IEEE P1003.2 Draft 11.2 – September 1991

Copyright © 1991 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change.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]

2.5 Locale

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_TIME Date and time formats.

LC_NUMERIC Numeric, nonmonetary formatting.

LC_MONETARY Monetary formatting.

LC_MESSAGES Formats of informative and diagnostic messages and interactive responses.

Conforming implementations shall provide the standard utilities and the interfaces in Annex B (if that option is supported) with the capability to modify their behavior based on the current locale, as defined in the Environment Variables subclause for each utility and interface.

Locales other than those supplied by the implementation can be created via the `localedef` utility (see 4.35), provided that the {POSIX 2_LOCALEDEF } symbol is defined on the system; see 2.13.2. Otherwise, only the implementation-provided locale(s) can be used. The input to the utility is described in 2.5.2. The value that shall be 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 2.5.1.) When the value of a locale environment variable begins with a slash (`/`), it shall be interpreted as the pathname of the locale definition. If the value of the locale value does not begin with a slash, the mechanism used to locate the locale is implementation defined.

If different character sets are used by the locale categories, the results achieved by an application utilizing these categories is undefined. Likewise, if different code sets are used for the data being processed by interfaces whose behavior is dependent on the current locale, or the code set is different from the code set assumed when the locale was created, the result is also undefined.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.5.0.1 Locale Rationale. *(This subclause is not a part of P1003.2)*

The description of locales is based on work performed in the UniForum Technical Committee Subcommittee on Internationalization. Wherever appropriate, keywords were taken from the C Standard {7} or the *X/Open Portability Guide* {B31}.

The value that shall be used to specify a locale when using environment variables is the name specified as the *name* operand to the `localedef` utility when the locale was created. This provides a verifiable method to create and invoke a locale.

The ''object'' definitions need not be portable, as long as ''source'' definitions are. Strictly speaking, ''source'' definitions are portable only between implementations using the same character set(s). Such ''source'' definitions can, if they use symbolic names only, easily be ported between systems using different code sets as long as the characters in the portable character set (Table 2-3) have common values between the code sets; this is frequently the case in historical implementations. Of course, this requires that the symbolic names used for characters outside the portable character set are identical between character sets. The definition of symbolic names for characters is outside the scope of this standard, but is certainly within the scope of other standards organizations. When such names are standardized, future versions of POSIX.2 should require the use of these names.

Applications can select the desired locale by invoking the *setlocale* () function (or equivalent) with the appropriate value. If the function is invoked with an empty string, the value of the corresponding environment variable is used. If the environment variable is unset or is set to the empty string, the implementation sets the appropriate environment as defined in 2.6.

2.5.1 POSIX Locale

Conforming implementations shall provide a *POSIX Locale* . The behavior of standard utilities in the POSIX Locale shall be as if the locale was defined via the `localedef` utility with input data from Table 2-5, Table 2-7, Table 2-9, Table 2-10, Table 2-8, and Table 2-11, all in 2.5.2.

The tables describe the characteristics and behavior of the POSIX Locale for data consisting entirely of characters from the portable character set in Table 2-3 and the control characters in Table 2-4. For characters other than those in the two tables, the behavior is unspecified.

The POSIX Locale can be specified by assigning the appropriate environment variables the values `"C"` or `"POSIX"`.

Table 2-5 shows the definition for the LC_CTYPE category.

Table 2-7 shows the definition for the LC_COLLATE category.

Table 2-8 shows the definition for the LC_MONETARY category.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2-9 shows the definition for the LC_NUMERIC category.

Table 2-10 shows the definition for the LC_TIME category.

Table 2-11 shows the definition for the LC_MESSAGES category.

2.5.1.1 POSIX Locale Rationale. *(This subclause is not a part of P1003.2)*

The POSIX Locale is equal to the `"C"` locale, as specified in POSIX.1 {8}. To avoid being classified as a C-language function, the name has been changed to the *POSIX Locale*; the environment variable value can be either `"POSIX"`, or, for historical reasons, `"C"`.

The POSIX definitions mirror the historical UNIX system behavior.

The use of symbolic names for characters in the tables does not imply that the POSIX Locale must be described using symbolic character names, but merely that it may be advantageous to do so.

Implementations must define a locale as the ‘‘default’’ locale, to be invoked when no environment variables are set, or set to the empty string. This default locale can be the POSIX Locale or any other, implementation-defined locale. Some implementations may provide facilities for local installation administrators to set the default locale, customizing it for each location. This standard does not require such a facility.

2.5.2 Locale Definition

The capability to specify additional locales to those provided by an implementation is optional (see 2.13.2). If the option is not supported, only implementation-supplied locales are available. Such locales shall be documented using the format specified in this clause.

Locales can be described with the file format presented in this subclause. The file format is that accepted by the `localedef` utility (see 4.35). For the purposes of this subclause, the file is referred to as the *locale definition file*, but no locales shall be affected by this file unless it is processed by `localedef` or some similar mechanism. Any requirements in this subclause imposed upon ‘‘the utility’’ shall apply to `localedef` or to any other similar utility used to install locale information using the locale definition file format described here.

The locale definition file shall contain one or more locale category source definitions, and shall not contain more than one definition for the same locale category. If the file contains source definitions for more than one category, implementation-defined categories, if present, shall appear after the categories defined by this clause (2.5). A category source definition shall contain either the definition of a category or a `copy` directive. For a description of the `copy` directive, see 4.35. In the event that some of the information for a locale category, as specified in this standard, is missing from the locale source definition, the behavior of that category, if it is referenced, is unspecified.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A category source definition shall consist of a category header, a category body, and a category trailer. A category header shall consist of the character string naming of the category, beginning with the characters `LC_`. The category trailer shall consist of the string `END`, followed by one or more `<blank>`s and the string used in the corresponding category header.

The category body shall consist of one or more lines of text. Each line shall contain an identifier, optionally followed by one or more operands. Identifiers shall be either keywords, identifying a particular locale element, or collating elements. In addition to the keywords defined in this standard, the source can contain implementation-defined keywords. Each keyword within a locale shall have a unique name (i.e., two categories cannot have a commonly-named keyword); no keyword shall start with the characters `LC_`. Identifiers shall be separated from the operands by one or more `<blank>`s.

Operands shall be characters, collating elements, or strings of characters. Strings shall be enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the *<escape character>* , described below. When a keyword is followed by more than one operand, the operands shall be separated by semicolons; `<blank>`s shall be allowed before and/or after a semicolon.

The first category header in the file can be preceded by a line modifying the comment character. It shall have the following format, starting in column 1:

`"comment_char %c\n"`, *<comment character>*

The comment character shall default to the number-sign (#). Blank lines and lines containing the *<comment char>* in the first position shall be ignored.

The first category header in the file can be preceded by a line modifying the escape character to be used in the file. It shall have the following format, starting in column 1:

`"escape_char %c\n"`, *<escape character>*

The escape character shall default to backslash, which is the character used in all examples shown in this standard.

A line can be continued by placing an escape character as the last character on the line; this continuation character shall be discarded from the input. Although the implementation need not accept any one portion of a continued line with a length exceeding {LINE_MAX} bytes, it shall place no limits on the accumulated length of the continued line. Comment lines shall not be continued on a subsequent line using an escaped `<newline>`.

