IEEE P1003.2 Draft 11.2 – September 1991

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only, and subject to the restrictions contained herein.

Permission is hereby also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copies of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

2.6 Environment Variables

Environment variables defined in this clause affect the operation of multiple utilities and applications. There are other environment variables that are of interest only to specific utilities. Environment variables that apply to a single utility only are defined as part of the utility description. See the Environment Variables subclause of the utility descriptions for information on environment variable usage.

The value of an environment variable is a string of characters, as described in 2.7 in POSIX.1 {8}.

Environment variable names used by the standard utilities shall consist solely of uppercase letters, digits, and the _ (underscore) from the characters defined in 2.4. The namespace of environment variable names containing lowercase letters shall be reserved for applications. Applications can define any environment variables with names from this namespace without modifying the behavior of the standard utilities.

If the following variables are present in the environment during the execution of an application or utility, they are given the meaning described below. They may be put into the environment, or changed, by either the implementation or the user. If they are defined in the utility's environment, the standard utilities assume they have the specified meaning. Conforming applications shall not set these environment variables to have meanings other than as described. See 7.2 and 3.12 for methods of accessing these variables.

HOME	A pathname of the user's home directory.
LANG	This variable shall determine the locale category for any category not specifically selected via a variable starting with **LC_**. **LANG** and the **LC_** variables can be used by applications to determine the language for messages and instructions, collating sequences, date formats, etc. Additional semantics of this variable, if any, are implementation defined.
LC_ALL	This variable shall override the value of the **LANG** variable and the value of any of the other variables starting with **LC_**.
LC_COLLATE	This variable shall determine the locale category for character collation information within bracketed regular expressions and for sorting. This environment variable determines the behavior of ranges, equivalence classes, and multicharacter collating elements. Additional semantics of this variable, if any, are implementation defined.
LC_CTYPE	This variable shall determine the locale category for character handling functions. This environment variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters), the classification of characters (e.g., 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. 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 this standard 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. 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. Additional semantics of this variable, if any, are implementation defined.
LOGNAME	The user's login name.
PATH	The sequence of path prefixes that certain functions and utilities apply in searching for an executable file known only by a filename. The prefixes shall be separated by a colon (**:**). When a nonzero-length prefix is applied to this filename, a slash shall be inserted between the prefix and the filename. A zero-length prefix is an obsolescent feature that indicates the current working directory. It appears as two adjacent colons (**::**), as an initial colon preceding the rest of the list, or as a trailing colon following the rest of the list. A Strictly Conforming POSIX.2 Application shall use an actual pathname (such as `'.'`) to represent the current working directory in **PATH** . The list shall be searched from beginning to end, applying the filename to each prefix, until an executable file with the specified name and appropriate execution permissions is found. If the pathname being sought contains a slash, the search through the path prefixes shall not be performed. If the pathname begins with a slash, the specified path shall be resolved as described in 2.2.2.104. If **PATH** is unset or is set to null, the path search is implementation-defined.

SHELL	A pathname of the user's preferred command language interpreter. If this interpreter does not conform to the shell command language in Section 3, utilities may behave differently than described in this standard.
TMPDIR	A pathname of a directory made available for programs that need a place to create temporary files.
TERM	The terminal type for which output is to be prepared. This information is used by utilities and application programs wishing to exploit special capabilities specific to a terminal. The format and allowable values of this environment variable are unspecified.
TZ	Time-zone information. The format is described in POSIX.1 {8}8.1.1.

The environment variables **LANG**, **LC_ALL**, **LC_COLLATE**, **LC_CTYPE**, **LC_MESSAGES**, **LC_MONETARY**, **LC_NUMERIC**, and **LC_TIME** (**LC_***) provide for the support of internationalized applications. The standard utilities shall make use of these environment variables as described in this clause and the individual Environment Variables subclauses for the utilities. If these variables specify locale categories that are not based upon the same underlying code set, the results are unspecified.

