");
bindtextdomain("othermail", "/messagecatalogs/example2/");

n_recipients = 3;
```

```
// The following outputs "recipients" with the same encoding as the
"recipients"
// argument to ngettext():
printf("%s\n", ngettext("recipient", "recipients", n_recipients));

// Because there is no locale named en_AU on the system, en_US is used:
setenv("LANGUAGE", "en_AU:en_US:en_GB", 1);
setlocale(LC_MESSAGES, "");
setlocale(LC_CTYPE, "");
bindtextdomain("mail", default_domain);

// The following outputs "2 to 9 recipients", encoded in UTF-8:
printf("%s\n", dngettext("mail", "recipient", "recipients", 3));

textdomain("mail");
bind_textdomain_codeset("mail", "UTF-8");
setlocale(LC_MESSAGES, "de_DE");
setlocale(LC_CTYPE, "de_DE");
// Clear the LANGUAGE environment variable, otherwise it would take
precedence
// over the locale set above, and en_US would continue to be used.
setenv("LANGUAGE", "", 1);

n_recipients = 1;
// The following outputs "1 Empfänger", encoded in UTF-8:
printf("%s\n", ngettext("recipient", "recipients", n_recipients));

bind_textdomain_codeset("mail", "ASCII");
setlocale(LC_CTYPE, "POSIX");

n_recipients = 1;
// The following outputs "recipient" with the same encoding as the
"recipient"
// argument to ngettext() - remember, the system is assumed to not
support
// conversion from ISO/IEC 8859-1 to ASCII:
printf("%s\n", ngettext("recipient", "recipients", n_recipients));
free(default_domain);
```

APPLICATION USAGE

These functions do not impose a limit on message length. Note that translated strings typically have a different length than the input strings, possibly much longer, and applications using these translations in formatted text (e.g., aligned columns for a table) should take that into account.

The *dcgettext*(), *dcgettext_l*(), *dcngettext*(), and *dcngettext_l*() functions are useful to retrieve locale-specific strings for a category other than *LC_MESSAGES*. For example, they can be used to obtain a time format string from the *LC_TIME* category; because the locale setting of *LC_TIME* and *LC_MESSAGES* can be different, using the other *gettext* family functions in such a case might cause an undesired result. All of the functions in the *gettext* family of functions, except *dcgettext*(), *dcgettext_l*(), *dcngettext*(), and *dcngettext_l*(), search for messages objects only in the *LC_MESSAGES* category.

Implementations typically, but are not required to, *mmap*() the messages object file the first time one of the *gettext* family of functions is called, and keep that map in place until it is no longer expected to be used. For example, a successful call to *bindtextdomain*() will typically cause the next call to one of the *gettext* family of functions to *munmap*() the previous file and *mmap*() the new file. Applications should not rely on this behavior, however: the implementation is allowed to cache previously used maps, or not use *mmap*() at all and re-open the file each time one of the *gettext* family of functions is called.

The *msgid* and *msgid_plural* arguments are typically in (US) English. The arguments are always used in the POSIX or C locale, and when a *gettext* family-function encounters an error, so they should not be abstract message identifiers (e.g., "message 123") and they should only use characters in the portable character set (to avoid outputting byte sequences that are not valid characters in the current output codeset). If the *xgettext* utility is used to extract the *msgid* and *msgid_plural* arguments from C source files into a template dot-po file, the arguments must be string literals in order for the resulting file to be useful to translators.

The strings returned by the *gettext* family of functions are not guaranteed to contain only characters that are valid in the current output codeset. In particular, byte sequences that do not form valid characters can occur when:

- The *msgid* or *msgid_plural* arguments use characters outside the portable character set.
- The messages object file does not specify a character set and uses characters outside the portable character set.

The strings returned by the *gettext* family of functions are guaranteed to remain valid until invalidated as described in the RETURN VALUE section. This includes strings that are created by codeset conversion; those strings are freed by the implementation, not the application. Thus, it is safe to call *gettext* family functions multiple times in situations such as:

```
printf("%s %s\n", gettext("foo"), gettext("bar"));
```

RATIONALE

Although the return type of these functions ought to be **const char ***, it is **char *** to match historical practice.

The *gettext* family of functions is frequently used in reporting errors. In fact, it is possible to have an application that attempts to create an error message that combines a translated string via *gettext*() with an error string provided by *strerror*(). The standard requires that the *gettext* family of functions cannot modify *errno*, so that an application need not worry about complications of providing sequencing points to capture a stable value of *errno* prior to the translation of the error message, and so that the user will still get a somewhat useful string (even if it is the untranslated original string) on any failure.

There are no wide character equivalents for these functions; historically no implementation is known to exist, and the multi-byte message returned from these functions can, in most instances, be converted to wide characters by the application if desired.

Some historical *gettext*() implementations returned the translated string from the messages object without codeset conversion if *iconv_open*() fails. This is considered to be a bug in those implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

bindtextdomain(), *catopen*(), *iconv*(), *setlocale*(), *uselocale*()

XBD **<libintl.h>**, **<limits.h>**

XCU *gettext*, *msgfmt*, *xgettext*

NAME

bindtextdomain, bind_textdomain_codeset, textdomain — text domain manipulation functions

SYNOPSIS

```
#include <libintl.h>

char *bindtextdomain(const char *domainname, const char *dirname);

char *bind_textdomain_codeset(const char *domainname,
          const char *codeset);

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

DESCRIPTION

The *textdomain*() function shall set or query the name of the current text domain of the calling process. The application shall ensure that the *domainname* argument is either a null pointer (when querying), an empty string, or a string that, when used by the *gettext* family of functions to construct a pathname to a messages object, results in a valid pathname. For portable applications, it should only contain characters from the portable filename character set.

The text domain setting made by the last successful call to *textdomain*() shall remain in effect across subsequent calls to *setlocale*(), *uselocale*(), and the *gettext* family of functions.

Applications should not use text domains whose names begin with the strings "SYS_" or "libc". These prefixes are reserved for implementation use.

