The Open Group Standard

Additional APIs for the Base Specifications Issue 8, Part 2

The Open Group Standard

Additional APIs for the Base Specifications Issue 8, Part 2

ISBN: 1-957866-06-2

Document Number: C228

Published by The Open Group, December 2022

Comments relating to the material contained in this document may be submitted to:

The Open Group, Apex Plaza, Forbury Road, Reading, Berkshire, RG1 1AX, United Kingdom

or by electronic mail to:

ogspecs@opengroup.org

Contents

Preface

The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through technology standards. Our diverse membership of more than 870 organizations includes customers, systems and solutions suppliers, tools vendors, integrators, academics, and consultants across multiple industries.

The mission of The Open Group is to drive the creation of Boundaryless Information Flow™ achieved by:

- Working with customers to capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Working with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies
- Offering a comprehensive set of services to enhance the operational efficiency of consortia
- Developing and operating the industry's premier certification service and encouraging procurement of certified products

Further information on The Open Group is available at www.opengroup.org.

The Open Group publishes a wide range of technical documentation, most of which is focused on development of Standards and Guides, but which also includes white papers, technical studies, certification and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/library.

This Document

This document has been prepared by The Open Group Base Working Group. The Open Group Base Working Group is considering submitting a number of additional APIs to the Austin Group as input to the Issue 8 revision of the Base Specifications.

This document contains the second set of these APIs.

Trademarks

Acknowledgements

The Open Group gratefully acknowledges the contribution of the following in the development of this document:

- The Open Group Base Working Group
- The Austin Group

The Open Group gratefully acknowledges the following reviewers who participated in the Company Review of this document:

- Eric Ackermann
- Eric Blake
- Geoff Clare
- Donald W. Cragun
- Andrew Josey
- Nick Stoughton
- Mark Ziegast

1 Introduction

1.1 Scope

The purpose of this document is to define a set of additional APIs for inclusion in the Issue 8 revision of the Base Specifications of the Single UNIX® Specification.

The additional APIs proposed by participants in the Austin Group that The Open Group has agreed to sponsor are as follows:

Header

<libintl.h>

Functions

bind_textdomain_codeset()	*getresgid*()
bindtextdomain()	*getresuid*()
dcgettext()	*gettext*()
dcgettext_l()	*gettext_l*()
dcngettext()	*ngettext*()
dcngettext_l()	*ngettext_l*()
dgettext()	*setresgid*()
dgettext_l()	*setresuid*()
dngettext()	*textdomain*()
dngettext_l()	

Utilities

gettext	*realpath*
msgfmt	*timeout*
ngettext	*xgettext*
readlink	

1.2 Relationship to Other Formal Standards

This Standard is being forwarded to the Austin Group for consideration as input to the Issue 8 revision of the Base Specifications.

2 Application Program Interfaces

The following pages are extracted from a complete draft of the Base Specifications in which the proposed changes have been applied, with change bars showing the differences from Issue 8 Draft 2.1. Only pages with technical changes are included – editorial changes such as additions to SEE ALSO and CHANGE HISTORY sections have been omitted (unless they appear on the same page as a technical change). The complete draft is also being made available for reference.

As a consequence of the change to NLSPATH in XBD Section 8.2, a change will also need to be made to the NLSPATH description on all existing utility reference pages. These changes are not included here but will be made during the preparation of Issue 8 Draft 3.

2.1 Change Bars

Changed lines are marked with a '|' in the right-hand margin, new lines with a '+', and deleted lines with a '-'.

Note that sometimes the placement of change bars is slightly inaccurate. In particular, changes may extend into a line following a set of change-barred lines. Also, changes within tables do not have change bars.

2.2 Reference Pages

The reference pages for the new header, function, and utility additions, and pages with related changes follow.

3.106 Dot

In the context of naming files, the filename consisting of a single <period> character (`'.'`).

Note: In the context of shell special built-in utilities, see *dot* in XCU Section 2.14 (on page 2382).

Pathname Resolution is defined in detail in Section 4.14 (on page 93).

3.107 Dot-Dot

The filename consisting solely of two <period> characters (`".."`).

Note: Pathname Resolution is defined in detail in Section 4.14 (on page 93).

3.108 Dot-Po File +

See *Portable Messages Object Source File* in Section 3.257 (on page 65).

3.109 Double-Quote Character

The character `'"'`, also known as <quotation-mark>.

Note: The ``double'' adjective in this term refers to the two strokes in the character glyph. POSIX.1-202x never uses the term ``double-quote'' to refer to two apostrophes or quotation-marks.

3.110 Downshifting

The conversion of an uppercase character that has a single-character lowercase representation into this lowercase representation.

3.111 Driver

A module that controls data transferred to and received from devices.

Note: Drivers are traditionally written to be a part of the system implementation, although they are frequently written separately from the writing of the implementation. A driver may contain processor-specific code, and therefore be non-portable.

3.112 Effective Group ID

An attribute of a process that is used in determining various permissions, including file access permissions; see also Section 3.161 (on page 51).

3.113 Effective User ID

An attribute of a process that is used in determining various permissions, including file access permissions; see also Section 3.394 (on page 84).

3.197 Message Catalog

In the context of providing natural language messages to the user, a file or storage area containing program messages, command prompts, and responses to prompts for a particular native language, territory, and codeset.

3.198 Message Catalog Descriptor

In the context of providing natural language messages to the user, a per-process unique value used to identify an open message catalog. A message catalog descriptor may be implemented using a file descriptor.

3.199 Message Queue

In the context of programmatic message passing, an object to which messages can be added and removed. Messages may be removed in the order in which they were added or in priority order.

3.200 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 Section 3.374 (on page 81).

3.201 Mode

A collection of attributes that specifies a file's type and its access permissions.

Note: File Access Permissions are defined in detail in Section 4.6 (on page 90).

3.202 Monotonic Clock

A clock measuring real time, whose value cannot be set via *clock_settime*( ) and which cannot have negative clock jumps.

3.203 Mount Point

Either the system root directory or a directory for which the *st_dev* field of structure **stat** differs from that of its parent directory.

Note: The **stat** structure is defined in detail in **<sys/stat.h>**.

3.204 Multi-Character Collating Element

A sequence of two or more characters that collate as an entity. For example, in some coded

3.252 Pipe

An object identical to a FIFO which has no links in the file hierarchy.

Note: The *pipe*( ) function is defined in detail in the System Interfaces volume of POSIX.1-202x.

3.253 Polling

A scheduling scheme whereby the local process periodically checks until the pre-specified events (for example, read, write) have occurred.

3.254 Portable Character Set

The collection of characters that are required to be present in all locales supported by conforming systems.

Note: The Portable Character Set is defined in detail in Section 6.1 (on page 105).

This term is contrasted against the smaller portable filename character set; see also Section 3.256.

3.255 Portable Filename

A filename consisting only of characters from the portable filename character set.

Note: Applications should avoid using filenames that have the <hyphen-minus> character as the first character since this may cause problems when filenames are passed as command line arguments.

3.256 Portable Filename Character Set

The set of characters from which portable filenames are constructed.