For utilities used in internationalized applications, if the **LC_ALL** is not set in the environment or is set to the empty string, and if any of **LC_*** variables is not set in the environment or is set to the empty string, the operational behavior of the utility for the corresponding locale category shall be determined by the setting of the **LANG** environment variable. If the **LANG** environment variable is not set or is set to the empty string, the implementation-defined default locale shall be used.

If **LANG** (or any of the **LC_*** environment variables) contains the value `"C"`, or the value `"POSIX"`, the POSIX Locale shall be selected and the standard utilities shall behave in accordance with the rules in the 2.5.1 for the associated category.

If **LANG** (or any of the **LC_*** environment variables) 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 4.35.6.3. Referencing such a pathname shall result in that locale being used for the category indicated.

If **LANG** (or any of the **LC_*** environment variables) contains one of a set of implementation-defined values, the standard utilities shall behave in accordance with the rules in a corresponding implementation-defined locale description for the associated category.

If **LANG** (or any of the **LC_*** environment variables) contains a value that the implementation does not recognize, the behavior is unspecified.

Additional criteria for determining a valid locale name are implementation defined.

2.6.1 Environment Variables Rationale. *(This subclause is not a part of P1003.2)*

The standard is worded so that the specified variables *may* be provided to the application. There is no way that the implementation can guarantee that a utility will ever see an environment variable, as a parent process can change the environment for its children. The `env -i` command in this standard and the POSIX.1 {8} *exec* family both offer ways to remove any of these variables from the environment.

The language about locale implies that any utilities written in Standard C and conforming to POSIX.2 must issue the following call:

```
setlocale(LC_ALL, "")
```

If this were omitted, the C Standard {7} specifies that the C Locale would be used.

If any of the environment variables is invalid, it makes sense to default to an implementation-defined, consistent locale environment. It is more confusing for a user to have partial settings occur in case of a mistake. All utilities would then behave in one language/cultural environment. Furthermore, it provides a way of forcing the whole environment to be the implementation-defined default. Disastrous results could occur if a pipeline of utilities partially use the environment variables in different ways. In this case, it would be appropriate for utilities that use **LANG** and related variables to exit with an error if any of the variables are invalid. For example, users typing individual commands at a terminal might want `date` to work if **LC_MONETARY** is invalid as long as **LC_TIME** is valid. Since these are conflicting reasonable alternatives, POSIX.2 leaves the results unspecified if the locale environment variables would not produce a complete locale matching the user's specification.

The locale settings of individual categories cannot be truly independent and still guarantee correct results. For example, when collating two strings, characters must first be extracted from each string (governed by **LC_CTYPE**) before being mapped to collating elements (governed by **LC_COLLATE**) for comparison. That is, if **LC_CTYPE** is causing parsing according to the rules of a large, multibyte code set (potentially returning 20 000 or more distinct character code set values), but **LC_COLLATE** is set to handle only an 8-bit code set with 256 distinct characters, meaningful results are obviously impossible.

The **LC_MESSAGES** variable affects the language of messages generated by the standard utilities. This standard does not provide a means whereby applications can easily be written to perform similar feats. Future versions of POSIX.1 {8} and POSIX.2 are expected to provide both functions and utilities to accomplish multilanguage messaging (using message catalogs), but such facilities were not ready for standardization at the time the initial versions of the standards were developed.

This clause is not a full list of all environment variables, but only those of importance to multiple utilities. Nevertheless, to satisfy some members of the balloting group, here is a list of the other environment variable symbols mentioned in this standard:

Variable	Utility	Variable	Utility
CDPATH	`cd`	**MAKEFLAGS**	`make`
COLUMNS	`ls`	**OPTARG**	`getopts`
DEAD	`mailx`	**OPTIND**	`getopts`
IFS	`sh`	**PRINTER**	`lp`
LPDEST	`lp`	**PS 1**	`sh`
MAIL	`sh`	**PS 2**	`sh`
MAILRC	`mailx`		

The description of **PATH** is similar to that in POSIX.1 {8}, except:

- The behavior of a null prefix is marked obsolescent in favor of using a real pathname. This was done at the behest of some members of the balloting group, who apparently felt it offered a more secure environment, where the current directory would not be selected unintentionally.
- The POSIX.1 {8} *exec* description requires an implementation-defined path search when **PATH** is ''not present.'' POSIX.2 spells out that this means ''unset or set to null.'' Many implementations historically have used a default value of `/bin` and `/usr/bin`. POSIX.2 does not mandate that this default path be identical to that retrieved from `getconf _CS_PATH` because it is likely that a transition to POSIX.2 conformance will see the newly-standardized utilities in another directory that needs to be isolated from some historical applications.
- The POSIX.1 {8} **PATH** description is ambiguous about whether an ''executable file'' means one that has the appropriate permissions for the searching process to execute it. One reading would say that a file with any of the execution bits set on would satisfy the search and that an [EACCES] could be returned at that point. This is not the way historical systems work and POSIX.2 has clarified it to mean that the path search will continue until it finds the name with the execute permissions that would allow the process to execute it. (The case of the [ENOEXEC] error is handled in the text of 3.9.1.1.)

The terminology ''beginning to end'' is used in **PATH** to avoid the noninternationalized ''left to right.'' There is no way to have a colon character embedded within a pathname that is part of the **PATH** variable string. Colon is not a member of the portable filename character set, so this should not be a problem. A portable application can retrieve a default **PATH** value (that will allow access to all the standard utilities) from the system using the command:

```
getconf _CS_PATH
```

See the rationale with `command` for an example of using this.

The **SHELL** variable names the user's preferred shell; it is a guide to applications. There is no direct requirement that that shell conform to this standard—that decision should rest with the user. It is the intention of the developers of this standard that alternative shells be permitted, if the user chooses to develop or acquire one. An operating system that builds its shell into the ''kernel'' in such a manner that alternative shells would be impossible does not conform to the spirit of the standard.

The following environment variables are not currently used by the standard utilities (although they may be by future UPE utilities). Implementations should reserve the names for the following purposes:

EDITOR	The name of the user's preferred text file editor. The value of this variable is the name of a utility: either a pathname containing a slash, or a filename to be located using the **PATH** environment variable.
VISUAL	The name of the user's preferred ''visual,'' or full-screen, text file editor. The value of this variable is the name of a utility: either a pathname containing a slash, or a filename to be located using the **PATH** environment variable.

The decision to restrict conforming systems to the use of digits, uppercase letters, and underscores for environment variable names allows applications to use lowercase letters in their environment variable names without conflicting with any conforming system.

PROCLANG was added to an earlier draft for internationalized applications, but was removed from the standard because the working group determined that it was not of use.

USER was removed from an earlier draft because it was an unreasonable duplication of **LOGNAME** .

2.7 Required Files

The following directories shall exist on conforming systems and shall be used as described. Strictly Conforming POSIX.2 Applications shall not assume the ability to create files in any of these directories.

`/`	The root directory.
`/dev`	Contains `/dev/null` and `/dev/tty`, described below.

The following directory shall exist on conforming systems and shall be used as described.

```
%token          NUMBER                                                    2
%token          COLLSYMBOL COLLELEMENT                                    1
%token          CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR               1
%token          ELLIPSIS                                                  1
%token          EXTENDED0(3510)X 11 s 1396 3132(has)N 156[illegible]cted.)X 2194 MX  % 1
```

`/tmp`	A directory made available for programs that need a place to create temporary files. Applications shall be allowed to create files in this directory, but shall not assume that such files are preserved between invocations of the application.

The following files shall exist on conforming systems and shall be both readable and writable.

`/dev/null` An infinite data source/sink. Data written

`which requires that all the files in this clause be implemented via POSIX filesystems.)`

The `/tmp` directory is retained in the standard to accommodate historical applications that assume its availability. Future implementations are encouraged to provide suitable directory names in **TMPDIR** and future applications are encouraged