The current setting of the text domain can be queried without affecting the current state of the domain by calling *textdomain*() with *domainname* set to a null pointer. Calling *textdomain*() with a *domainname* argument of an empty string shall set the text domain to the default domain, "messages".

The *bindtextdomain*() function shall set or query the binding of a text domain to a *dirname* that is used by the *gettext* family of functions to construct a pathname to a messages object in the text domain:

- If *domainname* is a null pointer or an empty string, *bindtextdomain*() shall make no changes and return a null pointer without changing *errno*.
- Otherwise, if *dirname* is a non-empty string:
  - If *domainname* is not already bound, *bindtextdomain*() shall bind the text domain specified by *domainname* to the pathname pointed to by *dirname* and return the bound directory pathname on success or a null pointer on failure.
  - If *domainname* is already bound, *bindtextdomain*() shall replace the existing binding with the pathname pointed to by *dirname* and return the bound directory pathname on success or a null pointer on failure. On failure, the existing binding shall remain unchanged.

  It is unspecified whether the *bindtextdomain*() function performs pathname resolution on *dirname*, or whether that is done by the *gettext* family of functions.

- Otherwise, if *dirname* is a null pointer:
  - If *domainname* is bound, the function shall return the bound directory pathname.
  - If *domainname* is not bound, the function shall return the implementation-defined default directory pathname used by the *gettext* family of functions.

- Otherwise, *dirname* is an empty string and the behavior is unspecified.

If a text domain is bound to a relative pathname and the current working directory is changed after the binding is established, the pathnames used by the *gettext* family of functions to locate messages objects for that text domain are unspecified.

The *bind_textdomain_codeset*() function shall set or query the binding of a text domain to the output codeset used by the *gettext* family of functions for message strings retrieved from messages objects for the text domain specified by *domainname*:

- If *domainname* is a null pointer or an empty string, *bind_textdomain_codeset*() shall make no changes and return a null pointer without changing *errno*.
- Otherwise, if *codeset* is a non-empty string:
  - If *domainname* is not already bound, *bind_textdomain_codeset*() shall bind the text domain specified by *domainname* to the codeset pointed to by *codeset* and return the newly bound codeset on success or a null pointer on failure.
  - If *domainname* is already bound, *bind_textdomain_codeset*() shall replace the existing binding with the codeset pointed to by *codeset* and return the newly bound codeset on success or a null pointer on failure. On failure, the existing binding shall remain unchanged.

  The application shall ensure that the *codeset* argument, if non-empty, is a valid codeset name that can be used as the *tocode* argument of the *iconv_open*() function, and that in the codeset it specifies, the <NUL> character corresponds to a single null byte.
- Otherwise, if *codeset* is a null pointer:
  - If *domainname* is bound, the function shall return the bound codeset.
  - If *domainname* is not bound, the function shall return the implementation-defined default codeset used by the *gettext* family of functions.
- Otherwise, *codeset* is an empty string and the behavior is unspecified.

If *codeset* is a null pointer and *domainname* is a non-empty string, *bind_textdomain_codeset*() shall return the current codeset for the named domain, or a null pointer if a codeset has not yet been set. The *bind_textdomain_codeset*() function can be called multiple times. If successfully called multiple times with the same *domainname* argument, the last such call shall override the setting made by the previous such call.

RETURN VALUE

The return value from a successful *textdomain*() call shall be a pointer to a string containing the current setting of the text domain. If *domainname* is a null pointer, *textdomain*() shall return a pointer to the string containing the current text domain. If *textdomain*() was not previously called and *domainname* is a null string, the name of the default text domain shall be returned. The name of the default text domain shall be the string "messages". If *textdomain*() fails, a null pointer shall be returned and *errno* shall be set to indicate the error.

For *bindtextdomain*() return values see the DESCRIPTION. When *bindtextdomain*() is called with a non-empty *domainname* and returns a null pointer, it shall set *errno* to indicate the error. When *bindtextdomain*() returns a pathname for a bound text domain, the return value shall be a pointer to a copy of the *dirname* string passed to the *bindtextdomain*() call that created the binding. The returned string shall remain valid until the next successful call to *bindtextdomain*() with a non-empty *dirname* and same *domainname*. The application shall ensure that it does not modify the returned string.

A call to the *bind_textdomain_codeset*() function with a non-empty *domainname* argument shall return one of:

- the currently bound codeset name for that text domain if *codeset* is a null pointer,
- the newly bound codeset if *codeset* is non-empty,
- a null pointer without changing *errno* if no codeset has yet been bound for that text domain.

The application shall ensure that it does not modify the returned string. A subsequent call to *bind_textdomain_codeset*() with a non-empty *domainname* argument might invalidate the returned pointer or overwrite the string content. The returned pointer might also be invalidated if the calling thread is terminated. If *bind_textdomain_codeset*() fails, a null pointer shall be returned and *errno* shall be set to indicate the error.

ERRORS

For the conditions under which *bindtextdomain*()—if it performs pathname resolution—fails and may fail, refer to [xref to open()].

In addition, the *textdomain*(), *bindtextdomain*(), and *bind_textdomain_codeset*() functions may fail if:

[ENOMEM]
: Insufficient memory available.

EXAMPLES

See the examples for [xref to gettext].

APPLICATION USAGE

A text *domainname* is limited to {TEXTDOMAIN_MAX} bytes.

Application developers are responsible for ensuring that the text domain used is not used by other applications. To minimize the chances of collision, developers can prefix text domains with their company or application name (or both) and an underscore. For example, if your application name was "foo" and you wanted to use the text domain "errors", you could instead use the text domain "foo_errors". Note: If an application can be installed with a configurable name, a text domain prefix based on the application name should change with the application name.

Specifying a relative pathname to the *bindtextdomain*() function should be avoided, since it may result in messages objects being searched for in a directory relative to the current working directory of the calling process; if the process calls the *chdir*() function, the directory searched for may also be changed.