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -
```

The last three characters are the <period>, <underscore>, and <hyphen-minus> characters, respectively. See also Section 3.245 (on page 63).

3.257 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. +

3.368 System Process

An object other than a process executing an application, that is provided by the system and has a process ID.

3.369 System Reboot

See *System Boot* defined in Section 3.362 (on page 80).

3.370 System-Wide

Pertaining to events occurring in all processes existing in an implementation at a given point in time.

3.371 Tab Character (<tab>)

A character that in the output stream indicates that printing or displaying should start at the next horizontal tabulation position on the current line. It is the character designated by `'\t'` in the C language. If the current position is at or past the last defined horizontal tabulation position, the behavior is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the tabulation.

3.372 Terminal (or Terminal Device)

A character special file that obeys the specifications of the general terminal interface.

Note: The General Terminal Interface is defined in detail in Chapter 11 (on page 185).

3.373 Text Column

A roughly rectangular block of characters capable of being laid out side-by-side next to other text columns on an output page or terminal screen. The widths of text columns are measured in column positions.

3.374 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.

3.375 Text File

A file that contains characters organized into zero or more lines. The lines do not contain NUL

Chapter 7

Locale

7.1 General

A locale is the definition of the subset of a user's environment that depends on language and cultural conventions. It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names:

LC_CTYPE	Character classification and case conversion.
LC_COLLATE	Collation order.
LC_MONETARY	Monetary formatting.
LC_NUMERIC	Numeric, non-monetary formatting.
LC_TIME	Date and time formats.
LC_MESSAGES	Formats of informative and diagnostic messages and interactive responses.

The standard utilities in the Shell and Utilities volume of POSIX.1-202x shall base their behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each utility. The behavior of some of the C-language functions defined in the System Interfaces volume of POSIX.1-202x shall also be modified based on a locale selection. The locale to be used by these functions can be selected in the following ways:

1. For functions such as *isalnum_l*( ) that take a locale object as an argument, a locale object can be obtained from *newlocale*( ) or *duplocale*( ) and passed to the function.

2. For functions that do not take a locale object as an argument, the current locale for the thread can be set by calling *uselocale*( ) or the global locale for the process can be set by calling *setlocale*( ). Such functions shall use the current locale of the calling thread if one has been set for that thread; otherwise, they shall use the global locale.

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. +
+

Locales other than those supplied by the implementation can be created via the *localedef* utility, provided that the _POSIX2_LOCALEDEF symbol is defined on the system. Even if *localedef* is not provided, all implementations conforming to the System Interfaces volume of POSIX.1-202x shall provide one or more locales that behave as described in this chapter. The input to the utility is described in Section 7.3 (on page 116). The value that is used to specify a locale when using environment variables shall be the string specified as the *name* operand to the *localedef* utility when the locale was created. The strings `"C"` and `"POSIX"` are reserved as identifiers for the POSIX locale (see Section 7.2, on page 116). When the value of a locale environment variable begins with a <slash> (`'/'`), it shall be interpreted as the pathname of the locale definition; the type of file (regular, directory, and so on) used to store the locale definition is implementation-defined. If the value does not begin with a <slash>, the mechanism used to locate the locale is implementation-defined.

8.2 Internationalization Variables

This section describes environment variables that are relevant to the operation of internationalized interfaces described in POSIX.1-202x.

Users may use the following environment variables to announce specific localization requirements to applications. Applications can retrieve this information using the *setlocale*() function to initialize the correct behavior of the internationalized interfaces. The descriptions of the internationalization environment variables describe the resulting behavior only when the application locale is initialized in this way. The use of the internationalization variables by utilities described in the Shell and Utilities volume of POSIX.1-202x is described in the ENVIRONMENT VARIABLES section for those utilities in addition to the global effects described in this section.

LANG This variable shall determine the locale category for native language, local customs, and coded character set in the absence of the *LC_ALL* and other *LC_** (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) environment variables. This can be used by applications to determine the language to use for error messages and instructions, collating sequences, date formats, and so on.

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. 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** +

LC_ALL This variable shall determine the values for all locale categories. The value of the *LC_ALL* environment variable has precedence over any of the other environment variables starting with *LC_* (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) and the *LANG* environment variable.

LC_COLLATE
This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the *strcoll*( ) and *strxfrm*( ) functions. Additional semantics of this variable, if any, are implementation-defined.

LC_CTYPE This variable shall determine the locale category for character handling functions, such as *tolower*( ), *toupper*( ), and *isalpha*( ). This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.

LC_MESSAGES
This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. It also affects the behavior of the *catopen*( ) function in determining the message catalog. Additional semantics of this variable, if any, are implementation-defined. The language and cultural conventions of diagnostic and informative messages whose format is unspecified by POSIX.1-202x should be affected by the setting of *LC_MESSAGES*.

LC_MONETARY
This variable shall determine the locale category for monetary-related numeric formatting information. Additional semantics of this variable, if any, are implementation-defined.

LC_NUMERIC
This variable shall determine the locale category for numeric formatting (for example, thousands separator and radix character) information in various utilities as well as the formatted I/O operations in *printf*( ) and *scanf*( ) and the string conversion functions in *strtod*( ). Additional semantics of this variable, if any, are implementation-defined.

LC_TIME This variable shall determine the locale category for date and time formatting information. It affects the behavior of the time functions in *strftime*( ). Additional semantics of this variable, if any, are implementation-defined.

XSI *NLSPATH* This variable shall contain a sequence of templates to be used by *catopen*( ) when attempting to locate 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 of 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, trailing, 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.

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

XSI shall replace the default root directory pathname. *NLSPATH* has precedence over *TEXTDOMAINDIR*.

The environment variables *LANG*, *LC_ALL*, *LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*, and *NLSPATH* provide for the support of internationalized applications. The standard utilities shall make use of these environment variables as described in this section and the individual ENVIRONMENT VARIABLES sections for the utilities. If these variables specify locale categories that are not based upon the same underlying codeset, the results are unspecified.

The values of locale categories shall be determined by a precedence order; the first condition met below determines the value:

1. If the *LC_ALL* environment variable is defined and is not null, the value of *LC_ALL* shall be used.

2. If the *LC_** environment variable (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) is defined and is not null, the value of the environment variable shall be used to initialize the category that corresponds to the environment variable.

3. If the *LANG* environment variable is defined and is not null, the value of the *LANG* environment variable shall be used.

4. If the *LANG* environment variable is not set or is set to the empty string, the implementation-defined default locale shall be used.

If the locale value is `"C"` or `"POSIX"`, the POSIX locale shall be used and the standard utilities behave in accordance with the rules in Section 7.2 (on page 116) for the associated category.

If the locale value begins with a <slash>, it shall be interpreted as the pathname of a file that was created in the output format used by the *localedef* utility; see OUTPUT FILES under *localedef*. Referencing such a pathname shall result in that locale being used for the indicated category.

XSI If the locale value has the form:

```
language[_territory][.codeset]
```

it refers to an implementation-provided locale, where settings of language, territory, and codeset are implementation-defined.

LC_COLLATE, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*, and *LC_TIME* are defined to accept an additional field *@modifier*, which allows the user to select a specific instance of localization data within a single category (for example, for selecting the dictionary as opposed to the character ordering of data). The syntax for these environment variables is thus defined as:

```
[language[_territory][.codeset][@modifier]]
```

For example, if a user wanted to interact with the system in French, but required to sort German text files, *LANG* and *LC_COLLATE* could be defined as:

```
LANG=Fr_FR
LC_COLLATE=De_DE
```

This could be extended to select dictionary collation (say) by use of the *@modifier* field; for example:

```
LC_COLLATE=De_DE@dict
```

An implementation may support other formats.

NAME

libintl.h — international messaging

SYNOPSIS

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

DESCRIPTION

The **<libintl.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 Section 2.2.2, on page 467).

FUTURE DIRECTIONS

None.

SEE ALSO

XSH *gettext*, *bindtextdomain*()

CHANGE HISTORY

First released in Issue 8.

{PIPE_BUF}
Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
Minimum Acceptable Value: {_POSIX_PIPE_BUF}

ADV {POSIX_ALLOC_SIZE_MIN}
Minimum number of bytes of storage actually allocated for any portion of a file.
Minimum Acceptable Value: Not specified.

ADV {POSIX_REC_INCR_XFER_SIZE}
Recommended increment for file transfer sizes between the {POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
Minimum Acceptable Value: Not specified.

ADV {POSIX_REC_MAX_XFER_SIZE}
Maximum recommended file transfer size.
Minimum Acceptable Value: Not specified.

ADV {POSIX_REC_MIN_XFER_SIZE}
Minimum recommended file transfer size.
Minimum Acceptable Value: Not specified.

ADV {POSIX_REC_XFER_ALIGN}
Recommended file transfer buffer alignment.
Minimum Acceptable Value: Not specified.

{SYMLINK_MAX}
Maximum number of bytes in a symbolic link.
Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

{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

Runtime Increasable Values

The magnitude limitations in the following list shall be fixed by specific implementations. An application should assume that the value of the symbolic constant defined by **<limits.h>** in a specific implementation is the minimum that pertains whenever the application is run under that implementation. A specific instance of a specific implementation may increase the value relative to that supplied by **<limits.h>** for that implementation. The actual value supported by a specific instance shall be provided by the *sysconf*( ) function.

{BC_BASE_MAX}
Maximum *obase* values allowed by the *bc* utility.
Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}

{BC_DIM_MAX}
Maximum number of elements permitted in an array by the *bc* utility.
Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}

{BC_SCALE_MAX}
Maximum *scale* value allowed by the *bc* utility.
Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}

{BC_STRING_MAX}
Maximum length of a string constant accepted by the *bc* utility.
Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}

_PC_PIPE_BUF
_PC_PRIO_IO
_PC_REC_INCR_XFER_SIZE
_PC_REC_MAX_XFER_SIZE
_PC_REC_MIN_XFER_SIZE
_PC_REC_XFER_ALIGN
_PC_SYMLINK_MAX
_PC_SYNC_IO
_PC_TEXTDOMAIN_MAX +
_PC_TIMESTAMP_RESOLUTION
_PC_VDISABLE

The **<unistd.h>** header shall define the following symbolic constants for *sysconf*( ):

_SC_2_C_BIND
_SC_2_C_DEV
_SC_2_CHAR_TERM
_SC_2_FORT_RUN
_SC_2_LOCALEDEF
_SC_2_SW_DEV
_SC_2_UPE
_SC_2_VERSION
_SC_ADVISORY_INFO
_SC_AIO_LISTIO_MAX
_SC_AIO_MAX
_SC_AIO_PRIO_DELTA_MAX
_SC_ARG_MAX
_SC_ASYNCHRONOUS_IO
_SC_ATEXIT_MAX
_SC_BARRIERS
_SC_BC_BASE_MAX
_SC_BC_DIM_MAX
_SC_BC_SCALE_MAX
_SC_BC_STRING_MAX
_SC_CHILD_MAX
_SC_CLK_TCK
_SC_CLOCK_SELECTION
_SC_COLL_WEIGHTS_MAX
_SC_CPUTIME
_SC_DELAYTIMER_MAX
_SC_EXPR_NEST_MAX
_SC_FSYNC
_SC_GETGR_R_SIZE_MAX
_SC_GETPW_R_SIZE_MAX
_SC_HOST_NAME_MAX
_SC_IOV_MAX
_SC_IPV6
_SC_JOB_CONTROL
_SC_LINE_MAX
_SC_LOGIN_NAME_MAX
_SC_MAPPED_FILES
_SC_MEMLOCK
_SC_MEMLOCK_RANGE