Individual characters, characters in strings, and collating elements shall be represented using symbolic names, as defined below. In addition, characters can be represented using the characters themselves, or as octal, hexadecimal, or decimal constants. When nonsymbolic notation is used, the resultant locale definitions need not be portable between systems. The left angle bracket (<) is a reserved symbol, denoting the start of a symbolic name; when used to represent itself it shall be preceded by the escape character. The following rules apply to character representation:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

(1) A character can be represented via a symbolic name, enclosed within angle brackets (< and >). The symbolic name, including the angle brackets, shall exactly match a symbolic name defined in the charmap file specified via the `localedef -f` option, and shall be replaced by a character value determined from the value associated with the symbolic name in the charmap file. The use of a symbolic name not found in the *charmap* file shall constitute an error, unless the category is LC_CTYPE or LC_COLLATE , in which case it shall constitute a warning condition (see `localedef` in 4.35 for a description of action resulting from errors and warnings). The specification of a symbolic name in a `collating-element` or `collating-symbol` clause that duplicates a symbolic name in the charmap file (if present) is an error. Use of the escape character or a right angle bracket within a symbolic name shall be invalid unless the character is preceded by the escape character.

Example : `<c>;<c-cedilla> "<M><a><y>"`

(2) A character can be represented by the character itself, in which case the value of the character is implementation defined. Within a string, the double-quote character, the escape character, and the right angle bracket character shall be escaped (preceded by the escape character) to be interpreted as the character itself. Outside strings, the characters

`,` `;` `<` `>` *escape_char*

shall be escaped to be interpreted as the character itself.

Example : `c β "May"`

(3) A character can be represented as an octal constant. An octal constant shall be specified as the escape character followed by two or more octal digits. Each constant shall represent a byte value. Multibyte characters can be represented by concatenated constants.

Example : `\143;\347;\143\150 "\115\141\171"`

(4) A character can be represented as a hexadecimal constant. A hexadecimal constant shall be specified as the escape character followed by an `x` followed by two or more hexadecimal digits. Each constant shall represent a byte value. Multibyte characters can be represented by concatenated constants.

Example : `\x63;\xe7;\x63\x68 "\x4d\x61\x79"`

(5) A character can be represented as a decimal constant. A decimal constant shall be specified as the escape character followed by a `d` followed by two or more decimal digits. Each constant shall represent a byte value. Multibyte values can be represented by concatenated constants.

Example : `\d99;\d231;\d99\d104 "\d77\d97\d121"`

Implementations may accept single-digit octal, decimal, or hexadecimal constants following the escape character. Only characters existing in the character set for which the locale definition is created shall be specified, whether using symbolic

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

names, the characters themselves, or octal, decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not present in the charmap file can be specified and shall be ignored, as specified under item (1) above.

2.5.2.0.1 Locale Definition Rationale. *(This subclause is not a part of P1003.2)*

The decision to separate the file format from the `localedef` utility description was only partially editorial. Implementations may provide other interfaces than `localedef`. Requirements on ''the utility,'' mostly concerning error messages, are described in this way because they are meant to affect the other interfaces implementations may provide as well as `localedef`. (This is similar to the philosophy used by POSIX.1 {8} where the descriptions of the `tar` and `cpio` file formats impose requirements on any utilities processing them.)

The text about {POSIX 2_LOCALEDEF} does not mean that internationalization is optional; only that the functionality of the `localedef` utility is. Regular expressions, for instance, must still be able to recognize e.g., character class expressions such as `[[:alpha:]]`.

A possible analogy is with an applications development environment: while all conforming implementations must be capable of executing applications, not all need to have the development environment installed. The assumption is that the capability to modify the behavior of utilities (and applications) via locale settings must be supported. If the `localedef` utility is not present, then the only choice is to select an existing (presumably implementation-documented) locale. An implementation could, for example, chose to support only the POSIX Locale, which would in effect limit the amount of changes from historical implementations quite drastically. The `localedef` utility is still required, but would always terminate with an exit code indicating that no locale could be created. Supported locales must be documented using the syntax defined in 2.5. (This ensures that users can accurately determine what capabilities are provided. If the implementation decides to provide additional capabilities to the ones in 2.5, that is already provided for.)

If the option is present (i.e., locales can be created), then the `localedef` utility must be capable of creating locales based on the syntax and rules defined in 2.5. This does not mean that the implementation cannot also provide alternate means for creating locales.

The octal, decimal, and hexadecimal notations are the same employed by the charmap facility (see 2.4.1). To avoid confusion between an octal constant and a backreference, the octal, hexadecimal, and decimal constants must contain at least two digits. As single-digit constants are relatively rare, this should not impose any significant hardship. Each of the constants includes ''two or more'' digits to account for systems in which the byte size is larger than eight bits. For example, a Unicode system that has defined 16-bit bytes may require six octal, four hexadecimal, and five decimal digits.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

This standard is intended as an international (ISO/IEC) standard as well as an IEEE standard, and must therefore follow the ISO/IEC guidelines. One such rule is that characters outside the invariant part of ISO/IEC 646 {1} should not be used in portable specifications. The backslash character is not in the invariant part; the number-sign is, but with multiple representations: as a number-sign and as a pound sign. As far as general usage of these symbols, they are covered by the ''grandfather clause,'' but for newly defined interfaces, ISO has requested that POSIX provides alternate representations. Consequently, while the default escape character remains the backslash, and the default comment character is the number-sign, implementations are required to recognize alternative representations, identified in the applicable source file via the `escape_char` and `comment_char` keywords.

2.5.2.1 LC_CTYPE

The LC_CTYPE category shall define character classification, case conversion, and other character attributes. In addition, a series of characters can be represented by three adjacent periods representing an ellipsis symbol (''. . .''). The ellipsis specification shall be interpreted as meaning that all values between the values preceding and following it represent valid characters. The ellipsis specification only shall be valid within a single encoded character set. An ellipsis shall be interpreted as including in the list all characters with an encoded value higher than the encoded value of the character preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

Example : `\x30;...;\x39;` includes in the character class all characters with encoded values between the endpoints.

The following keywords shall be recognized. In the descriptions, the term ''automatically included'' means that it shall not be an error to either include the referenced characters or to omit them; the implementation shall provide them if missing and accept them silently if present.

`copy` Specify the name of an existing locale to be used as the source for the definition of this category. If this keyword is specified, no other keyword shall be specified.

`upper` Define characters to be classified as uppercase letters. No character specified for the keywords `cntrl`, `digit`, `punct`, or `space` shall be specified. If this keyword is not specified, the uppercase letters `A` through `Z`, as defined in Table 2-3 (see 2.4.1), shall automatically belong to this class, with implementation-defined character values.