Since pathname resolution of *dirname* might not be performed by *bindtextdomain*(), but could be performed later by the *gettext* family of functions, and since the latter have no way to report an error, applications should verify, using for example *stat*(), that the directory is accessible if this is desired.

RATIONALE

Although the return type of of these functions ought to be **const char** *, it is **char** * to match historical practice.

Pathname resolution of the *dirname* argument passed to *bindtextdomain*() may be performed by *bindtextdomain*() itself or by the *gettext* family of functions. If pathname resolution fails in one of the *gettext* family of functions, it is neither allowed to modify *errno* nor to return an error, but if pathname resolution fails in *bindtextdomain*(), it is required to report an error and set *errno* just like *open*() does.

Historically, *bindtextdomain*() did not perform pathname resolution. However, the standard developers decided to allow this as an option so that future implementations can, if desired, open a file descriptor for that directory in *bindtextdomain*() and then use that file descriptor with *openat*() in the *gettext* family of functions.

The *dirname* parameter to *bindtextdomain*() may need to be copied to avoid the possibility of the application releasing the memory used by the argument while the *gettext* family of functions may still need to reference it.

When *bindtextdomain*() is called with a non-empty *domainname* and an empty *dirname*, historical implementations of the *gettext* family of functions use the empty string for the *dirname* part of the messages object pathname, resulting in an absolute pathname of the form /*localename*/*categoryname*/*textdomainname*.**mo**. The standard developers did not believe this behavior to be useful. Using the empty *dirname* case as a way to remove an existing binding seemed to be a more useful behavior, and would be consistent with the behavior of *textdomain*(). However, because no historical implementations behave this way, the behavior is left unspecified.

Some implementations set errno to [EAGAIN] to signal memory allocation failures that might succeed if retried and [ENOMEM] for failures that are unlikely to ever succeed, for example due to configured limits. Section 2.3 (on page xxx) permits this behavior; when multiple error conditions are simultaneously true there is no precedence between them.

FUTURE DIRECTIONS

A future version of this standard may require implementations to prefix implementation-provided text domains with either SYS_ or a prefix related to the implementor's company name to avoid namespace collisions.

A future version of this standard may require *bindtextdomain*() to remove any binding for *domainname* when called with a non-empty *domainname* and an empty *dirname*.

SEE ALSO

gettext(), *iconv_open*(), *setlocale*(), *uselocale*()

XBD **<libintl.h>**, **<limits.h>**

XCU *msgfmt*, *xgettext*

Create pointer pages for:

- dcgettext, dcgettext_l, dcngettext, dcngettext_l
- dgettext, dgettext_l
- dngettext, dngettext_l
- ngettext, ngettext_l
- textdomain

NAME

gettext, ngettext — retrieve text string from messages object

SYNOPSIS

```
gettext [-e|-E] [-d textdomain] [textdomain] msgid

gettext [-e|-E] [-n] -s [-d textdomain] msgid...

ngettext [-e|-E] [-d textdomain] [textdomain] msgid msgid_plural n
```

DESCRIPTION

The *gettext* and *ngettext* utilities shall write to standard output the message string(s) that would result from the following calls to functions defined in the System Interfaces volume of POSIX.1-202x:

```
if (textdomainname == NULL || textdomainname[0] == '\0')
    message_string = msgid;
else {
    setlocale(LC_ALL, "");
    if (textdomaindir != NULL)
        bindtextdomain(textdomainname, textdomaindir);
    if (msgid_plural == NULL)
        message_string = dgettext(textdomainname, msgid);
    else
       message_string = dngettext(textdomainname, msgid, msgid_plural,
       n);
}
```

where:

- The *textdomaindir* variable is a string containing the value of the *TEXTDOMAINDIR* environment variable, if set and not empty, or is NULL otherwise.

- The *textdomainname* variable is a string containing either the text domain name obtained from, in decreasing order of precedence:
  - the optional operand *textdomain*, if present
  - the **-d** *textdomain* option, if specified
  - the *TEXTDOMAIN* environment variable, if set and not empty

  If the text domain name cannot be obtained from these sources, the *textdomainname* variable is NULL.

- If the **-s** option of *gettext* is not specified and for the *ngettext* utility, the *msgid* variable is a string containing:
  - The value of the *msgid* operand, if the **-E** option is specified
  - The value of the *msgid* operand with C-language escape sequences processed (see below), if the **-e** option is specified
  - The value of the *msgid* operand with C-language escape sequences optionally processed (see below), otherwise.

- If the **-s** option of *gettext* is specified, the *msgid* variable is a string containing:
  - The value of each *msgid* operand in turn, if the **-E** option is specified or neither the **-e** nor the **-E** option is specified
  - The value of each *msgid* operand in turn with C-language escape sequences processed (see below), if the **-e** option is specified.

- For the *gettext* utility, the *msgid_plural* variable is NULL. For the *ngettext* utility, the *msgid_plural* variable is a string containing:

        - The value of the *msgid_plural* operand, if the **-E** option is specified
        - The value of the *msgid_plural* operand with C-language escape sequences processed (see below), if the **-e** option is specified
        - The value of the *msgid_plural* operand with C-language escape sequences optionally processed (see below), otherwise.

    - For the *gettext* utility, *n* is one. For the *ngettext* utility the *n* variable is the *n* operand, parsed as an integer as if by using the *strtoul*() function with a *base* argument of 10.

When C-language escape sequences are processed, they shall be processed as specified for character string literals in the ISO C standard, except that *universal-character-name* escape sequences need not be supported. Implementations may also support a <backslash> 'c' escape sequence; if supported, the '\c' and all characters following it shall be removed and, if the **-s** option is specified, the behavior shall be as if the **-n** option is also specified.

For the *ngettext* utility, and for the *gettext* utility if the **-s** option is not specified, the resulting message string shall be written to standard output. If the **-s** option of *gettext* is specified, the resulting message string for each *msgid* shall be written to standard output with consecutive message strings separated by a single <space> character and, if the **-n** option is not specified, a <newline> shall be written after the last message string. If the -**s** and -**n** options are specified, the trailing <newline> shall be omitted.