```
pid_t       getpid(void);
pid_t       getppid(void);
int         getresgid(gid_t *, gid_t *, gid_t *);
int         getresuid(uid_t *, uid_t *, uid_t *);
pid_t       getsid(pid_t);
uid_t       getuid(void);
int         isatty(int);
int         lchown(const char *, uid_t, gid_t);
int         link(const char *, const char *);
int         linkat(int, const char *, int, const char *, int);
int         lockf(int, int, off_t);
off_t       lseek(int, off_t, int);
int         nice(int);
long        pathconf(const char *, int);
int         pause(void);
int         pipe(int [2]);
int         pipe2(int [2], int);
int         posix_close(int, int);
ssize_t     pread(int, void *, size_t, off_t);
ssize_t     pwrite(int, const void *, size_t, off_t);
ssize_t     read(int, void *, size_t);
ssize_t     readlink(const char *restrict, char *restrict, size_t);
ssize_t     readlinkat(int, const char *restrict, char *restrict, size_t);
int         rmdir(const char *);
int         setegid(gid_t);
int         seteuid(uid_t);
int         setgid(gid_t);
int         setpgid(pid_t, pid_t);
int         setregid(gid_t, gid_t);
int         setresgid(gid_t, gid_t, gid_t);
int         setresuid(uid_t, uid_t, uid_t);
int         setreuid(uid_t, uid_t);
pid_t       setsid(void);
int         setuid(uid_t);
unsigned    sleep(unsigned);
void        swab(const void *restrict, void *restrict, ssize_t);
int         symlink(const char *, const char *);
int         symlinkat(const char *, int, const char *);
void        sync(void);
long        sysconf(int);
pid_t       tcgetpgrp(int);
int         tcsetpgrp(int, pid_t);
int         truncate(const char *, off_t);
char       *ttyname(int);
int         ttyname_r(int, char *, size_t);
int         unlink(const char *);
int         unlinkat(int, const char *, int);
ssize_t     write(int, const void *, size_t);
```

The **<unistd.h>** header shall declare the following external variables:

```
extern char  *optarg;
extern int    opterr, optind, optopt;
```

	Header	Prefix	Suffix	Complete Name
	<aio.h>	aio_, lio_, AIO_, LIO_		
	<arpa/inet.h>	inet_		
	<ctype.h>	to[a-z], is[a-z]		
	<dlfcn.h>	RTLD_, dli_		
	<dirent.h>	d_, DT_		
	<fcntl.h>	l_		
XSI	**<fmtmsg.h>**	MM_		
	<fnmatch.h>	FNM_		
XSI	**<ftw.h>**	FTW		
	<glob.h>	gl_, GLOB_		
	<grp.h>	gr_		
	<libintl.h>			TEXTDOMAINMAX
	<limits.h>		_MAX, _MIN	
XSI	**<math.h>**	M_		
MSG	**<mqueue.h>**	mq_, MQ_		
XSI	**<ndbm.h>**	dbm_, DBM_		
	<netdb.h>	ai_, h_, n_, p_, s_		
	<net/if.h>	if_, IF_		
	<netinet/in.h>	in_, ip_, s_, sin_, INADDR_, IPPROTO_		
IP6		in6_, in6addr_, s6_, sin6_, IPV6_		
	<netinet/tcp.h>	TCP_		
	<nl_types.h>	NL_		
	<poll.h>	pd_, ph_, ps_, POLL		
	<pthread.h>	pthread_, PTHREAD_		
	<pwd.h>	pw_		
	<regex.h>	re_, rm_, REG_		
	<sched.h>	sched_, SCHED_		
	<semaphore.h>	sem_, SEM_		
CX	**<signal.h>**	sa_, si_, sigev_, sival_, uc_, BUS_, CLD_, FPE_, ILL_, SA_, SEGV_, SI_, SIGEV_,		
XSI		ss_, sv_, SS_, TRAP_		
	<stdlib.h>	str[a-z]		
	<string.h>	str[a-z], mem[a-z], wcs[a-z]		
XSI	**<sys/ipc.h>**	ipc_, IPC_		key, pad, seq
	<sys/mman.h>	shm_, MAP_, MCL_, MS_, PROT_		
XSI	**<sys/msg.h>**	msg, MSG_[A-Z]		msg
XSI	**<sys/resource.h>**	rlim_, ru_, PRIO_, RLIMIT_, RUSAGE_		
	<sys/select.h>	fd_, fds_, FD_		

2.9.5.2 *Cancellation Points*

Cancellation points shall occur when a thread is executing the following functions:

accept( )	*nanosleep*( )	*recvmsg*( )
accept4( )	*open*( )	*select*( )
aio_suspend( )	*openat*( )	*send*( )
clock_nanosleep( )	*pause*( )	*sendmsg*( )
close( )	*poll*( )	*sendto*( )
connect( )	*ppoll*( )	*sigsuspend*( )
creat( )	*pread*( )	*sigtimedwait*( )
fcntl( )†	*pselect*( )	*sigwait*( )
fdatasync( )	*pthread_cond_clockwait*( )	*sigwaitinfo*( )
fsync( )	*pthread_cond_timedwait*( )	*sleep*( )
lockf( )††	*pthread_cond_wait*( )	*tcdrain*( )
mq_receive( )	*pthread_join*( )	*wait*( )
mq_send( )	*pthread_testcancel*( )	*waitid*( )
mq_timedreceive( )	*pwrite*( )	*waitpid*( )
mq_timedsend( )	*read*( )	*write*( )
msgrcv( )	*readv*( )	*writev*( )
msgsnd( )	*recv*( )	
msync( )	*recvfrom*( )	

A cancellation point may also occur when a thread is executing the following functions:

access( )	*dngettext_l*( )	*fgetws*( )
bindtextdomain( )	*dprintf*( )	*fmtmsg*( )
catclose( )	*endhostent*( )	*fopen*( )
catopen( )	*endnetent*( )	*fpathconf*( )
chmod( )	*endprotoent*( )	*fprintf*( )
chown( )	*endservent*( )	*fputc*( )
closedir( )	*faccessat*( )	*fputs*( )
closelog( )	*fchmod*( )	*fputwc*( )
ctermid( )	*fchmodat*( )	*fputws*( )
dcgettext( )	*fchown*( )	*fread*( )
dcgettext_l( )	*fchownat*( )	*freopen*( )
dcngettext( )	*fclose*( )	*fscanf*( )
dcngettext_l( )	*fcntl*( )†††	*fseek*( )
dgettext( )	*fflush*( )	*fseeko*( )
dgettext_l( )	*fgetc*( )	*fsetpos*( )
dlclose( )	*fgetpos*( )	*fstat*( )
dlopen( )	*fgets*( )	*fstatat*( )
dngettext( )	*fgetwc*( )	*ftell*( )

† When the *cmd* argument is F_SETLKW.

†† When the *function* argument is F_LOCK.

††† For any value of the *cmd* argument. +

ftello( )	+ *mkstemp*( )	*rewind*( )
futimens( )	*mktime*( )	*rewinddir*( )
fwprintf( )	*ngettext*( )	*scandir*( )
fwrite( )	*ngettext_l*( )	*scanf*( )
fwscanf( )	*opendir*( )	*seekdir*( )
getaddrinfo( )	*openlog*( )	*sem_clockwait*( )
getc( )	*pathconf*( )	*sem_timedwait*( )
getc_unlocked( )	*perror*( )	*sem_wait*( )
getchar( )	*popen*( )	*semop*( )
getchar_unlocked( )	*posix_fadvise*( )	*sethostent*( )
getcwd( )	*posix_fallocate*( )	*setnetent*( )
getdelim( )	*posix_getdents*( )	*setprotoent*( )
getgrgid_r( )	*posix_madvise*( )	*setservent*( )
getgrnam_r( )	*posix_openpt*( )	*stat*( )
gethostid( )	*posix_spawn*( )	*strerror_l*( )
gethostname( )	*posix_spawnp*( )	*strerror_r*( )
getline( )	*posix_typed_mem_open*( )	*strftime*( )
getlogin_r( )	*printf*( )	*strftime_l*( )
getnameinfo( )	*psiginfo*( )	*symlink*( )
getpwnam_r( )	*psignal*( )	*symlinkat*( )
getpwuid_r( )	*pthread_rwlock_clockrdlock*( )	*sync*( )
gettext( )	*pthread_rwlock_clockwrlock*( )	*syslog*( )
gettext_l( )	*pthread_rwlock_rdlock*( )	*tmpfile*( )
getwc( )	*pthread_rwlock_timedrdlock*( )	*tmpnam*( )
getwchar( )	*pthread_rwlock_timedwrlock*( )	*ttyname_r*( )
glob( )	*pthread_rwlock_wrlock*( )	*tzset*( )
iconv_close( )	*ptsname*( )	*ungetc*( )
iconv_open( )	*ptsname_r*( )	*ungetwc*( )
link( )	*putc*( )	*unlink*( )
linkat( )	*putc_unlocked*( )	*unlinkat*( )
lio_listio( )	*putchar*( )	*utimensat*( )
localtime_r( )	*putchar_unlocked*( )	*utimes*( )
lockf( )	*puts*( )	*vdprintf*( )
lseek( )	*putwc*( )	*vfprintf*( )
lstat( )	*putwchar*( )	*vfwprintf*( )
mkdir( )	*readdir_r*( )	*vprintf*( )
mkdirat( )	*readlink*( )	*vwprintf*( )
mkdtemp( )	*readlinkat*( )	*wcsftime*( )
mkfifo( )	*remove*( )	*wordexp*( )
mkfifoat( )	*rename*( )	*wprintf*( )
mknod( )	*renameat*( )	*wscanf*( )
mknodat( )		

In addition, a cancellation point may occur when a thread is executing any function that this standard does not require to be thread-safe but the implementation documents as being thread-safe. If a thread is cancelled while executing a non-thread-safe function, the behavior is undefined.

An implementation shall not introduce cancellation points into any other functions specified in this volume of POSIX.1-202x.

The side-effects of acting upon a cancellation request while suspended during a call of a function are the same as the side-effects that may be seen in a single-threaded program when a call to a function is interrupted by a signal and the given function returns [EINTR]. Any such side-

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 following:

- 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 *open*( ).

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

[ENOMEM] Insufficient memory available.

EXAMPLES

See the examples for *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 that 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 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 475) 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*

CHANGE HISTORY

First released in Issue 8.

NAME

catopen — open a message catalog

SYNOPSIS