`lower` Define characters to be classified as lowercase letters. No character specified for the keywords `cntrl`, `digit`, `punct`, or `space` shall be specified. If this keyword is not specified, the lowercase letters `a` through `z`, as defined in Table 2-3 (see 2.4.1), shall automatically belong to this class, with implementation-defined character values.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2-5 – LC_CTYPE Category Definition in the POSIX Locale

```
LC_CTYPE
# The following is the POSIX Locale LC_CTYPE.
# "alpha" is by default "upper" and "lower"
# "alnum" is by definition "alpha" and "digit"
# "print" is by default "alnum", "punct" and the <space> character
# "graph" is by default "alnum" and "punct"
#
upper   <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>
#
lower   <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>
#
digit   <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;<eight>;<nine>
#
space   <tab>;<newline>;<vertical-tab>;<form-feed>;<carriage-return>;<space>
#
cntrl   <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
        <form-feed>;<carriage-return>;\
        <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
        <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
        <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
        <IS1>;<DEL>
#
punct   <exclamation-mark>;<quotation-mark>;<number-sign>;\
        <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
        <left-parenthesis>;<right-parenthesis>;<asterisk>;\
        <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
        <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
        <greater-than-sign>;<question-mark>;<commercial-at>
        <left-square-bracket>;<backslash>;<right-square-bracket>;\
        <circumflex>;<underline>;<grave-accent>;\
        <left-curly-bracket>;<vertical-line>;<right-curly-bracket>;<tilde>
#
xdigit  <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;<eight>;\
        <nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
#
blank   <space>;<tab>
#
toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
        (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
        (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
        (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
        (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
#
tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
        (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
        (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
        (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
        (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
END LC_CTYPE
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1810 `alpha` Define characters to be classified as letters. No character
1811 specified for the keywords `cntrl`, `digit`, `punct`, or `space` shall
1812 be specified. In addition, characters classified as either `upper`
1813 or `lower` shall automatically belong to this class.

1814 `digit` Define the characters to be classified as numeric digits. Only the 2
1815 digits `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, and `9` shall be specified, and in 2
1816 ascending sequence by numerical value. If this keyword is not 2
1817 specified, the digits `0` through `9`, as defined in Table 2-3 (see 2
1818 2.4.1), shall automatically belong to this class, with 2
1819 implementation-defined character values. 2

1820 `space` Define characters to be classified as white-space characters. No
1821 character specified for the keywords `upper`, `lower`, `alpha`,
1822 `digit`, `graph`, or `xdigit` shall be specified. If this keyword is 2
1823 not specified, the characters `<space>`, `<form-feed>`, `<new-` 2
1824 `line>`, `<carriage-return>`, `<tab>`, and `<vertical-tab>`, as 2
1825 defined in Table 2-3 (see 2.4.1), shall automatically belong to 2
1826 this class, with implementation-defined character values. Any 2
1827 characters included in the class `blank` shall be automatically 1
1828 included. 1

1829 `cntrl` Define characters to be classified as control characters. No char-
1830 acter specified for the keywords `upper`, `lower`, `alpha`, `digit`,
1831 `punct`, `graph`, `print`, or `xdigit` shall be specified. 1

1832 `punct` Define characters to be classified as punctuation characters. No
1833 character specified for the keywords `upper`, `lower`, `alpha`,
1834 `digit`, `cntrl`, `xdigit`, or as the `<space>` character shall be
1835 specified.

1836 `graph` Define characters to be classified as printable characters, not
1837 including the `<space>` character. If this keyword is not
1838 specified, characters specified for the keywords `upper`, `lower`,
1839 `alpha`, `digit`, `xdigit`, and `punct` shall belong to this character
1840 class. No character specified for the keyword `cntrl` shall be
1841 specified.

1842 `print` Define characters to be classified as printable characters, includ-
1843 ing the `<space>` character. If this keyword is not provided,
1844 characters specified for the keywords `upper`, `lower`, `alpha`,
1845 `digit`, `xdigit`, `punct`, and the `<space>` character shall belong
1846 to this character class. No character specified for the keyword
1847 `cntrl` shall be specified.

1848 `xdigit` Define the characters to be classified as hexadecimal digits. 2
1849 Only the characters defined for the class `digit` shall be 2
1850 specified, in ascending sequence by numerical value, followed by 2
1851 one or more sets of six characters representing the hexadecimal 2
1852 digits 10 through 15, with each set in ascending order (for exam- 2
1853 ple `A`, `B`, `C`, `D`, `E`, `F`, `a`, `b`, `c`, `d`, `e`, `f`). If this keyword is not 2
1854 specified, the digits `0` through `9`, the uppercase letters `A` through 2

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F, and the lowercase letters a through f, as defined in Table 2-3 (see 2.4.1), shall automatically belong to this class, with implementation-defined character values.

`blank` Define characters to be classified as <blank> characters. If this keyword is unspecified, the characters <space> and <tab> shall belong to this character class.

`toupper` Define the mapping of lowercase letters to uppercase letters. The operand shall consist of character pairs, separated by semicolons. The characters in each character pair shall be separated by a comma and the pair enclosed by parentheses. The first character in each pair shall be the lowercase letter, the second the corresponding uppercase letter. Only characters specified for the keywords `lower` and `upper` shall be specified. If this keyword is not specified, the lowercase letters a through z, and their corresponding uppercase letters A through Z, as defined in Table 2-3 (see 2.4.1), shall automatically be included, with implementation-defined character values.

`tolower` Define the mapping of uppercase letters to lowercase letters. The operand shall consist of character pairs, separated by semicolons. The characters in each character pair are separated by a comma and the pair enclosed by parentheses. The first character in each pair shall be the uppercase letter, the second the corresponding lowercase letter. Only characters specified for the keywords `lower` and `upper` shall be specified.

The `tolower` keyword is optional. If specified, the uppercase letters A through Z, as defined in Table 2-3, and their corresponding lowercase letter, shall be specified. If this keyword is not specified, the mapping shall be the reverse mapping of the one specified for `toupper`.

Table 2-6 shows the allowed character class combinations.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2-6 – Valid Character Class Combinations

In Class	Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
upper	-	-	M	X	X	X	X	D	D	-	X
lower	-	-	M	X	X	X	X	D	D	-	X
alpha	-	-	-	X	X	X	X	D	D	-	X
digit	X	X	X	-	X	X	X	D	D	-	X
space	X	X	X	X	-	-	*	*	*	X	-
cntrl	X	X	X	X	-	-	X	X	X	X	-
punct	X	X	X	X	-	X	-	D	D	X	-
graph	-	-	-	-	-	X	-	-	-	-	-
print	-	-	-	-	-	X	-	-	-	-	-
xdigit	-	-	-	-	X	X	X	D	D	-	X
blank	X	X	X	X	M	-	*	*	*	X	-

NOTES:

(1) Explanation of codes:

M Always

D Default; belongs to class if not specified

\- Permitted

X Mutually exclusive

* See note (2)

(2) The <space> character, which is part of the `space` and `blank` classes, cannot belong to `punct` or `graph`, but automatically shall belong to the `print` class. Other `space` or `blank` characters can be classified as `punct`, `graph`, and/or `print`.

2.5.2.1.1 LC_CTYPE Rationale. *(This subclause is not a part of P1003.2)*

The LC_CTYPE category primarily is used to define the encoding-independent aspects of a character set, such as character classification. In addition, certain encoding-dependent characteristics are also defined for an application via the LC_CTYPE category. POSIX.2 does not mandate that the encoding used in the locale is the same as the one used by the application, because an implementation may decide that it is advantageous to define locales in a system-wide encoding rather than having multiple, logically identical locales in different encodings, and to convert from the application encoding to the system-wide encoding on usage. Other implementations could require encoding-dependent locales.

In either case, the LC_CTYPE attributes that are directly dependent on the encoding, such as `mb_cur_max` and the display width of characters, are not user-specifiable in a locale source, and are consequently not defined as keywords.

As the LC_CTYPE character classes are based on the C Standard {7} character-class definition, the category does not support multicharacter elements. For

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

instance, the German character <sharp-s> is traditionally classified as a lowercase letter. There is no corresponding uppercase letter; in proper capitalization of German text the <sharp-s> will be replaced by SS; i.e., by two characters. This kind of conversion is outside the scope of the `toupper` and `tolower` keywords.

Where POSIX. 2 specifies that only certain characters can be specified, as for the keywords `digit` and `xdigit`, the specified characters must be from the portable character set, as shown. As an example, only the Arabic digits 0 through 9 are acceptable as digits.

The character classes `digit`, `xdigit`, `lower`, `upper`, and `space` have a set of automatically included characters. These only need to be specified if the character values (i.e., encoding) differs from the implementation default values.

The definition of character class `digit` requires that only ten characters—the ones defining digits—can be specified; alternate digits (e.g., Hindi or Kanji) cannot be specified here. However, the encoding may vary if an implementation supports more than one encoding.

The definition of character class `xdigit` requires that the characters included in character class digit are included here also, and allows for different symbols for the hexadecimal digits 10 through 15.

2.5.2.2 LC_COLLATE

A collation sequence definition shall define the relative order between collating elements (characters and multicharacter collating elements) in the locale. This order is expressed in terms of collation values; i.e., by assigning each element one or more collation values (also known as collation weights). This does not imply that implementations shall assign such values, but that ordering of strings using the resultant collation definition in the locale shall behave as if such assignment is done and used in the collation process. The collation sequence definition shall be used by regular expressions, pattern matching, and sorting. The following capabilities are provided:

(1) **Multicharacter collating elements** . Specification of multicharacter collating elements (i.e., sequences of two or more characters to be collated as an entity).

(2) **User-defined ordering of collating elements** . Each collating element shall be assigned a collation value defining its order in the character (or basic) collation sequence. This ordering is used by regular expressions and pattern matching and, unless collation weights are explicitly specified, also as the collation weight to be used in sorting.

(3) **Multiple weights and equivalence classes** . Collating elements can be assigned one or more (up to the limit {COLL_WEIGHTS_MAX}) collating weights for use in sorting. The first weight is hereafter referred to as the primary weight.

(4) **One-to-Many mapping** . A single character is mapped into a string of collating elements.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

(5) **Many-to-Many substitution** . A string of one or more characters is substituted by another string (or an empty string, i.e., the character or characters shall be ignored for collation purposes).

(6) **Equivalence class definition** . Two or more collating elements have the same collation value (primary weight).

(7) **Ordering by weights** . When two strings are compared to determine their relative order, the two strings are first broken up into a series of collating elements, and each successive pair of elements are compared according to the relative primary weights for the elements. If equal, and more than one weight has been assigned, then the pairs of collating elements are recompared according to the relative subsequent weights, until either a pair of collating elements compare unequal or the weights are exhausted.

The following keywords shall be recognized in a collation sequence definition. They are described in detail in the following subclauses.

`copy`	Specify the name of an existing locale to be used as the source for the definition of this category. If this keyword is specified, no other keyword shall be specified.
`collating-element`	Define a collating-element symbol representing a multicharacter collating element. This keyword is optional.
`collating-symbol`	Define a collating symbol for use in collation order statements. This keyword is optional.
`order_start`	Define collation rules. This statement is followed by one or more collation order statements, assigning character collation values and collation weights to collating elements.
`order_end`	Specify the end of the collation-order statements.

2.5.2.2.1 `collating-element` Keyword

In addition to the collating elements in the character set, the `collating-element` keyword shall be used to define multicharacter collating elements. The syntax is

"`collating-element %s from %s\n`", *<collating-symbol>* , *<string>*

The *<collating-symbol>* operand shall be a symbolic name, enclosed between angle brackets (< and >), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand shall be a string of two or more characters that shall collate as an entity. A *<collating-element>* defined via this keyword is only recognized with the LC_COLLATE category.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2-7 – LC_COLLATE Category Definition in the POSIX Locale