Under conditions where the *textdomainname* variable in the above code would be NULL, these utilities may write a diagnostic message to standard error and exit with non-zero status.

OPTIONS

These utilities shall conform to XBD Section 12.2 (on page XXX).

The following options shall be supported:

-d *textdomain*
: Retrieve the translated message from the domain *textdomain*, if *textdomain* is not specified as an operand.

-e
: Process C-language escape sequences in *msgid* and *msgid_plural* operands.

-E
: Do not process C-language escape sequences in *msgid* and *msgid_plural* operands.

The *gettext* utility shall also support the following options:

-**n**

Modify the behavior of the **-s** option such that a <newline> is not appended to the output.

-s

Separate the message strings obtained from each *msgid* operand with <space> characters in the output, and (if **-n** is not also specified) append a <newline> to the output.

If neither of the mutually exclusive **-e** and **-E** options is specified, it is unspecified which is the default, except that if the **-s** option of *gettext* is specified then **-E** shall be the default.

OPERANDS

The following operands shall be supported:

textdomain

A text domain name used to retrieve the translated message. This shall override the specification by the **-d** option, if present.

msgid

A key to retrieve the translated message.

msgid_plural

A default plural if no corresponding plural message can be found.

n

A non-negative decimal integer to be used as the *n* argument to *dngettext*() (see the DESCRIPTION).

STDIN

Not used.

INPUT FILES

The input files are messages object files (see [xref to *msgfmt*]).

ENVIRONMENT VARIABLES

LANG

Provide a default value for the internationalization variables that are unset or null. (See [xref to XBD Section 8.2] for the precedence of internationalization variables used to determine the values of locale categories.)

LANGUAGE

Determine the location of messages objects [XSI]if *NLSPATH* is not set or the evaluation of *NLSPATH* did not lead to a suitable messages object being found[/XSI].

LC_ALL

If set to a non-empty string value, override the values of all the other internationalization variables.

LC_MESSAGES

Determine the locale name used to locate messages objects, and the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

[XSI]*NLSPATH*

Determine the location of messages objects and message catalogs.[/XSI]

TEXTDOMAIN

Specify the text domain name. (See [xref to XBD 3.403].)

TEXTDOMAINDIR

Specify the pathname to the messages object hierarchy.[XSI] *NLSPATH* shall have precedence over *TEXTDOMAINDIR*.[/XSI]

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the DESCRIPTION.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

 0 Successful completion.
>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Since it is unspecified which of the **-e** or **-E** options is the default, except when the **-s** option of *gettext* is specified, portable applications need to ensure that **-e**, **-E**, or (for *gettext*) **-s** is specified whenever a *msgid* or *msgid_plural* operand contains, or might contain, a <backslash> character.

Note that, unless the **-s** option of *gettext* is specified without **-n**, the message(s) written to standard output are not followed by a <newline>. (Therefore the output only ends with a <newline> if the last message ends with one.)

Both *msgid* and *msgid_plural* should be properly quoted for the shell.

EXAMPLES

The following examples assume that the following portable messages object source file (dot-po file) has been compiled to a valid file **mail.mo** by the *msgfmt* utility. See the EXTENDED DESCRIPTION section of the *msgfmt* utility for a description of the dot-po file format.

```
msgid ""
msgstr ""
"Content-Type: text/plain; charset=utf-8\n"
"Plural-Forms: nplurals=4; plural= n==1?0: (n>1 && n<= 10)?1: (n==0)?
2:3;\n"

msgid "recipient"
msgid_plural "recipients"
msgstr[0] "1 recipient"
msgstr[1] "2 to 10 recipients"
msgstr[2] "no recipients"
msgstr[3] "more than 10 recipients"

msgid "%d attachment\n"
msgid_plural "%d attachments\n"
```

```
msgstr[0] "1 (%d) attachment\n"
msgstr[1] "2 to 10 (%d) attachments\n"
msgstr[2] "no (%d) attachments\n"
msgstr[3] "more than 10 (%d) attachments\n"
```

They also assume that **mail.mo** is installed in the directory that *gettext* and *ngettext* search for the current locale. See the options and environmental variables above and the description of *gettext*() for details on how this search is performed.

The command
```
ngettext -d mail recipient recipients 0
```
will write "no recipients".

The command
```
ngettext -d mail recipient recipients 1
```
will write "1 recipient".

The command
```
ngettext -d mail recipient recipients 5
```
will write "2 to 10 recipients ".

The command
```
ngettext -d mail recipient recipients 11
```
will write "more than 10 recipients".

The command
```
ngettext -d mail Call Calls 1
```
will write "Call". Note that "Call" is not in the messages object.

The command
```
ngettext -d mail Call Calls 0
```
will write "Calls".

The command
```
ngettext -d mail Call Calls 10
```
will write "Calls".

The command
```
ngettext -e -d mail "%d attachment\n" "%d attachments\n" 1
```
will write the same as
```
printf "1 (%%d) attachment\n"
```
(i.e. "1 (%d) attachment" followed by a <newline> character). The output of *ngettext* can be used as a format string for *printf*.

The command

```
printf "$(ngettext -e -d mail "%d attachment\n" "%d attachments\n" 1)" 10
```

will write the same as

```
printf "1 (%d) attachment\n" 10
```

(i.e. "1 (10) attachment" followed by a <newline> character).

The command

```
ngettext -e -d mail "\tsubject\n" "\tsubjects\n" 0
```

will write the same as

```
printf "\tsubjects\n"
```

(i.e. a <tab> character, followed by "subjects" followed by a <newline> character). Note that "\tsubject\n" is not in the messages object.

The command

```
printf "%s\n" "$(ngettext -E -d mail "subject" "subjects" 0)"
```

will write the same as

```
printf "subjects\n"
```

(i.e. "subjects" followed by a <newline> character). Note that "subject" is not in the messages object.