```
#include <nl_types.h>

nl_catd catopen(const char *name, int oflag);
```

DESCRIPTION

The *catopen*( ) function shall open a message catalog and return a message catalog descriptor. The *name* argument specifies the name of the message catalog to be opened. If *name* contains a `'/'`, then *name* specifies a pathname for the message catalog. Otherwise, the environment variable *NLSPATH* is used with *name* substituted for the `%N` conversion specification (see XBD Chapter 8, on page 155); if *NLSPATH* exists in the environment when the process starts, then if the process has appropriate privileges, the behavior of *catopen*( ) is undefined. If *NLSPATH* does not exist in the environment, or if a message catalog cannot be found in any of the components specified by *NLSPATH*, then an implementation-defined default path shall be used. This default may be affected by the setting of *LC_MESSAGES* if the value of *oflag* is NL_CAT_LOCALE, or the *LANG* environment variable if *oflag* is 0. XSI When searching *NLSPATH*, *catopen*( ) shall ignore any files it finds that are not valid message catalog files. +

A message catalog descriptor shall remain valid in a process until that process closes it, or a successful call to one of the *exec* functions. A change in the setting of the *LC_MESSAGES* category may invalidate existing open catalogs.

If a file descriptor is used to implement message catalog descriptors, the FD_CLOEXEC flag shall be set; see **<fcntl.h>**.

If the value of the *oflag* argument is 0, the *LANG* environment variable is used to locate the catalog without regard to the *LC_MESSAGES* category. If the *oflag* argument is NL_CAT_LOCALE, the *LC_MESSAGES* category is used to locate the message catalog (see XBD Section 8.2, on page 157).

RETURN VALUE

Upon successful completion, *catopen*( ) shall return a message catalog descriptor for use on subsequent calls to *catgets*( ) and *catclose*( ). Otherwise, *catopen*( ) shall return (**nl_catd**) −1 and set *errno* to indicate the error.

ERRORS

The *catopen*( ) function may fail if:

[EACCES] Search permission is denied for the component of the path prefix of the message catalog or read permission is denied for the message catalog.

[EMFILE] All file descriptors available to the process are currently open.

[ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

[ENAMETOOLONG] The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

[ENFILE] Too many files are currently open in the system.

[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 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.

[ENOMEM] Insufficient storage space is available.

[ENOTDIR] A component of the path prefix of the message catalog names an existing file that is neither a directory nor a symbolic link to a directory, or the pathname of the message catalog contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EXAMPLES

None.

APPLICATION USAGE

Some implementations of *catopen*( ) use *malloc*( ) to allocate space for internal buffer areas. The *catopen*( ) function may fail if there is insufficient storage space available to accommodate these buffers.

Conforming applications must assume that message catalog descriptors are not valid after a call to one of the *exec* functions.

Application developers should be aware that guidelines for the location of message catalogs have not yet been developed. Therefore they should take care to avoid conflicting with catalogs used by other applications and the standard utilities.

To be sure that messages produced by an application running with appropriate privileges cannot be used by an attacker setting an unexpected value for *NLSPATH* in the environment to confuse a system administrator, such applications should use pathnames containing a `'/'` to get defined behavior when using *catopen*( ) to open a message catalog.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catclose( ), *catgets*( )

XBD Chapter 8 (on page 155), **<fcntl.h>**, **<nl_types.h>**,

CHANGE HISTORY

First released in Issue 2.

Issue 7

Austin Group Interpretation 1003.1-2001 #143 is applied.

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *catopen*( ) function is moved from the XSI option to the Base.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0045 [324] is applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0054 [645], XSH/TC2-2008/0055 [497], and XSH/TC2-2008/0056 [497] are applied.

NAME

dcgettext, dcgettext_l, dcngettext, dcngettext_l, dgettext, dgettext_l — message handling functions

SYNOPSIS

```
#include <libintl.h>

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

DESCRIPTION

Refer to *gettext*.

NAME

dngettext, dngettext_l — message handling functions

SYNOPSIS

```
#include <libintl.h>

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);
```

DESCRIPTION

Refer to *gettext*.

NAME

fpathconf, pathconf — get configurable pathname variables

SYNOPSIS

```
#include <unistd.h>

long fpathconf(int fildes, int name);
long pathconf(const char *path, int name);
```

DESCRIPTION

The *fpathconf*( ) and *pathconf*( ) functions shall determine the current value of a configurable limit or option (*variable*) that is associated with a file or directory.

For *pathconf*( ), the *path* argument points to the pathname of a file or directory.

For *fpathconf*( ), the *fildes* argument is an open file descriptor.

The *name* argument represents the variable to be queried relative to that file or directory. Implementations shall support all of the variables listed in the following table and may support others. The variables in the following table come from **<limits.h>** or **<unistd.h>** and the symbolic constants, defined in **<unistd.h>**, are the corresponding values used for *name*.

Variable	Value of *name*	Requirements
{FILESIZEBITS}	_PC_FILESIZEBITS	4,7
{LINK_MAX}	_PC_LINK_MAX	1
{MAX_CANON}	_PC_MAX_CANON	2
{MAX_INPUT}	_PC_MAX_INPUT	2
{NAME_MAX}	_PC_NAME_MAX	3,4
{PATH_MAX}	_PC_PATH_MAX	4,5
{PIPE_BUF}	_PC_PIPE_BUF	6
{POSIX2_SYMLINKS}	_PC_2_SYMLINKS	4
{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	10
{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	10
{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	10
{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	10
{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	10
{SYMLINK_MAX}	_PC_SYMLINK_MAX	4,9
{TEXTDOMAIN_MAX}	_PC_TEXTDOMAIN_MAX	3,4
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3,4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_FALLOC	_PC_FALLOC	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8
_POSIX_TIMESTAMP_RESOLUTION	_PC_TIMESTAMP_RESOLUTION	1

NAME

getresgid — get real group ID, effective group ID, and saved set-group-ID

SYNOPSIS

XSI
```
#include <unistd.h>

int getresgid(gid_t *rgid, gid_t *egid, gid_t *sgid);
```

DESCRIPTION

The *getresgid*( ) function shall store the real group ID, effective group ID, and saved set-group-ID of the calling process in the locations pointed to by the arguments *rgid*, *egid*, and *sgid*, respectively.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 8.

NAME

getresuid — get real user ID, effective user ID, and saved set-user-ID

SYNOPSIS

XSI
```
#include <unistd.h>

int getresuid(uid_t *ruid, uid_t *euid, uid_t *suid);
```

DESCRIPTION

The *getresuid*( ) function shall store the real user ID, effective user ID, and saved set-user-ID of the calling process in the locations pointed to by the arguments *ruid*, *euid*, and *suid*, respectively.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 8.