```
LC_COLLATE
# This is the POSIX Locale definition for the LC_COLLATE category.
# The order is the same as in the ASCII code set.
order_start forward
<NUL>
<SOH>
<STX>
<ETX>
<EOT>
<ENQ>
<ACK>
<alert>
<backspace>
<tab>
<newline>
<vertical-tab>
<form-feed>
<carriage-return>
<SO>
<SI>
<DLE>
<DC1>
<DC2>
<DC3>
<DC4>
<NAK>
<SYN>
<ETB>
<CAN>
<EM>
<SUB>
<ESC>
<IS4>
<IS3>
<IS2>
<IS1>
<space>
<exclamation-mark>
<quotation-mark>
<number-sign>
<dollar-sign>
<percent-sign>
<ampersand>
<apostrophe>
<left-parenthesis>
<right-parenthesis>
<asterisk>
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2-7 – LC_COLLATE Category Definition in the POSIX Locale (*continued*)

```
<plus-sign>
<comma>
<hyphen>
<period>
<slash>
<zero>
<one>
<two>
<three>
<four>
<five>
<six>
<seven>
<eight>
<nine>
<colon>
<semicolon>
<less-than-sign>
<equals-sign>
<greater-than-sign>
<question-mark>
<commercial-at>
<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>
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2-7 – LC_COLLATE Category Definition in the POSIX Locale (*concluded*)

```
<left-square-bracket>
<backslash>
<right-square-bracket>
<circumflex>
<underline>
<grave-accent>
<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>
<left-curly-bracket>
<vertical-line>
<right-curly-bracket>
<tilde>
<DEL>
order_end
#
END LC_COLLATE
```

Example:

```
     collating-element <ch> from <c><h>
     collating-element <e-acute> from <acute><e>
     collating-element <ll> from ll
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.5.2.2.2 `collating-symbol` Keyword

This keyword shall be used to define symbols for use in collation sequence statements; i.e., between the `order_start` and the `order_end` keywords. The syntax is

"`collating-symbol %s\n`", *<collating-symbol>*

The *<collating-symbol>* shall be a symbolic name, enclosed between angle brackets (< and >), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. A *<collating-symbol>* defined via this keyword is only recognized with the LC_COLLATE category.

Example:

```
collating-symbol <UPPER_CASE>
collating-symbol <HIGH>
```

2.5.2.2.3 `order_start` Keyword

The `order_start` keyword shall precede collation order entries and also defines the number of weights for this collation sequence definition and other collation rules.

The syntax of the `order_start` keyword is:

"`order_start %s;%s;...;%s\n`", *<sort-rules>* , *<sort-rules>* ...

The operands to the `order_start` keyword are optional. If present, the operands define rules to be applied when strings are compared. The number of operands define how many weights each element is assigned; if no operands are present, one `forward` operand is assumed. If present, the first operand defines rules to be applied when comparing strings using the first (primary) weight; the second when comparing strings using the second weight, and so on. Operands shall be separated by semicolons (`;`). Each operand shall consist of one or more collation directives, separated by commas (`,`). If the number or operands exceeds the {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following directives shall be supported:

`forward` Specifies that comparison operations for the weight level shall proceed from start of string towards the end of string.

`backward` Specifies that comparison operations for the weight level shall proceed from end of string towards the beginning of string.

`position` Specifies that comparison operations for the weight level will consider the relative position of non-`IGNORE`d elements in the strings. The string containing a non-

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

`IGNORE`d element after the fewest `IGNORE`d collating elements from the start of the compare shall collate first. If both strings contain a non-`IGNORE`d character in the same relative position, the collating values assigned to the elements shall determine the ordering. In case of equality, subsequent non-`IGNORE`d characters shall be considered in the same manner.

The directives `forward` and `backward` are mutually exclusive.

Example :

```
order_start     forward;backward
```

If no operands are specified, a single `forward` operand shall be assumed.