The command

```
gettext -s -d mail "recipient"
```

will write "1 recipient" followed by a <newline> character.

The command

```
gettext -s -n -d mail "recipient"
```

will write "1 recipient" without a <newline> character.

RATIONALE

Historical implementations did not support the `"\a"` C-language escape sequence. This standard requires it to be supported for consistency with other utilities that support the table in XBD Chapter 5 (on page NNN).

Unlike other standard utilities, the behavior of *gettext* and *ngettext* is not undefined when *NLSPATH* overrides the system default path or *TEXTDOMAINDIR* overrides the default root directory; see XBD Section 8.2 (on page NNN). This is so that applications can use these utilities to obtain message strings from messages objects in other locations. However, it also means that they need to be implemented in such a way that they do not do anything that would result in undefined behavior when they need to write a diagnostic message. In particular, they should not use a string obtained from a message catalog or a messages object as a format string (or should only do so after checking that the string contains the correct conversions).

FUTURE DIRECTIONS

None.

SEE ALSO

msgfmt, *printf*

XBD Chapter 7 (on page NNN), Chapter 8 (on page NNN)

XSH *gettext*, *iconv*(), *setlocale*()

NAME

msgfmt — create messages objects from portable messages object source files

SYNOPSIS

```
msgfmt [-cfSv] [-D dir] [-o outputfile] pathname...
```

DESCRIPTION

The *msgfmt* utility shall create messages object files from portable messages object source files (dot-po files).

A dot-po file contains messages to be output by system commands or by applications. The messages in these files should be able to be translated to any language supported by the system.

The *msgfmt* utility shall interpret message strings for output as characters according to the codeset specified in the dot-po file or, if not present, the current setting of the *LC_CTYPE* locale category.

OPTIONS

The *msgfmt* utility shall conform to XBD Section 12.2 (on page XXX).

The following options shall be supported:

-c

If this option and **-v** are both specified, *msgfmt* shall detect and diagnose input file abnormalities which might represent translation errors. The **msgid** and **msgstr** strings shall be compared. It shall be considered abnormal if one string starts or ends with a <newline> while the other does not. Also, if the flag **c-format** appears in a "#," comment for a **msgid** directive (see EXTENDED DESCRIPTION), it shall be considered abnormal if the strings do not have the same number of '%' conversion specifiers, or if corresponding conversion specifiers take different argument types (see [xref to fprintf()]). If an abnormality is detected, the exit status shall be non-zero and a diagnostic message shall be output. Additional checks beyond those described here may also be performed. These checks may produce diagnostics or informational messages and need not affect the exit status. If **-c** is specified without **-v** or **-v** is specified without **-c**, the behavior is unspecified.

-D *dir*

Add *dir* to the list of directories to search for input files.

-f

Use fuzzy entries in output. If this option is not specified, fuzzy entries shall not be included in the output.

-o *outputfile*

Specify the name of an output file to be used instead of the default filename(s) specified in EXTENDED DESCRIPTION. All **domain** *domainname* directives in the dot-po file(s) shall be ignored.

-S

Append the suffix **.mo** to each generated messages object filename if it does not have this suffix.

-v

See **-c**.

OPERANDS

The following operand shall be supported:

pathname   A pathname of a dot-po file.

STDIN

The standard input shall not be used.

INPUT FILES

The input files shall be text files in the format described in EXTENDED DESCRIPTION.

ENVIRONMENT VARIABLES

LANG
: Provide a default value for the internationalization variables that are unset or null. (See [xref to XBD Section 8.2] for the precedence of internationalization variables used to determine the values of locale categories.)

LANGUAGE
: Determine the location of messages objects [XSI]if *NLSPATH* is not set or the evaluation of *NLSPATH* did not lead to a suitable messages object being found[/XSI].

LC_ALL
: If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE
: Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
: Determine the locale name used to locate messages objects, and the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

[XSI]*NLSPATH*
: Determine the location of messages objects and message catalogs.[/XSI]

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall not be used.

STDERR

The standard error shall be used for diagnostic messages and may also be used for warning messages. If the **-c** and **-v** options are specified, additional unspecified informational messages may be written to standard error.

OUTPUT FILES

The format of the created messages object files is unspecified.

EXTENDED DESCRIPTION

The *msgfmt* utility shall accept portable messages object source files (dot-po files) in the following format.

A dot-po file contains zero or more lines, with each non-blank line containing a comment, a statement, or a statement continuation. A comment has an unquoted <number-sign> ('#') as the first non-<blank> character and ends with the next <newline> character. A statement continuation is a double-quoted string on a line by itself, optionally preceded and/or followed by <blank> characters, and the string shall be concatenated with the string on the previous statement line. If a comment occurs between a statement and a statement continuation, the behavior is unspecified. All other comments, except for comments beginning with <number-sign><comma> ("#,"), and blank lines shall be ignored.

The format of a statement is:
directive *value*

The *directive* starts at the first non-<blank> character of the line and is separated from the *value* by one or more <blank> characters. The *value* consists of a double-quoted string optionally followed by <blank> characters. Zero or more statement continuation lines (see above) can follow the statement. The following directives shall be supported:

domain *domainname*
msgid *message_identifier*
msgid_plural *untranslated_string_plural*
msgstr *message_string*
msgstr[*index***]** *message_string*

A dot-po file consists of zero or more sections. Each section specifies the messages to be processed in a domain. The first directive in each section shall be a **domain** directive (except for the first section which shall behave as if

```
domain "messages"
```

had been specified if the first directive is not a **domain** directive).

The behavior of the **domain** directive is affected by the options used. See OPTIONS for the behavior when the **-o** option is specified. If the **-o** option is not specified, all data obtained from the non-domain directives in a dot-po section shall be output to the messages object file named *domainname***.mo** when the **-S** option is specified. When the **-S** option is not specified, it is implementation-defined whether *domainname* or *domainname***.mo** is used.