NAME

dgettext, dgettext_l, dcgettext, dcgettext_l, gettext, gettext_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 *bindtextdomain*( )), 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 *msgfmt*).
- 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 application shall ensure that the *msgid* and *msgid_plural* arguments are 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 *category* argument 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:

XSI 1. If the *NLSPATH* environment variable is set to a non-empty string, an *NLSPATH* search shall be performed as described in XBD Section 8.2 (on page 157). If *NLSPATH* identifies more than one template to use, each template in turn shall be used until a valid messages object is found.

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.

XSI For the NLSPATH search and 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 one of the *gettext* family of functions returns a message string that was found in a messages object, it shall convert the codeset of the message string to the output codeset if a codeset is specified in the messages object (see *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**
- **/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 (for example, 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 reopen 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 (for example, `"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 does not 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*

CHANGE HISTORY

First released in Issue 8.

NAME

ngettext, ngettext_l — message handling functions

SYNOPSIS

```
#include <libintl.h>

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

Refer to *gettext*.

NAME

setresgid — set real group ID, effective group ID, and saved set-group-ID

SYNOPSIS

XSI

```
#include <unistd.h>

int setresgid(gid_t rgid, gid_t egid, gid_t sgid);
```

DESCRIPTION

The *setresgid*( ) function shall set the real group ID, effective group ID, and saved set-group-ID of the calling process to the values specified by *rgid*, *egid*, and *sgid*, respectively.

If an argument is −1, the corresponding ID shall not be changed.

Only a process with appropriate privileges can set the real group ID, effective group ID, and saved set-group-ID to any valid value.

A non-privileged process can set its real group ID, effective group ID, and saved set-group-ID, each to one of the values that it currently holds in its real group ID, effective group ID, or saved set-group-ID.

The real group ID, effective group ID, and saved set-group-ID can be set to different values in the same call.

Any supplementary group IDs of the calling process shall remain unchanged.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error, and none of the IDs shall be changed.

ERRORS

The *setresgid*( ) function shall fail if:

[EINVAL]	The value of the *rgid*, *egid*, or *sgid* argument is invalid or out-of-range.
[EPERM]	The calling process does not have appropriate privileges and an attempt was made to change the real group ID, effective group ID, or saved set-group-ID to a value that is not currently present in one of those IDs.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresuid*( ), *setreuid*( ), *setuid*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 8.

NAME

setresuid — set real user ID, effective user ID, and saved set-user-ID

SYNOPSIS

XSI

```
#include <unistd.h>

int setresuid(uid_t ruid, uid_t euid, uid_t suid);
```

DESCRIPTION

The *setresuid*( ) function shall set the real user ID, effective user ID, and saved set-user-ID of the calling process to the values specified by *ruid*, *euid*, and *suid*, respectively.

If an argument is −1, the corresponding ID shall not be changed.

Only a process with appropriate privileges can set the real user ID, effective user ID, and saved set-user-ID to any valid value.

A non-privileged process can set its real user ID, effective user ID, and saved set-user-ID, each to one of the values that it currently holds in its real user ID, effective user ID, or saved set-user-ID.

The real user ID, effective user ID, and saved set-user-ID can be set to different values in the same call.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error, and none of the IDs shall be changed.

ERRORS

The *setresuid*( ) function shall fail if:

[EINVAL]	The value of the *ruid*, *euid*, or *suid* argument is invalid or out-of-range.
[EPERM]	The calling process does not have appropriate privileges and an attempt was made to change the real user ID, effective user ID, or saved set-user-ID to a value that is not currently present in one of those IDs or an attempt was made to change the real user ID to a value not permitted by the implementation.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getresgid*( ), *getresuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setresgid*( ), *setreuid*( ), *setuid*( )

XBD **<unistd.h>**

CHANGE HISTORY

First released in Issue 8.

NAME

textdomain — text domain manipulation function

SYNOPSIS

```
#include <libintl.h>

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

DESCRIPTION

Refer to *bindtextdomain*( ).

STDOUT

When the −**v** option is specified, standard output shall be formatted as:

```
"%s\n", <pathname or command>
```

When the −**V** option is specified, standard output shall be formatted as:

```
"%s\n", <unspecified>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

When the −**v** or −**V** options are specified, the following exit values shall be returned:

0	Successful completion.
>0	The *command_name* could not be found or an error occurred.

Otherwise, the following exit values shall be returned:

126	The utility specified by *command_name* was found but could not be invoked.
127	An error occurred in the *command* utility or the utility specified by *command_name* could not be found.

Otherwise, the exit status of *command* shall be that of the simple command specified by the arguments to *command*.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility is required to be intrinsic. See Section 1.7 (on page 2336) for details.

The order for command search allows functions to override regular built-ins and path searches. This utility is necessary to allow functions that have the same name as a utility to call the utility (instead of a recursive call to the function).

The system default path is available using *getconf*; however, since *getconf* may need to have the *PATH* set up before it can be called itself, the following can be used:

```
command -p getconf PATH
```

There are some advantages to suppressing the special characteristics of special built-ins on occasion. For example:

```
command exec > unwritable-file
```

does not cause a non-interactive script to abort, so that the output status can be checked by the script.

The *command*, *env*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127 + if an error occurs so that applications can distinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PATH Determine the location of the *utility*, as described in XBD Chapter 8 (on page 155). If *PATH* is specified as a *name=value* operand to *env*, the *value* given shall be used in the search for *utility*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If no *utility* operand is specified, each *name=value* pair in the resulting environment shall be written in the form:

```
"%s=%s\n", <name>, <value>
```

If the *utility* operand is specified, the *env* utility shall not write to standard output.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If *utility* is invoked, the exit status of *env* shall be the exit status of *utility*; otherwise, the *env* utility shall exit with one of the following values:

0	The *env* utility completed successfully.
1–125	An error occurred in the *env* utility.
126	The utility specified by *utility* was found but could not be invoked.
127	The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason. +

Historical implementations of the *env* utility use the *execvp*() or *execlp*() functions defined in the System Interfaces volume of POSIX.1-202x to invoke the specified utility; this provides better performance and keeps users from having to escape characters with special meaning to the shell. Therefore, shell functions, special built-ins, and built-ins that are only provided by the shell are not found by this type of *env* implementation. However, *env* can be implemented as a shell built-in, in which case it may be able to execute shell functions and built-ins. An application wishing to ensure execution of a non-built-in utility can use *exec* in a subshell for this purpose.

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 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, the *n* variable is 1 (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 201).

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 *msgfmt*).

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *gettext* and *ngettext*:

	LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 157) for the precedence of internationalization variables used to determine the values of locale categories.)
XSI	*LANGUAGE*	Determine the location of messages objects if *NLSPATH* is not set or the evaluation of *NLSPATH* did not lead to a suitable messages object being found.
	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.
	TEXTDOMAIN	Specify the text domain name. (See XBD Section 3.374 (on page 81).)
XSI	*TEXTDOMAINDIR*	Specify the pathname to the messages object hierarchy. *NLSPATH* shall have precedence over *TEXTDOMAINDIR*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See 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 ENVIRONMENT VARIABLES sections 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 -ed 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 -ed 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 101).

Unlike other standard utilities, the behavior of *gettext* and *ngettext* is not undefined when *NLSPATH* overrides the system default path; see XBD Section 8.2 (on page 157). 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 115), Chapter 8 (on page 155), Section 12.2 (on page 201)

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

CHANGE HISTORY

First released in Issue 8.

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 201).

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 XSH *fprintf*( ), on page 909). 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

Not used.

INPUT FILES

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

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *msgfmt*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 157) for the precedence of internationalization variables used to determine the values of locale categories.)

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

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.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not 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"
```

then *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 utility.

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

```
"charset=codeset"
```

then *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 XSH *gettext* and *gettext*. The output string's codeset shall be determined by the current or specified locale's codeset.

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.

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 values for *flag* 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*

CHANGE HISTORY

First released in Issue 8.

NAME

ngettext — retrieve text string from messages object

SYNOPSIS

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

DESCRIPTION

Refer to *gettext*.

PATH Determine the search path used to locate the utility to be invoked. See XBD Chapter 8 (on page 155).

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If *utility* is invoked, the exit status of *nice* shall be the exit status of *utility*; otherwise, the *nice* utility shall exit with one of the following values:

1-125	An error occurred in the *nice* utility.
126	The utility specified by *utility* was found but could not be invoked.
127	The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The only guaranteed portable uses of this utility are:

nice utility
Run *utility* with the default higher or equal nice value.

nice **−n** <*positive integer*> *utility*
Run *utility* with a higher nice value.

On some implementations they have no discernible effect on the invoked utility and on some others they are exactly equivalent.

Historical systems have frequently supported the <*positive integer*> up to 20. Since there is no error penalty associated with guessing a number that is too high, users without access to the system conformance document (to see what limits are actually in place) could use the historical 1 to 20 range or attempt to use very large numbers if the job should be truly low priority.

The nice value of a process can be displayed using the command:

```
ps -o nice
```

The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any +

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).