2.5.2.2.4 Collation Order

The `order_start` keyword shall be followed by collating element entries. The syntax for the collating element entries is

`"%s %s;%s;...;%s\n"`, <*collating-element*> , <*weight*> , <*weight*> , ...

Each *collating-element* shall consist of either a character (in any of the forms defined in 2.5.2), a <*collating-element*> , a <*collating-symbol*> , an ellipsis, or the special symbol `UNDEFINED`. The order in which collating elements are specified determines the character collation sequence, such that each collating element shall compare less than the elements following it. The NUL character shall compare lower than any other character.

A <*collating-element*> shall be used to specify multicharacter collating elements, and indicates that the character sequence specified via the <*collating-element*> is to be collated as a unit and in the relative order specified by its place.

A <*collating-symbol*> shall be used to define a position in the relative order for use in weights.

The ellipsis symbol (‘‘`...`’’) specifies that a sequence of characters shall collate according to their encoded character values. It shall be interpreted as indicating that all characters with a coded character set value higher than the value of the character in the preceding line, and lower than the coded character set value for the character in the following line, in the current coded character set, shall be placed in the character collation order between the previous and the following character in ascending order according to their coded character set values. An initial ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing ellipsis as if the following line specified the highest coded character set value in the current coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do not specify characters in the current coded character set. The use of the ellipsis symbol ties the definition to a specific coded character set and may preclude the definition from being portable between implementations.

The symbol `UNDEFINED` shall be interpreted as including all coded character set values not specified explicitly or via the ellipsis symbol. Such characters shall be

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

inserted in the character collation order at the point indicated by the symbol, and in ascending order according to their coded character set values. If no UNDEFINED symbol is specified, and the current coded character set contains characters not specified in this clause, the utility shall issue a warning message and place such characters at the end of the character collation order.

The optional operands for each collation-element shall be used to define the primary, secondary, or subsequent weights for the collating element. The first operand specifies the relative primary weight, the second the relative secondary weight, and so on. Two or more collation-elements can be assigned the same weight; they belong to the same *equivalence class* if they have the same primary weight. Collation shall behave as if, for each weight level, IGNOREd elements are removed. Then each successive pair of elements shall be compared according to the relative weights for the elements. If the two strings compare equal, the process shall be repeated for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

Weights shall be expressed as characters (in any of the forms specified in 2.5.2), <*collating-symbol*> s, <*collating-element*> s, an ellipsis, or the special symbol IGNORE. A single character, a <*collating-symbol*>, or a <*collating-element*> shall represent the relative order in the character collating sequence of the character or symbol, rather than the character or characters themselves.

One-to-many mapping is indicated by specifying two or more concatenated characters or symbolic names. Thus, if the character "<eszet>" is given the string <s><s> as a weight, comparisons shall be performed as if all occurrences of the character <eszet> are replaced by <s><s>. If it is desirable to define <eszet> and <s><s> as an equivalence class, then a collating-element must be defined for the string "ss", as in the example below.

All characters specified via an ellipsis shall by default be assigned unique weights, equal to the relative order of characters. Characters specified via an explicit or implicit UNDEFINED special symbol shall by default be assigned the same primary weight (i.e., belong to the same equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each character in the sequence shall have unique weights, equal to the relative order of their character in the character collation sequence. Secondary and subsequent weights have unique values. The use of the ellipsis as a weight shall be treated as an error if the collating element is neither an ellipsis nor the special symbol UNDEFINED.

The special keyword IGNORE as a weight shall indicate that when strings are compared using the weights at the level where IGNORE is specified, the collating element shall be ignored; i.e., as if the string did not contain the collating element. In regular expressions and pattern matching, all characters that are IGNOREd in their primary weight form an equivalence class.

An empty operand shall be interpreted as the collating-element itself.

For example, the order statement

```
<a>      <a>;<a>
```

is equal to

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2279 `<a>`

2280 An ellipsis can be used as an operand if the collating-element was an ellipsis, and
2281 shall be interpreted as the value of each character defined by the ellipsis.

2282 The collation order as defined in this clause defines the interpretation of bracket 1
2283 expressions in regular expressions (see 2.8.3.2). 1

2284 *Example* :

```
2285 order_start   forward;backward
2286 UNDEFINED     IGNORE;IGNORE
2287 <LOW>
2288 <space>       <LOW>;<space>
2289 ...           <LOW>;...
2290 <a>           <a>;<a>
2291 <a-acute>     <a>;<a-acute>
2292 <a-grave>     <a>;<a-grave>
2293 <A>           <a>;<A>
2294 <A-acute>     <a>;<A-acute>
2295 <A-grave>     <a>;<A-grave>
2296 <ch>          <ch>;<ch>
2297 <Ch>          <ch>;<Ch>
2298 <s>           <s>;<s>
                                                    2
2299 <eszet>       <s><s>;<eszet><eszet>
2300 ...           <HIGH>;...
2301 <HIGH>
2302 order_end
```

2303 This example is interpreted as follows:

2304 (1) The `UNDEFINED` means that all characters not specified in this definition
2305 (explicitly or via the ellipsis) shall be ignored for collation purposes; for
2306 regular expression purposes they are ordered first.

2307 (2) All characters between `<space>` and `<a>` shall have the same primary
2308 equivalence class and individual secondary weights based on their ordi-
2309 nal encoded values.

2310 (3) All characters based on the upper- or lowercase character `a` belong to the
2311 same primary equivalence class.

2312 (4) The multicharacter collating element `<c><h>` is represented by the col-
2313 lating symbol `<ch>` and belongs to the same primary equivalence class as
2314 the multicharacter collating element `<C><h>`.

2315 (5) Note that it is not possible to use the collating element `<ss>` as a weight 1
2316 and expect it to be expanded to the string ‘‘ss’’. When used as a weight, 1
2317 any collating-element represents the relative order assigned to it in the 1
2318 character collation sequence, not the string from which it was derived 1
2319 (compare with `<ch>`). 1

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.5.2.2.5 `order_end` Keyword

The collating order entries shall be terminated with an `order_end` keyword.

2.5.2.2.6 LC_COLLATE Rationale. *(This subclause is not a part of P1003.2)*

The LC_COLLATE category governs the collation order in the locale, and thus the processing of the C Standard {7} *strxfrm* () and *strcoll* () functions, as well as a number of POSIX. 2 utilities.

The rules governing collation depends to some extent on the use. At least five different levels of increasingly complex collation rules can be distinguished:

(1) Byte/machine code order. This is the historical collation order in the UNIX system and many proprietary operating systems. Collation is here done character by character, without any regard to context. The primary virtue is that it usually is quite fast, and also completely deterministic; it works well when the native machine collation sequence matches the user expectations.

(2) Character order. On this level, collation is also done character by character, without regard to context. The order between characters is, however, not determined by the code values, but on the user's expectations of the ''correct'' order between characters. In addition, such a (simple) collation order can specify that certain characters collate equal (e.g., upper- and lowercase letters).

(3) String ordering. On this level, entire strings are compared based on relatively straightforward rules. At this level, several ''passes'' may be required to determine the order between two strings. Characters may be ignored in some passes, but not in others; the strings may be compared in different directions; and simple string substitutions may be made before strings are compared. This level is best described as ''dictionary'' ordering; it is based on the spelling, not the pronunciation, or meaning, of the words.

(4) Text search ordering. This is a further refinement of the previous level, best described as ''telephone book ordering''; some common homonyms (words spelled differently but with same pronunciation) are collated together; numbers are collated as if spelled with words, and so on.