If multiple **domain** directives specify the same *domainname*, the sections shall be processed as if there was only one section that starts with a **domain** "*domainname*" statement which contained the statements of the sections, in the same order, excluding all but the first **domain** "*domainname*" statement.

Within each section, there can be a header. A header is identified by having a **msgid** directive with the empty string ("") as the *message_identifier* immediately followed by a statement containing a **msgstr** directive. The *message_string* in this **msgstr** statement in a header shall be treated specially. If *message_string* contains a specification of the form:

  "nplurals=*count***; plural=***expression"*

count indicates the number of plural forms for messages in that domain, and *expression* is a C language expression that evaluates to an unsigned integer value which determines the **msgstr[***index***]** directive to be used. The value of *expression* is used as the *index* value. The variable *n* in *expression* is assigned the value of the *n* argument to the *ngettext*(), *ngettext_l*(), *dngettext*(), *dngettext_l*(), *dcngettext*()**,** and *dcngettext_l*() functions or of the *n* operand of the *ngettext* utility before *expression* is evaluated. The application shall ensure that *expression* evaluates to a non-negative value less than *count* for all *n* that can be supplied by the aforementioned functions and utilities.

If *message_string* in the header contains a specification of the form:

  "charset=*codeset"*

codeset indicates the codeset to be used to encode the message strings in this section's domain (overriding *LC_CTYPE*). If the output string's codeset is different from the message string's codeset, codeset conversion from the message string's codeset to the output string's codeset shall be performed by the *gettext* family of functions and by the *gettext* and *ngettext* utilities. See [xref to XSH gettext] and [xref to gettext]. The output string's codeset shall be determined by the current or specified locale's codeset.
<small>**Note:** it is the responsibility of translators to ensure that the characters they enter into message strings in a dot-po file are encoded in the codeset specified in the header.</small>

If a header is present in a section, the application shall ensure that the header is provided by the first **msgid** directive in that section.

After the header, if present, zero or more messages are identified by a **msgid** directive with a *message_identifier* that is not an empty string.   Each of these directives start a subsection that is used to get a translated message from the *gettext* family of functions and from the *gettext* and *ngettext* utilities.   If the *message_identifier* string is the string identified by the *gettext* family of functions *msgid* argument or by the *gettext* and *ngettext* utility *msgid* operand, this subsection specifies how that translation is to be processed.

If there is only a singular form for the given *message_identifier*, the application shall ensure that the statement containing the **msgid** directive is immediately followed by a **msgstr** directive.

If there are plural forms for the given *message_identifier* and the header for this section exists and contains an **"nplurals=***count***; plural=***expression"* specification, the application shall ensure that the statement containing the **msgid** directive is immediately followed by a **msgid_plural** directive and that each statement containing a **msgid_plural** directive is followed by *count* statements containing **msgstr[***index***]** directives, starting with **msgstr[0]** and ending with **msgstr[***count***-1]** in monotonically increasing order. If a header for this section does not exist or does not contain an "**nplurals**=*count*; **plural**=*expression"* specification, the application shall ensure that no **msgid_plural** or **msgstr[***index***]** directives are used in this section.

For example, if the header*'s message_string* contains the specification:

```
"nplurals=2; plural= n == 1 ? 0 : 1"
```

there are two forms in the domain; **msgstr[**0**]** is used if *n* is equal to 1, otherwise **msgstr[**1**]** is used. For another example, if the header's *message_string* contains:

```
"nplurals=3; plural= n == 1 ? 0 : n == 2 ? 1 : 2"
```

there are three forms in the domain; **msgstr[**0**]** is used if *n* is equal to 1, **msgstr[**1**]** is used if *n* is equal to 2, otherwise **msgstr[**2**]** is used.

C-language escape sequences in strings shall be processed as specified for character string literals in the ISO C standard, except that *universal-character-name* escape sequences need not be supported.

Comments in a dot-po file can be in one of the following formats:

#: *reference*
#. *utility-added-comments*
#, *flag*
#*translator-comments*   (where *translator-comments* does not begin with '.', ':' or ',')

A **"#:"** *reference* comment indicates the location(s) of the **msgid** string in the source files, in *pathname1*:*linenumber1* **[***pathname2:linenumber2 ...***]** format. They can be added, as might "**#.**" prefixed additional comments of unspecified format, by the *xgettext* utility. All comments that do not begin with "#," are informative only and shall be silently ignored by the *msgfmt* utility. In **"#,"** comments the following *flag*s can be specified:

fuzzy

This flag indicates that the **msgstr** string might not be a correct translation at this point in time. Only the translator can judge if the translation requires further modification or is acceptable as is. Once satisfied with the translation, the translator should remove this **fuzzy** flag. If this flag is specified, the *msgfmt* utility shall not generate the entry for the next following **msgid** in the output message catalog, unless the **-f** option is specified. If other flag comments are specified between **fuzzy** and the **msgid**, the behavior is unspecified.

c-format
no-c-format

The **c-format** flag indicates that the next following **msgid** string contains a *printf*() format string. When the **c-format** flag is given and the **-c** and **-v** options are specified, the *msgfmt* utility shall perform additional tests to check the validity of the translation (see OPTIONS); these additional tests may also be performed if neither **c-format** nor **no-c-format** is given. When the **no-c-format** flag is given for a string, no additional checks shall be performed for the string. When both the **c-format** and the **no-c-format** flags are given, the last flag specified takes precedence.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.
>0 An error occurred.

CONSEQUENCES OF ERRORS

The *msgfmt* utility need not continue processing later *pathname* operands when an error condition that affects the exit status is detected. It is unspecified whether a messages object file is written when checks performed for the **-c** and **-v** options fail.

APPLICATION USAGE

The *xgettext* utility can be used to create template dot-po files from C-language source files.

Installing messages object files for the POSIX or C locale is not recommended, since they may be ignored for the sake of efficiency.

The first section for each domain in a dot-po file should include a header containing a **charset**=*codeset* specification. If this specification is omitted, message conversions in the *gettext* family of functions and in the *gettext* and *ngettext* utilities may fail.