LC_MESSAGES
Determine 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 message catalogs for the processing of *LC_MESSAGES*.

PATH Determine the search path that is used to locate the utility to be invoked. See XBD Chapter 8 (on page 155).

ASYNCHRONOUS EVENTS

The *nohup* utility shall take the standard action for all signals except that SIGHUP shall be ignored.

STDOUT

If the standard output is not a terminal, the standard output of *nohup* shall be the standard output generated by the execution of the *utility* specified by the operands. Otherwise, nothing shall be written to the standard output.

STDERR

If the standard output is a terminal, a message shall be written to the standard error, indicating the name of the file to which the output is being appended. The name of the file shall be either **nohup.out** or **$HOME/nohup.out**.

OUTPUT FILES

Output written by the named utility is appended to the file **nohup.out** (or **$HOME/nohup.out**), if the conditions hold as described in the DESCRIPTION.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

126 The utility specified by *utility* was found but could not be invoked.

127 An error occurred in the *nohup* utility or the utility specified by *utility* could not be found.

Otherwise, the exit status of *nohup* shall be that of the utility specified by the *utility* operand.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason. +

NAME

readlink — display the contents of a symbolic link

SYNOPSIS

```
readlink [−n] file
```

DESCRIPTION

If the *file* operand names a symbolic link, the *readlink* utility shall not follow the symbolic link when resolving *file* and shall write the contents of the symbolic link to standard output. If the **−n** option is not specified, the output to standard output shall be followed by a <newline> character.

If *file* does not name a symbolic link, *readlink* shall write a diagnostic message to standard error and exit with non-zero status.

OPTIONS

The *readlink* utility shall conform to XBD Section 12.2 (on page 201).

The following option shall be supported:

−n	Do not output a trailing <newline> character.

OPERANDS

The following operand shall be supported:

file	A pathname of a symbolic link to be read.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *readlink*:

LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 157) for the precedence of internationalization variables used to determine the values of locale categories.)
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 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.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See 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

None.

EXAMPLES

None.

RATIONALE

The *readlink* utility was added because using *ls* −**l** to obtain the contents of a symbolic link is difficult if the output includes more than one occurrence of the string `" -> "`.

The −**f** option found in many implementations was not included, as the *realpath* utility provides equivalent functionality with a choice of behaviors.

FUTURE DIRECTIONS

None.

SEE ALSO

ln, *ls*, *realpath*

XBD Chapter 8 (on page 155), Section 12.2 (on page 201)

XSH *readlink*( )

CHANGE HISTORY

First released in Issue 8.

NAME

realpath — resolve a pathname

SYNOPSIS

```
realpath [-E|-e] file
```

DESCRIPTION

The *realpath* utility shall canonicalize the pathname specified by the *file* operand as follows:

If a call to the *realpath*( ) function with the specified pathname as its first argument would succeed, the canonicalized pathname shall be the pathname that would be returned by that *realpath*( ) call. Otherwise:

- If the −**e** option is specified, the canonicalization shall fail.
- If the −**E** option is specified, then if a call to the *realpath*( ) function with the specified pathname as its first argument would encounter an error condition other than [ENOENT], the canonicalization shall fail; if the call would encounter an [ENOENT] error, *realpath* shall expand all symbolic links that would be encountered in an attempt to resolve the specified pathname using the algorithm specified in XBD Section 4.14 (on page 93), except that any trailing <slash> characters that are not also leading <slash> characters shall be ignored. If this expansion succeeds and the path prefix of the expanded pathname resolves to an existing directory, the canonicalized pathname shall be the expanded pathname. In all other cases, the canonicalization shall fail. If the expanded pathname is not empty, does not begin with a <slash>, and has exactly one pathname component, it shall be treated as if it had a path prefix of `"./"`.
- If no options are specified, *realpath* shall canonicalize the specified pathname in an unspecified manner such that the resulting absolute pathname does not contain any components that refer to files of type symbolic link and does not contain any components that are dot or dot-dot.

Upon successful canonicalization, *realpath* shall write the canonicalized pathname, followed by a <newline> character, to standard output.

If canonicalization fails, or the canonicalized pathname is empty, nothing shall be written to standard output, a diagnostic message shall be written to standard error, and *realpath* shall exit with non-zero status.

OPTIONS

The *realpath* utility shall conform to XBD Section 12.2 (on page 201).

The following options shall be supported:

−**E**	Do not treat it as an error if attempting to resolve the last component of the canonicalized form of the *file* operand results in an [ENOENT] error condition.
−**e**	Treat it as an error if attempting to resolve the last component of the canonicalized form of the *file* operand results in an [ENOENT] error condition.

Specifying more than one of the mutually-exclusive options −**E** and −**e** shall not be considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

The following operand shall be supported:

file	A pathname to be canonicalized.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *realpath*:

LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 157) for the precedence of internationalization variables used to determine the values of locale categories.)
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 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.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See 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

If neither the −**e** nor the −**E** option is specified, some implementations behave as if −**e** had been specified and others as if −**E** had been specified, but there are also implementations where the behavior differs from both of these. For example, the *mksh* shell has an internal implementation of *realpath* that canonicalizes **/dir/regular_file/..** to **/dir**, whereas the *realpath*() function would return an [ENOTDIR] error in this case. Portable applications should always specify either −**e** or −**E**.

EXAMPLES

None.

RATIONALE

The *realpath* utility was added in preference to a −**f** option found in some implementations of the *readlink* utility because it allows the application to specify whether or not a missing final component is to be treated as an error.

The behavior with the −**E** option when *file* does not resolve (with symbolic links followed) to an existing file is not the same as simply calling *realpath*() with the path prefix of the *file* operand and writing the resulting pathname, a <slash>, and the last component of *file* to standard output. For example, if **/tmp/nofile** does not exist, and *file* is **A/B** where **A** is an existing directory and **B** is a symbolic link to **/tmp/nofile**, *realpath* with −**E** will output **/tmp/nofile**, but if **B** is a symbolic link to **/tmp/nofile/foo**, *realpath* with −**E** will treat this as an error. In both cases `realpath("A/B")` would fail with *errno* set to [ENOENT]. Even though `realpath("A")` would succeed, in neither case is anything ending **/B** the result.

Trailing <slash> characters (that follow a non-<slash>) are handled differently with −**E** than with −**e**. With −**e** they are handled as for the *realpath*() function. With −**E** they are sometimes effectively ignored, and they are never included in the output. For example, if **/tmp/nofile** does not exist and **/tmp/regfile** is an existing regular file:

```
$ realpath -E /tmp/nofile/
/tmp/nofile
$ realpath -E /tmp/regfile/
realpath: /tmp/regfile/: Not a directory
```

Although the behavior of the *realpath* utility is specified by reference to the *realpath*() function, which is part of the XSI option, non-XSI implementations that do not support *realpath*() are nevertheless required to implement *realpath* in accordance with the requirements described in this standard for *realpath*().

FUTURE DIRECTIONS

None.

SEE ALSO

ln, *ls*, *pwd*, *readlink*

XBD Chapter 8 (on page 155), Section 12.2 (on page 201)

XSH Section 2.3 (on page 475), *realpath*()

CHANGE HISTORY

First released in Issue 8.

127 The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason. +

EXAMPLES

It is frequently desirable to apply *time* to pipelines or lists of commands. This can be done by placing pipelines and command lists in a single file; this file can then be invoked as a utility, and the *time* applies to everything in the file.

Alternatively, the following command can be used to apply *time* to a complex command:

```
time sh -c 'complex-command-line'
```

RATIONALE

When the *time* utility was originally proposed to be included in the ISO POSIX-2: 1993 standard, questions were raised about its suitability for inclusion on the grounds that it was not useful for conforming applications, specifically:

- The underlying CPU definitions from the System Interfaces volume of POSIX.1-202x are vague, so the numeric output could not be compared accurately between systems or even between invocations.
- The creation of portable benchmark programs was outside the scope this volume of POSIX.1-202x.

However, *time* does fit in the scope of user portability. Human judgement can be applied to the analysis of the output, and it could be very useful in hands-on debugging of applications or in providing subjective measures of system performance. Hence it has been included in this volume of POSIX.1-202x.

The default output format has been left unspecified because historical implementations differ greatly in their style of depicting this numeric output. The −**p** option was invented to provide scripts with a common means of obtaining this information.

In the KornShell, *time* is a shell reserved word that can be used to time an entire pipeline, rather than just a simple command. The POSIX definition has been worded to allow this implementation. Consideration was given to invalidating this approach because of the historical model from the C shell and System V shell. However, since the System V *time* utility historically has not produced accurate results in pipeline timing (because the constituent processes are not all owned by the same parent process, as allowed by POSIX), it did not seem worthwhile to break historical KornShell usage.

The term *utility* is used, rather than *command*, to highlight the fact that shell compound commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility* includes user application programs and shell scripts, not just the standard utilities.