(5) Semantic level ordering. Words and strings are collated based on their meaning; entire words (such as ''the'') are eliminated, the ordering is not deterministic. This usually requires special software, and is highly dependent on the intended use.

While the historical collation order formally is at level 1, for the English language it corresponds roughly to elements at level 2. The user expects to see the output from the `ls` utility sorted very much as as it would be in a dictionary. While telephone book ordering would be an optimal goal for standard collation, this was ruled out as the order would be language dependent. Furthermore, a requirement was that the order must be determined solely from the text string and the

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

collation rules; no external information (e.g., ‘‘pronunciation dictionaries’’) could be required.

As a result, the goal for the collation support is at level 3. This also matches the requirements for the proposed Canadian collation order, as well as other, known collation requirements for alphabetic scripts. It specifically rules out collation based on pronunciation rules, or based on semantic analysis of the text.

The syntax for the LC_COLLATE category source is the result of a cooperative effort between representatives for many countries and organizations working with international issues, such as UniForum, X/Open, and ISO, and it meets the requirements for level 3, and has been verified to produce the correct result with examples based on French, Canadian, and Danish collation order, as well as meeting the requirements in the X/Open Portability Guide, Issue 3. {B31}. Because it supports multicharacter collating elements, it is also capable of supporting collation in code sets where a character is expressed using nonspacing characters followed by the base character (such as ISO 6937 {B6}).

The directives that can be specified in an operand to the `order_start` keyword are based on the requirements specified in several proposed standards and in customary use. The following is a rephrasing of rules defined for ‘‘lexical ordering in English and French’’ by the Canadian Standards Association (text is brackets is rephrased):

(1) Once special characters ([punctuation]) have been removed from original strings, the ordering is determined by scanning forward (left to right) [disregarding case and diacriticals].

(2) In case of equivalence, special characters are once again removed from original strings and the ordering is determined scanning backward (starting from the rightmost character of the string and back), character by character, [disregarding case but considering diacriticals].

(3) In case of repeated equivalence, special characters are removed again from original strings and the ordering is determined scanning forward, character by character, [considering both case and diacriticals].

(4) If there is still an ordering equivalence after rules (1) through (3) have been applied, then only special characters and the position they occupy in the string are considered to determine ordering. The string that has a special character in the lowest position comes first. If two strings have a special character in the same position, the character [with the lowest collation value] comes first. In case of equality, the other special characters are considered until there is a difference or all special characters have been exhausted.

It is estimated that the standard covers the requirements for all European languages, and no particular problems are anticipated with Slavic or Middle East character sets.

The Far East (particularly Japanese/Chinese) collations are often based on contextual information and pronunciation rules (the same ideogram can have different meanings and different pronunciations). Such collation, in general, falls

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

outside the desired goal of the standard. There are, however, several other collation rules (stroke/radical, or ‘‘most common pronunciation’’) which can be supported with the mechanism described here.

Previous drafts contained a `substitute` statement, which performed a regular expression style replacement before string compares. It has been withdrawn based on balloter objections that it was not required for the types of ordering POSIX. 2 is aimed at.

The character (and collating element) order is defined by the order in which characters and elements are specified between the `order_start` and `order_end` keywords. This character order is used in range expressions in regular expressions (see 2.8). Weights assigned to the characters and elements defines the collation sequence; in the absence of weights, the character order is also the collation sequence.

The `position` keyword was introduced to provide the capability to consider, in a compare, the relative position of non-`IGNORE`d characters. As an example, consider the two strings ‘‘o-ring’’ and ‘‘or-ing’’. Assuming the hyphen is `IGNORE`d on the first pass, the two strings will compare equal, and the position of the hyphen is immaterial. On second pass, all characters except the hyphen are `IGNORE`d, and in the normal case the two strings would again compare equal. By taking position into account, the first collates before the second.

2.5.2.3 LC_MONETARY

The LC_MONETARY category shall define the rules and symbols that shall be used to format monetary numeric information. The operands are strings. For some keywords, the strings can contain only integers. Keywords that are not provided, string values set to the empty string (`""`), or integer keywords set to −1, shall be used to indicate that the value is unspecified. The following keywords shall be recognized:

`copy` Specify the name of an existing locale to be used as the source for the definition of this category. If this keyword is specified, no other keyword shall be specified.

`int_curr_symbol` The international currency symbol. The operand shall be a four-character string, with the first three characters containing the alphabetic international currency symbol in accordance with those specified in ISO 4217 {3} (*Codes for the representation of currencies and funds*). The fourth character shall be the character used to separate the international currency symbol from the monetary quantity.

`currency_symbol` The string that shall be used as the local currency symbol.

`mon_decimal_point` The operand is a string containing the symbol that shall be used as the decimal delimiter in monetary formatted quantities. In contexts where other standards

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2-8 – LC_MONETARY Category Definition in the POSIX Locale

```
LC_MONETARY
# This is the POSIX Locale definition for
# the LC_MONETARY category.
#
int_curr_symbol     ""
currency_symbol     ""
mon_decimal_point   ""
mon_thousands_sep   ""
mon_grouping        ""
positive_sign       ""
negative_sign       ""
int_frac_digits     -1
p_cs_precedes       -1
p_sep_by_space      -1
n_cs_precedes       -1
n_sep_by_space      -1
p_sign_posn         -1
n_sign_posn         -1
#
END LC_MONETARY
```

limit the `mon_decimal_point` to a single byte, the result of specifying a multibyte operand is unspecified.

`mon_thousands_sep` The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in formatted monetary quantities. In contexts where other standards limit the `mon_thousands_sep` to a single byte, the result of specifying a multibyte operand is unspecified.

`mon_grouping` Define the size of each group of digits in formatted monetary quantities. The operand is a sequence of integers separated by semicolons. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not −1, then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is −1, then no further grouping shall be performed.

`positive_sign` A string that shall be used to indicate a nonnegative-valued formatted monetary quantity.

`negative_sign` A string that shall be used to indicate a negative-valued formatted monetary quantity.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2495 `int_frac_digits` An integer representing the number of fractional digits
2496 (those to the right of the decimal delimiter) to be writ-
2497 ten in a formatted monetary quantity using
2498 `int_curr_symbol`.

2499 `frac_digits` An integer representing the number of fractional digits
2500 (those to the right of the decimal delimiter) to be writ-
2501 ten in a formatted monetary quantity using
2502 `currency_symbol`.

2503 `p_cs_precedes` An integer set to 1 if the `currency_symbol` or
2504 `int_curr_symbol` precedes the value for a nonnega-
2505 tive formatted monetary quantity, and set to 0 if the
2506 symbol succeeds the value.

2507 `p_sep_by_space` An integer set to 0 if no space separates the
2508 `currency_symbol` or `int_curr_symbol` from the
2509 value for a nonnegative formatted monetary quantity,
2510 set to 1 if a space separates the symbol from the value,
2511 and set to 2 if a space separates the symbol and the
2512 sign string, if adjacent.

2513 `n_cs_precedes` An integer set to 1 if the `currency_symbol` or
2514 `int_curr_symbol` precedes the value for a negative
2515 formatted monetary quantity, and set to 0 if the sym-
2516 bol succeeds the value.

2517 `n_sep_by_space` An integer set to 0 if no space separates the
2518 `currency_symbol` or `int_curr_symbol` from the
2519 value for a negative formatted monetary quantity, set
2520 to 1 if a space separates the symbol from the value,
2521 and set to 2 if a space separates the symbol and the
2522 sign string, if adjacent.