The **msgid_plural** directive's *untranslated_string_plural* string comes from the *msgid_plural* arguments in calls to the *ngettext*(), *ngettext_l*(), *dngettext*(), *dngettext_l*(), *dcngettext*()**,** and *dcngettext_l*() functions when a prototype dot-po file is created by the *xgettext* utility. These strings (and the *msgid_plural* operands in calls to the *ngettext* utility) can provide context when a translator is modifying a template dot-po file into a dot-po file for a specific language. These functions and the *ngettext* utility do not try to match the *msgid_plural* arguments or operands with anything in a messages object file; they only match the *msgid* arguments and operands.

Unlike shell command language strings, double-quoted strings in dot-po files cannot contain a literal <newline> character.

EXAMPLES

In this example, **module1.po** and **module2.po** are portable messages object source files.

```
$ cat module1.po
# default domain "messages"
msgid ""
msgstr "charset=utf-8"
msgid  "msg 1"
msgstr "msg 1 translation"
#
domain "help_domain"
msgid ""
msgstr "charset=utf-8"
msgid  "help 2"
msgstr "help 2 translation"
#
domain "error_domain"
msgid ""
msgstr "charset=utf-8"
msgid  "error 3"
msgstr "error 3 translation"

$ cat module2.po
# default domain "messages"
msgid ""
msgstr "charset=utf-8"
msgid  "mesg 4"
msgstr "mesg 4 translation"
#
domain "error_domain"
msgid ""
```

```
msgstr "charset=utf-8"
#, c-format
msgid  "error 5 %s"
msgstr "error 5 translation %s"
#
domain "window_domain"
msgid ""
msgstr "charset=utf-8"
msgid  "window 6"
msgstr "window 6 translation"

$ cat module3.po
# default domain "messages"
# header will be used for the whole output file in the third example
msgid ""
msgstr "charset=utf-8"
msgid  "info 0"
msgstr "info 0 translation"

$ cat opt_debug.po
#
domain "debug_domain"
msgid  "debug 8"
msgstr "debug 8 translation"
```

The following command will produce the output files **messages.mo**, **help_domain.mo**, and **error_domain.mo**:

```
$ msgfmt -S module1.po
```

The following command will produce the output files **messages.mo**, **help_domain.mo**, **error_domain.mo**, and **window_domain.mo**:

```
$ msgfmt -S module1.po module2.po
```

The following command will produce the output file **hello.mo**:

```
$ msgfmt -o hello.mo module3.po opt_debug.po
```

RATIONALE

Some implementations are less strict about the format of dot-po files and simply treat all occurrences of one or more white space characters as a separator. The format described in this standard is accepted by all known implementations.

In some implementations, duplicate **msgid** directives within a domain are ignored, and only an entry for the first **msgid** directive and the following **msgid**, **msgid_plural**, **msgstr** or **msgstr[***index***]** directives is created. However, some implementations consider duplicate **msgid** directives within a domain to be an error and do not produce output at all. Consequently this standard does not specify the behavior of *msgfmt* if duplicate **msgid** directives are encountered within one domain.

FUTURE DIRECTIONS

None.

SEE ALSO

gettext, *xgettext*

XSH *fprintf*(), *gettext*

Create a pointer page for ngettext

NAME

xgettext — extract *gettext* call strings from C-language source files (DEVELOPMENT)

SYNOPSIS

```
[CD] xgettext [-j] [-n] [-d default-domain] [-K keyword-spec]... [-p
pathname] file...

    xgettext -a [-n] [-d default-domain] [-p pathname] [-x exclude-file]
  file... [/CD]
```

DESCRIPTION

The *xgettext* utility shall automate the creation of portable messages object source files (dot-po files). A dot-po file shall contain copies of string literals that are found in C-language source code in files specified by *file* operands. The dot-po file can be used as input to the *msgfmt* utility, to produce a messages object file that can be used by applications.

The *xgettext* utility shall write *msgid* argument strings that are passed as string literals in *gettext*(), *gettext_l*(), *ngettext*(), and *ngettext_l*() calls in C-language source code to the default output file; this file is named **messages.po** unless it is changed by the **-d** option. The *xgettext* utility shall also write *msgid* argument strings that are passed as string literals in *dcgettext*(), *dcgettext_l*(), *dcngettext*(), *dcngettext_l*(), *dgettext*(), *dgettext_l*(), *dngettext*(), and *dngettext_l*() calls either to the default output file or to the output file *domainname***.po** where *domainname* is the first parameter to the call; it is implementation-defined which of those output files is used. A **msgid** directive shall precede each *msgid* argument string. For the functions that have a *msgid_plural* argument, a **msgid_plural** directive followed by that argument string shall also be written directly after the corresponding **msgid** directive. A **msgstr** directive or **msgstr[***index***]** directives with an empty string shall be written after the corresponding **msgid** or **msgid_plural** directive, respectively. The function names that *xgettext* searches for can be changed using the **-K** option.

The first directive in each created dot-po file shall be a **domain** directive giving the associated domain name, except that this directive is optional in the default output file.

If the **-p** *pathname* option is specified, *xgettext* shall create the dot-po files in the *pathname* directory. Otherwise, the dot-po files shall be created in the current working directory.

The **msgid** values shall be in the same order that the strings are extracted from each *file* and subsections with duplicate **msgid** values shall be written to the dot-po files as comment lines.

OPTIONS

The *xgettext* utility shall conform to XBD Section 12.2 (on page XXX).

The following options shall be supported:

-a

Extract all strings, not just those found in calls to *gettext* family functions. Only one dot-po file shall be created.

-d *default-domain*

Name the default output file *default-domain***.po** instead of **messages.po**.