NAME

timeout — execute a utility with a time limit

SYNOPSIS

```
timeout [-fp] [-k time] [-s signal_name] duration utility [argument...]
```

DESCRIPTION

The *timeout* utility shall execute the utility named by the *utility* operand, with arguments supplied as the *argument* operands (if any), in a child process. If the value of the *duration* operand is non-zero and the child process has not terminated after the specified time period, *timeout* shall send the signal specified by the −**s** option, or the SIGTERM signal if −**s** is not given.

If the −**f** option is specified, the signal shall be sent only to the child process. Otherwise, it is implementation defined which one of the following methods is used to signal additional processes:

- The *timeout* utility ensures it is a process group leader before creating the child process which executes the utility, in which case it shall send the signal to its process group.
- The *timeout* utility arranges for any descendents of the child process that are orphaned to have their parent process changed to the *timeout* utility, in which case the signal shall be sent to the child process and all of its descendents.

If the subsequent wait status of the child process shows that it was stopped by a signal, a SIGCONT signal shall also be sent in the same manner as the first signal; otherwise, a SIGCONT signal may be sent in the same manner.

If the −**k** option is specified, and the child process created to execute the utility still has not terminated after the time period specified by the *time* option-argument has elapsed since the first signal was sent, *timeout* shall send a SIGKILL signal in the same manner as the first signal. If *timeout* receives a signal and propagates it to the child process (see ASYNCHRONOUS EVENTS below), this shall be treated as the first signal.

OPTIONS

The *timeout* utility shall conform to XBD Section 12.2 (on page 201).

The following options shall be supported:

−**f** Only time out the utility itself, not its descendents.

−**k** *time* Send a SIGKILL signal if the child process created to execute the utility has not terminated after the time period specified by *time* has elapsed since the first signal was sent. The value of *time* shall be interpreted as specified for the *duration* operand (see OPERANDS below).

−**p** Always preserve (mimic) the wait status of the executed utility, even if the time limit was reached.

−**s** *signal_name*
Specify the signal to send when the time limit is reached, using one of the symbolic names defined in the **<signal.h>** header. Values of *signal_name* shall be recognized in a case-independent fashion, without the SIG prefix. By default, SIGTERM shall be sent.

OPERANDS

The following operands shall be supported:

duration The maximum amount of time to allow the utility to run, specified as a decimal number with an optional decimal fraction and an optional suffix, which can be:

s	seconds
m	minutes
h	hours
d	days

If a decimal fraction is present, the application shall ensure that it is separated from the units by a <period>. If no suffix is present, the value shall specify seconds.

If the value is zero, *timeout* shall not enforce a time limit.

utility The name of a utility that is to be executed. If the *utility* operand names any of the special built-in utilities in Section 2.14 (on page 2382), the results are undefined.

argument Any string to be supplied as an argument when executing the utility named by the *utility* operand.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *timeout*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 157) for the precedence of internationalization variables used to determine the values of locale categories.)

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 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.

PATH Determine the search path that is used to locate the utility to be executed. See XBD Section 8.3 (on page 162).

ASYNCHRONOUS EVENTS

The default behavior specified in Section 1.4 (on page 2328) shall apply, except that:

- The *timeout* utility shall ignore SIGTTIN and SIGTTOU signals.
- The *timeout* utility may alter the disposition of SIGALRM if the inherited disposition was for it to be ignored.
- If the signal specified with the −**s** option, or any signal whose default action is to terminate the process, is delivered to the *timeout* utility, then unless the signal is SIGKILL or SIGSTOP, the *timeout* utility shall immediately send the same signal to the process or processes to which it would send a signal when the time limit is reached. If the delivered signal is SIGALRM, *timeout* may behave as if the time limit had been reached instead of sending SIGALRM.

- If the −**f** option is not specified, then if *timeout* sends a signal to its process group, it shall briefly change the disposition of that signal to ignored while it sends the signal, so that it does not receive the signal itself.

With the single exception of the signal specified with the −**s** option, or SIGTERM if −**s** is not used, all signal dispositions inherited by the utility specified by the *utility* operand shall be the same as the disposition that *timeout* inherited.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If the −**p** option is not specified and the time limit was reached:

- If the −**k** option was not specified or the utility terminated before the time period specified by the *time* option-argument elapsed since the first signal was sent, the exit status shall be 124.
- If the −**k** option was specified and the SIGKILL signal was sent, it is unspecified whether the exit status is 124 or the behavior is as if the −**p** option was specified.

Otherwise, if the executed utility terminated by exiting, the exit status of *timeout* shall be that of the utility; if the utility was terminated by a signal, *timeout* shall terminate itself with the same signal while ensuring that a core image is not created.

If an error occurs, the following exit values shall be returned:

125	An error other than the two described below occurred.
126	The utility specified by *utility* was found but could not be executed.
127	The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Unlike the *kill* utility, the −**s** option of *timeout* is not required to accept the symbolic name 0 to represent signal value zero.

When the value of *duration* is zero, *timeout* does not time out the utility, but it does still perform signal propagation (including to descendents of the utility if −**f** is not specified).

Regardless of locale, the <period> character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in the *duration* operand and the *time* option-argument.

The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127 if a utility to be invoked cannot be found, so that applications can distinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason. The *timeout* utility extends these special exit codes to 125 and 124, with the meanings described in EXIT STATUS. A *timeout* exit status below 124 can only result from passing through the exit status of the executed utility.

EXAMPLES

None.

RATIONALE

Some *timeout* implementations make themselves a process group leader (when −**f** is not used) in order to be able to send signals to descendents of the child process. However, using this method means that any descendents which change their process group do not receive the signal. To ensure all descendents receive the signal, some implementations instead make use of a feature whereby descendents that are orphaned have their parent process changed to the *timeout* utility—that is, *timeout* becomes their ``reaper''—together with the ability of a reaper to send a signal to all of its descendents.

Some historical *timeout* implementations exited with status 128+*signal_number* when the child process was terminated by a signal before the time limit was reached (or when −**p** was used). This is reasonable when *timeout* is invoked from a shell which sets $? to 128+*signal_number*, but not all shells do that. In particular, the KornShell sets $? to 256+*signal_number* and so an exit status of 128+*signal_number* from *timeout* would be misleading. In order to avoid any possible ambiguity, this standard requires that *timeout* mimics the wait status of the child process by terminating itself with the same signal. When it does this it needs to ensure that it does not create a core image, otherwise it could overwrite one created by the invoked utility.

The *timeout* utility ignores SIGTTIN and SIGTTOU so that if the utility it executes reads from or writes to the controlling terminal and this generates a SIGTTIN or SIGTTOU for the process group, *timeout* will not be stopped by the signal and can still time out the utility.

Some historical *timeout* implementations always set the disposition for SIGTTIN and SIGTTOU in the child process to default, even if these signals were inherited as ignored. This could result in processes being stopped unexpectedly. Likewise, they did not ensure that for signals they caught, the disposition inherited by the executed utility was the same as the disposition that was inherited by *timeout*. This meant that, for example, if *timeout* was used in a script that was run with *nohup*, the utility executed by *timeout* would unexpectedly not be protected from SIGHUP. This standard requires that all signal dispositions inherited by the utility specified by the *utility* operand are the same as the disposition that *timeout* inherited, with the single exception of the signal that *timeout* sends when the time limit is reached, which needs to be inherited as default in order for the timeout to take effect (without resorting to SIGKILL if −**k** is specified).

Some historical *timeout* implementations only propagated a subset of the signals whose default action is to terminate the process to the child process if one was delivered to the *timeout* utility. Propagating these signals is beneficial, as otherwise termination of the *timeout* utility by a signal results in the utility it executed being left running indefinitely (unless it also received the signal, for example a terminal-generated SIGINT). There is no reason to select a subset of these signals to be propagated, therefore this standard requires them all to be propagated (except SIGKILL, which cannot). In the event that a user wants to prevent the utility being timed out, sending *timeout* a SIGKILL can be used for this purpose.

FUTURE DIRECTIONS

None.

SEE ALSO

kill

XBD Chapter 8 (on page 155), Section 12.2 (on page 201), **<signal.h>** (on page 329)

CHANGE HISTORY

First released in Issue 8.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0	All invocations of *utility* returned exit status zero.
1-125	A command line meeting the specified requirements could not be assembled, one or more of the invocations of *utility* returned a non-zero exit status, or some other error occurred.
126	The utility specified by *utility* was found but could not be invoked.
127	The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

If a command line meeting the specified requirements cannot be assembled, the utility cannot be invoked, an invocation of the utility is terminated by a signal, or an invocation of the utility exits with exit status 255, the *xargs* utility shall write a diagnostic message and exit without processing any remaining input.

APPLICATION USAGE

The 255 exit status allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the current data stream will succeed. Thus, *utility* should explicitly *exit* with an appropriate value to avoid accidentally returning with 255.