2523 `p_sign_posn` An integer set to a value indicating the positioning of
2524 the `positive_sign` for a nonnegative formatted
2525 monetary quantity. The following integer values shall
2526 be recognized:

2527 `0` Parentheses enclose the quantity and the
2528 `currency_symbol` or `int_curr_symbol`.

2529 `1` The sign string precedes the quantity and the
2530 `currency_symbol` or `int_curr_symbol`.

2531 `2` The sign string succeeds the quantity and the
2532 `currency_symbol` or `int_curr_symbol`.

2533 `3` The sign string immediately precedes the
2534 `currency_symbol` or `int_curr_symbol`.

2535 `4` The sign string immediately succeeds the
2536 `currency_symbol` or `int_curr_symbol`.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

`n_sign_posn` An integer set to a value indicating the positioning of the `negative_sign` for a negative formatted monetary quantity. The following integer values shall be recognized:

- `0` Parentheses enclose the quantity and the `currency_symbol` or `int_curr_symbol`.
- `1` The sign string precedes the quantity and the `currency_symbol` or `int_curr_symbol`.
- `2` The sign string succeeds the quantity and the `currency_symbol` or `int_curr_symbol`.
- `3` The sign string immediately precedes the `currency_symbol` or `int_curr_symbol`.
- `4` The sign string immediately succeeds the `currency_symbol` or `int_curr_symbol`.

2.5.2.3.1 LC_MONETARY Rationale. *(This subclause is not a part of P1003.2)*

The currency symbol does not appear in LC_MONETARY because it is not defined in the C Standard's {7}C locale.

The C Standard {7} limits the size of decimal points and thousands delimiters to single-byte values. In locales based on multibyte coded character sets this cannot be enforced, obviously; this standard does not prohibit such characters, but makes the behavior unspecified [in the text ''In contexts where other standards . . . ''].

The grouping specification is based on, but not identical to, the C Standard {7}. The ''–1'' signals that no further grouping shall be performed, the equivalent of {CHAR_MAX} in the C Standard {7}).

The locale definition is an extension of the C Standard {7} *localeconv* () specification. In particular, rules on how `currency_symbol` is treated are extended to also cover `int_curr_symbol`, and `p_set_by_space` and `n_sep_by_space` have been augmented with the value 2, which places a space between the sign and the symbol (if they are adjacent; otherwise it should be treated as a 0). The following table shows the result of various combinations:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

		p_sep_by_space		
		2	1	0
p_cs_precedes = 1	p_sign_posn = 0	($1.25)	($ 1.25)	($1.25)
	p_sign_posn = 1	+ $1.25	+$ 1.25	+$1.25
	p_sign_posn = 2	$1.25 +	$ 1.25+	$1.25+
	p_sign_posn = 3	+ $1.25	+$ 1.25	+$1.25
	p_sign_posn = 4	$ +1.25	$+ 1.25	$+1.25
p_cs_precedes = 0	p_sign_posn = 0	(1.25 $)	(1.25 $)	(1.25$)
	p_sign_posn = 1	+1.25 $	+1.25 $	+1.25$
	p_sign_posn = 2	1.25$ +	1.25 $+	1.25$+
	p_sign_posn = 3	1.25+ $	1.25 +$	1.25+$
	p_sign_posn = 4	1.25$ +	1.25 $+	1.25$+

The following is an example of the interpretation of the `mon_grouping` keyword. Assuming that the value to be formatted is `123456789` and the `mon_thousands_sep` is `'`, then the following table shows the result. The third column shows the equivalent C Standard {7} string that would be used to accommodate this grouping. It is the responsibility of the utility to perform mappings of the formats in this clause to those used by language bindings such as the C Standard {7}.

mon_grouping	Formatted Value	C Standard {7} String
3;-1	123456'789	"\3\177"
3	123'456'789	"\3"
3;2;-1	1234'56'789	"\3\2\177"
3;2	12'34'56'789	"\3\2"
-1	123456789	"177"

In these examples, the octal value of {CHAR_MAX} is 177.

2.5.2.4 LC_NUMERIC

The LC_NUMERIC category shall define the rules and symbols that shall be used to format nonmonetary numeric information. The operands are strings. For some keywords, the strings only can contain integers. Keywords that are not provided, string values set to the empty string (""), or integer keywords set to –1, shall be used to indicate that the value is unspecified. The following keywords shall be recognized:

`copy` Specify the name of an existing locale to be used as the source for the definition of this category. If this keyword is specified, no other keyword shall be specified.

`decimal_point` The operand is a string containing the symbol that shall be used as the decimal delimiter in numeric, nonmonetary formatted quantities. This keyword cannot be omitted and cannot be set to the empty string. In contexts where other standards limit the `decimal_point` to a single byte, the result of specifying a multibyte operand is

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

unspecified.

`thousands_sep` The operand is a string containing the symbol that shall be used as a separator for groups of digits to the left of the decimal delimiter in numeric, nonmonetary formatted monetary quantities. In contexts where other standards limit the `thousands_sep` to a single byte, the result of specifying a multibyte operand is unspecified.

`grouping` Define the size of each group of digits in formatted nonmonetary quantities. The operand is a sequence of integers separated by semicolons. Each integer specifies the number of digits in each group, with the initial integer defining the size of the group immediately preceding the decimal delimiter, and the following integers defining the preceding groups. If the last integer is not −1, then the size of the previous group (if any) shall be repeatedly used for the remainder of the digits. If the last integer is −1, then no further grouping shall be performed.

Table 2-9 – LC_NUMERIC Category Definition in the POSIX Locale

```
LC_NUMERIC
# This is the POSIX Locale definition for
# the LC_NUMERIC category.
#
decimal_point       "<period>"
thousands_sep       ""
grouping            0
#
END LC_NUMERIC
```

2.5.2.4.1 LC_NUMERIC Rationale. *(This subclause is not a part of P1003.2)*

See the rationale for LC_MONETARY (2.5.2.3.1) for a description of the behavior of grouping.

2.5.2.5 LC_TIME

The LC_TIME category shall define the interpretation of the field descriptors supported by the `date` utility (see 4.15).

The following mandatory keywords shall be recognized:

`copy` Specify the name of an existing locale to be used as the source for the definition of this category. If this keyword is specified, no other keyword shall be specified.