-j

Join messages from C-language source files with existing dot-po files. For each dot-po file that *xgettext* writes messages to, if the file does not exist, it shall be created. New messages shall be appended but any subsections with duplicate *msgid*s except the first (including **msgid** values found in an existing dot-po file) shall either be commented out or omitted in the resulting dot-po file; if omitted, a warning message may be written to standard error. Domain directives in the existing dot-po files shall be ignored; the assumption is that all previous *msgid*s belong to the same domain. The behavior is unspecified if an existing dot-po file was not created by *xgettext* or has been modified by another application.

-K *keyword-spec*
Specify an additional keyword to be looked for:

- If *keyword-spec* is an empty string, this shall disable the use of default keywords for the *gettext* family of functions.
- If *keyword-spec* is a C identifier, *xgettext* shall look for strings in the first argument of each call to the function or macro *keyword-spec*.
- If *keyword-spec* is of the form *id*:*argnum*, *xgettext* shall treat the *argnum*-th argument of a call to the function or macro *id* as the *msgid* argument, where *argnum* 1 is the first argument.
- If *keyword-spec* is of the form *id*:*argnum1*,*argnum2*, *xgettext* shall treat strings in the *argnum1*-th argument and in the *argnum2*-th argument of a call to the function or macro *id* as the *msgid* and *msgid_plural* arguments respectively.

For all mentioned forms, the application shall ensure that if *argnum2* is given, it is not equal to *argnum1*. All numeric values shall be converted as specified in [xref to XBD 12.1] item 6.

-n
Add comment lines to the output file indicating pathnames and line numbers in the source files where each extracted string is encountered. These lines shall appear before each **msgid** directive. Such comments should have the format **#:** *pathname1*:*linenumber1* **[***pathname2:linenumber2 ...***]**.

-p *pathname*
Create output files in the directory specified by *pathname* instead of in the current working directory.

-x *exclude-file*
Specify a file containing strings that shall not be extracted from the input files. The format of *exclude-file* is identical to that of a dot-po file. However, only statements containing **msgid** directives in *exclude-file* shall be used. All other statements shall be ignored.

OPERANDS

The following operand shall be supported:

file

A pathname of an input file containing C-language source code. If '-' is specified for an instance of *file*, the standard input shall be used.

STDIN

The standard input shall not be used unless a *file* operand is specified as '-'.

INPUT FILES

The input files specified as *file* operands shall be C-language source files. The input file specified as the option-argument for the **-x** option shall be a dot-po file in the format specified as input for the *msgfmt* utility.

ENVIRONMENT VARIABLES

LANG

Provide a default value for the internationalization variables that are unset or null. (See [xref to XBD Section 8.2] for the precedence of internationalization variables used to determine the values of locale categories.)

LANGUAGE

Determine the location of messages objects [XSI]if *NLSPATH* is not set or the evaluation of *NLSPATH* did not lead to a suitable messages object being found[/XSI]**.**

LC_ALL

If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES

Determine the locale name used to locate messages objects, and the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

[XSI]*NLSPATH*

Determine the location of messages objects and message catalogs.[/XSI]

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall not be used.

STDERR

The standard error shall be used for diagnostic messages and may be used for warning messages.

OUTPUT FILES

The output files shall be dot-po files in the format specified as input for the *msgfmt* utility. It is unspecified whether each output file includes a header (**msgid** "") before the content derived from the input C-language source files.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0	Successful completion.
>0	An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Implementations differ as to whether they write all output to the default output file or split the output into separate per-domain files. Portable applications can either ensure that each C-language source file contains calls to *gettext* family functions for only a single domain, or force all output to be to the default output file by using the **-K** option to override the default keywords.

Some implementations of *xgettext* are not able to extract cast strings (unless -a is used), for example casts of literal strings to **(const char *)**. Use of a cast is unnecessary anyway, since the prototypes in **<libintl.h>** already specify this type.

The *xgettext* utility is not required to handle C preprocessor directives. Therefore if, for example, calls to *gettext* family functions are wrapped by macros, they might not be found unless the **-K** option is used to tell *xgettext* to look for the macro calls.

EXAMPLES

Example 1
The following example shows how **-K** can be used to force all output to be to the default output file:

```
xgettext -K "" -K gettext:1 -K dgettext:2 -K dcgettext:2 -K ngettext:1,2 -K dngettext:2,3 -K
dcngettext:2,3 source.c
```

By overriding the default keywords using the **-K** option as above, the *xgettext* utility is directed to ignore the *domainname* arguments to the *dgettext*(), *dcgettext*(), *dngettext*() and *dcngettext*() functions. Thus, the utility treats the functions as their respective equivalent without the *d* prefix, ignoring the *domainname* argument and writing generated output to the default output file, **messages.po**. Additional **-K** options would be needed for the variants of the functions with an *_l* suffix if they are used.

Example 2
If the source uses a macro definition such as:

```
#define i18n gettext
```

the use of:

```
xgettext -K i18n:1 source.c
```

will pick up **msgid** values from a line such as:

```
fprintf(stdout, i18n("The value is %s"), value1);
```

RATIONALE

The **-K** option is based on the **-k** option of GNU *xgettext*; the only difference is that GNU's **-k** takes an optional option-argument whereas **-K** in this standard has a mandatory option-argument in order to comply with the syntax guidelines.

The standard developers considered including functionality equivalent to the **-c, -m,** and **-M** options in existing implementations. However, those letters could not be used as the syntax differed between implementations. The usual solution of adding an uppercase equivalent of lowercase options with the standard syntax instead was not possible, for obvious reasons for **-m** and **-M**, and as **-C** was already in use for another purpose in one implementation.

The **-s** option is not included as it has been deprecated in at least one implementation because it has been found to deprive translators of valuable context.

FUTURE DIRECTIONS

A future version of this standard may change the description of the **-n** option to use "shall" instead of "should".

SEE ALSO

gettext, msgfmt

XSH *gettext*

Global change for all utilities: NLSPATH text should not require that diagnostic messages come from message catalogues, but should allow messages objects as well. New text should be consistent with the new XCU pages above, as in change them to match:

[XSI]*NLSPATH*

Determine the location of messages objects and message catalogs.[/XSI]