Note that since input is parsed as lines, <blank> characters separate arguments, and <backslash>, <apostrophe>, and double-quote characters are used for quoting, if *xargs* is used to bundle the output of commands like *find dir* −**print** or *ls* into commands to be executed, unexpected results are likely if any filenames contain <blank>, <newline>, or quoting characters. This can be solved by using find to call a script that converts each file found into a quoted string that is then piped to *xargs*, but in most cases it is preferable just to have *find* do the argument aggregation itself by using −**exec** with a `'+'` terminator instead of `';'`. Note that the quoting rules used by *xargs* are not the same as in the shell. They were not made consistent here because existing applications depend on the current rules. An easy (but inefficient) method that can be used to transform input consisting of one argument per line into a quoted form that *xargs* interprets correctly is to precede each non-<newline> character with a <backslash>. More efficient alternatives are shown in Example 2 and Example 5 below.

On implementations with a large value for {ARG_MAX}, *xargs* may produce command lines longer than {LINE_MAX}. For invocation of utilities, this is not a problem. If *xargs* is being used to create a text file, users should explicitly set the maximum command line length with the −**s** option.

The *command*, *env*, *nice*, *nohup*, *time*, *timeout*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ``failure to find a utility'' from ``invoked utility exited with an error indication''. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ``normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason. +

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...
```

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 shall be 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 201).

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** values 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 values 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* then *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* then *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 item 6 in XBD Section 12.1 (on page 199).

−**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

The following environment variables shall affect the execution of *xgettext*:

LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 157) for the precedence of internationalization variables used to determine the values of locale categories.)
XSI *LANGUAGE*	Determine the location of messages objects if *NLSPATH* is not set or the evaluation of *NLSPATH* did not lead to a suitable messages object being found.
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.

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*

XBD Chapter 8 (on page 155), Section 12.2 (on page 201)

XSH *gettext*

CHANGE HISTORY

First released in Issue 8.

POSIX_DEVICE_SPECIFIC_R: Thread-Safe General Terminal
ttyname_r( )

POSIX_DYNAMIC_LINKING: Dynamic Linking
dladdr( ), *dlclose*( ), *dlerror*( ), *dlopen*( ), *dlsym*( )

POSIX_FD_MGMT: File Descriptor Management
dup( ), *dup2*( ), *dup3*( ), *fcntl*( ), *fgetpos*( ), *fseek*( ), *fseeko*( ), *fsetpos*( ), *ftell*( ), *ftello*( ), *ftruncate*( ), *lseek*( ), *rewind*( )

POSIX_FIFO: FIFO
mkfifo( )

POSIX_FIFO_FD: FIFO File Descriptor Routines
mkfifoat( ), *mknodat*( )

POSIX_FILE_ATTRIBUTES: File Attributes
chmod( ), *chown*( ), *fchmod*( ), *fchown*( ), *umask*( )

POSIX_FILE_ATTRIBUTES_FD: File Attributes File Descriptor Routines
fchmodat( ), *fchownat*( )

POSIX_FILE_LOCKING: Thread-Safe Stdio Locking
flockfile( ), *ftrylockfile*( ), *funlockfile*( ), *getc_unlocked*( ), *getchar_unlocked*( ), *putc_unlocked*( ), *putchar_unlocked*( )

POSIX_FILE_SYSTEM: File System
access( ), *chdir*( ), *closedir*( ), *creat*( ), *fchdir*( ), *fpathconf*( ), *fstat*( ), *fstatvfs*( ), *getcwd*( ), *link*( ), *mkdir*( ), *mkostemp*( ), *mkstemp*( ), *opendir*( ), *pathconf*( ), *posix_getdents*( ), *readdir*( ), *remove*( ), *rename*( ), *rewinddir*( ), *rmdir*( ), *stat*( ), *statvfs*( ), *tmpfile*( ), *tmpnam*( ), *truncate*( ), *unlink*( )

POSIX_FILE_SYSTEM_EXT: File System Extensions
alphasort( ), *dirfd*( ), *getdelim*( ), *getline*( ), *mkdtemp*( ), *scandir*( )

POSIX_FILE_SYSTEM_FD: File System File Descriptor Routines
faccessat( ), *fdopendir*( ), *fstatat*( ), *linkat*( ), *mkdirat*( ), *openat*( ), *renameat*( ), *unlinkat*( ), *utimensat*( )

POSIX_FILE_SYSTEM_GLOB: File System Glob Expansion
glob( ), *globfree*( )

POSIX_FILE_SYSTEM_R: Thread-Safe File System
readdir_r( )

POSIX_I18N: Internationalization
bind_textdomain_codeset( ), *bindtextdomain*( ), *catclose*( ), *catgets*( ), *catopen*( ), *dcgettext*( ), +
dcgettext_l( ), *dcngettext*( ), *dcngettext_l*( ), *dgettext*( ), *dgettext_l*( ), *dngettext*( ), *dngettext_l*( ), +
gettext( ), *gettext_l*( ), *iconv*( ), *iconv_close*( ), *iconv_open*( ), *ngettext*( ), *ngettext_l*( ), |
nl_langinfo( ), *textdomain*( )

POSIX_JOB_CONTROL: Job Control
setpgid( ), *tcgetpgrp*( ), *tcsetpgrp*( ), *tcgetsid*( )

POSIX_MAPPED_FILES: Memory Mapped Files
mmap( ), *munmap*( )

POSIX_MEMORY_PROTECTION: Memory Protection
mprotect( )

POSIX_WIDE_CHAR_DEVICE_IO: Device Input and Output
fgetwc( ), *fgetws*( ), *fputwc*( ), *fputws*( ), *fwide*( ), *fwprintf*( ), *fwscanf*( ), *getwc*( ), *getwchar*( ), *putwc*( ), *putwchar*( ), *ungetwc*( ), *vfwprintf*( ), *vfwscanf*( ), *vwprintf*( ), *vwscanf*( ), *wprintf*( ), *wscanf*( )

XSI_C_LANG_SUPPORT: XSI General C Library
a64l( ), *daylight*, *drand48*( ), *erand48*( ), *ffs*( ), *ffsl*( ), *ffsll*( ), *getdate*( ), *hcreate*( ), *hdestroy*( ), *hsearch*( ), *initstate*( ), *insque*( ), *jrand48*( ), *l64a*( ), *lcong48*( ), *lfind*( ), *lrand48*( ), *lsearch*( ), *memccpy*( ), *mrand48*( ), *nrand48*( ), *random*( ), *remque*( ), *seed48*( ), *setstate*( ), *signgam*, *srand48*( ), *srandom*( ), *strptime*( ), *swab*( ), *tdelete*( ), *tfind*( ), *timezone*, *tsearch*( ), *twalk*( )

XSI_DBM: XSI Database Management
dbm_clearerr( ), *dbm_close*( ), *dbm_delete*( ), *dbm_error*( ), *dbm_fetch*( ), *dbm_firstkey*( ), *dbm_nextkey*( ), *dbm_open*( ), *dbm_store*( )

XSI_DEVICE_IO: XSI Device Input and Output
fmtmsg( ), *readv*( ), *writev*( )

XSI_DEVICE_SPECIFIC: XSI General Terminal
grantpt( ), *posix_openpt*( ), *ptsname*( ), *unlockpt*( )

XSI_FILE_SYSTEM: XSI File System
basename( ), *dirname*( ), *lockf*( ), *mknod*( ), *nftw*( ), *realpath*( ), *seekdir*( ), *sync*( ), *telldir*( )

XSI_GENERAL_TERMINAL_R: XSI Thread-Safe General Terminal
ptsname_r( )

XSI_IPC: XSI Interprocess Communication
ftok( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), *semctl*( ), *semget*( ), *semop*( ), *shmat*( ), *shmctl*( ), *shmdt*( ), *shmget*( )

XSI_MATH: XSI Maths Library
j0( ), *j1*( ), *jn*( ), *y0*( ), *y1*( ), *yn*( )

XSI_MULTI_PROCESS: XSI Multiple Process
getpriority( ), *getrlimit*( ), *getrusage*( ), *nice*( ), *setpriority*( ), *setrlimit*( )

XSI_SIGNALS: XSI Signal
killpg( ), *sigaltstack*( )

XSI_SINGLE_PROCESS: XSI Single Process
gethostid( ), *putenv*( )

XSI_SYSTEM_DATABASE: XSI System Database
endpwent( ), *getpwent*( ), *setpwent*( )

XSI_SYSTEM_LOGGING: XSI System Logging
closelog( ), *openlog*( ), *setlogmask*( ), *syslog*( )

XSI_THREADS_EXT: XSI Threads Extensions
pthread_attr_getstack( ), *pthread_attr_setstack*( )

XSI_USER_GROUPS: XSI User and Group
endgrent( ), *endutxent*( ), *getgrent*( ), *getresgid*( ), *getresuid*( ), *getutxent*( ), *getutxid*( ), +
getutxline( ), *pututxline*( ), *setgrent*( ), *setregid*( ), *setresgid*( ), *setresuid*( ), *setreuid*( ), *setutxent*( ) +

XSI_WIDE_CHAR: XSI Wide-Character Library
wcswidth( ), *wcwidth*( )