`abday` Define the abbreviated weekday names, corresponding to the `%a` field descriptor. The operand shall consist of seven semicolon-separated strings. The first string shall be the abbreviated name

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2-10 – LC_TIME Category Definition in the POSIX Locale

```
LC_TIME
# This is the POSIX Locale definition for
# the LC_TIME category.
#
# Abbreviated weekday names (%a)
abday   "<S><u><n>";"<M><o><n>";"<T><u><e>";"<W><e><d>";\
        "<T><h><u>";"<F><r><i>";"<S><a><t>"
#
# Full weekday names (%A)
day     "<S><u><n><d><a><y>";"<M><o><n><d><a><y>";\
        "<T><u><e><s><d><a><y>";"<W><e><d><n><e><s><d><a><y>";\
        "<T><h><u><r><s><d><a><y>";"<F><r><i><d><a><y>";\
        "<S><a><t><u><r><d><a><y>"
#
# Abbreviated month names (%b)
abmon   "<J><a><n>";"<F><e><b>";"<M><a><r>";\
        "<A><p><r>";"<M><a><y>";"<J><u><n>";\
        "<J><u><l>";"<A><u><g>";"<S><e><p>";\
        "<O><c><t>";"<N><o><v>";"<D><e><c>"
#
# Full month names (%B)
mon     "<J><a><n><u><a><r><y>";"<F><e><b><r><u><a><r><y>";\
        "<M><a><r><c><h>";"<A><p><r><i><l>";\
        "<M><a><y>";"<J><u><n><e>";\
        "<J><u><l><y>";"<A><u><g><u><s><t>";\
        "<S><e><p><t><e><m><b><e><r>";"<O><c><t><o><b><e><r>";\
        "<N><o><v><e><m><b><e><r>";"<D><e><c><e><m><b><e><r>"
#
# Equivalent of AM/PM (%p)      "AM";"PM"
am_pm   "<A><M>";"<P><M>"
#
# Appropriate date and time representation (%c)
#       "%a %b %e %H:%M:%S %Y"                                                  1
d_t_fmt "<percent-sign><a><space><percent-sign><b><space><percent-sign><e>\     1
<space><percent-sign><H><colon><percent-sign><M>\
<colon><percent-sign><S><space><percent-sign><Y>"
#
# Appropriate date representation (%x)   "%m/%d/%y"
d_fmt   "<percent-sign><m><slash><percent-sign><d><slash><percent-sign><y>"
#
# Appropriate time representation (%X)   "%H:%M:%S"
t_fmt   "<percent-sign><H><colon><percent-sign><M><colon><percent-sign><S>"
#
# Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
t_fmt_ampm "<percent-sign><I><colon><percent-sign><M><colon>\
<percent-sign><S> <percent_sign><p>"
#
END LC_TIME
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

of the first day of the week (Sunday), the second the abbreviated name of the second day, and so on.

`day` Define the full weekday names, corresponding to the `%A` field descriptor. The operand shall consist of seven semicolon-separated strings. The first string shall be the full name of the first day of the week (Sunday), the second the full name of the second day, and so on.

`abmon` Define the abbreviated month names, corresponding to the `%b` field descriptor. The operand shall consist of twelve semicolon-separated strings. The first string shall be the abbreviated name of the first month of the year (January), the second the abbreviated name of the second month, and so on.

`mon` Define the full month names, corresponding to the `%B` field descriptor. The operand shall consist of twelve semicolon-separated strings. The first string shall be the full name of the first month of the year (January), the second the full name of the second month, and so on.

`d_t_fmt` Define the appropriate date and time representation, corresponding to the `%c` field descriptor. The operand shall consist of a string, and can contain any combination of characters and field descriptors. In addition, the string can contain escape sequences defined in Table 2-15.

`d_fmt` Define the appropriate date representation, corresponding to the `%x` field descriptor. The operand shall consist of a string, and can contain any combination of characters and field descriptors. In addition, the string can contain escape sequences defined in Table 2-15.

`t_fmt` Define the appropriate time representation, corresponding to the `%X` field descriptor. The operand shall consist of a string, and can contain any combination of characters and field descriptors. In addition, the string can contain escape sequences defined in Table 2-15.

`am_pm` Define the appropriate representation of the *ante meridiem* and *post meridiem* strings, corresponding to the `%p` field descriptor. The operand shall consist of two strings, separated by a semicolon. The first string shall represent the *ante meridiem* designation, the last string the *post meridiem* designation.

`t_fmt_ampm`
Define the appropriate time representation in the 12-hour clock format with `am_pm`, corresponding to the `%r` field descriptor. The operand shall consist of a string and can contain any combination of characters and field descriptors. If the string is empty, the 12-hour format is not supported in the locale.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

It is implementation defined whether the following optional keywords shall be recognized. If they are not supported, but present in a `localedef` source, they shall be ignored.

`era` Shall be used to define alternate Eras, corresponding to the `%E` field descriptor modifier. The format of the operand is unspecified, but shall support the definition of the `%EC` and `%Ey` field descriptors, and may also define the `era_year` format (`%EY`).

`era_year` Shall be used to define the format of the year in alternate Era format, corresponding to the `%EY` field descriptor.

`era_d_fmt` Shall be used to define the format of the date in alternate Era notation, corresponding to the `%Ex` field descriptor.

`alt_digits` Shall be used to define alternate symbols for digits, corresponding to the `%O` field descriptor modifier. The operand shall consist of semicolon-separated strings. The first string shall be the alternate symbol corresponding with zero, the second string the symbol corresponding with one, and so on. Up to 100 alternate symbol strings can be specified. The `%O` modifier indicates that the string corresponding to the value specified via the field descriptor shall be used instead of the value.

2.5.2.5.1 LC_TIME Rationale. *(This subclause is not a part of P1003.2)*

Although certain of the field descriptors in the POSIX Locale (such as the name of the month) are shown with initial capital letters, this need not be the case in other locales. Programs using these fields may need to adjust the capitalization if the output is going to be used at the beginning of a sentence.

The LC_TIME descriptions of `abday`, `daya`, and `abmon` imply a Gregorian style calendar (7-day weeks, 12-month years, leap years, etc.). Formatting time strings for other types of calendars is outside the scope of this standard.

As specified under the `date` command, the field descriptors corresponding to the optional keywords consist of a modifier followed by a traditional field descriptor (for instance `%Ex`). If the optional keywords are not supported by the implementation or are unspecified for the current locale, these field descriptors shall be treated as the traditional field descriptor. For instance, assume the following keywords:

```
alt_digits      "0th";"1st";"2nd";"3rd";"4th";"5th";\
                "6th";"7th";"8th";"9th";"10th"
d_fmt           "The %Od day of %B in %Y"
```

On 7/4/1776, the `%x` field descriptor would result in ‘‘The 4th day of July in 1776,’’ while 7/14/1789 would come out as ‘‘The 14 day of July in 1789.’’ It can be noted that the above example is for illustrative purposes only; the `%O` modifier is primarily intended to provide for Kanji or Hindi digits in `date` formats.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

While it is clear that an alternate year format is required, there is no consensus on the format or the requirements. As a result, while these keywords are reserved, the details are left unspecified. It is expected that National Standards Bodies will provide specifications.

2.5.2.6 LC_MESSAGES

The LC_MESSAGES category shall define the format and values for affirmative and negative responses. The operands shall be strings or extended regular expressions; see 2.8.4. The following keywords shall be recognized:

`copy` Specify the name of an existing locale to be used as the source for the definition of this category. If this keyword is specified, no other keyword shall be specified.

`yesexpr` The operand shall consist of an extended regular expression that describes the acceptable affirmative response to a question expecting an affirmative or negative response.

`noexpr` The operand shall consist of an extended regular expression that describes the acceptable negative response to a question expecting an affirmative or negative response.

Table 2-11 – LC_MESSAGES Category Definition in the POSIX Locale

```
LC_MESSAGES
# This is the POSIX Locale definition for
# the LC_MESSAGES category.
#
yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
#
noexpr  "<circumflex><left-square-bracket><n><N><right-square-bracket>"
END LC_MESSAGES
```

2.5.2.6.1 LC_MESSAGES Rationale. *(This subclause is not a part of P1003.2)*

The LC_MESSAGES category is described in 2.6 as affecting the language used by utilities for their output. The mechanism used by the implementation to accomplish this, other than the responses shown here in the locale definition file, is not specified by this version of this standard. The POSIX. 1 working group is developing an interface that would allow applications (and, presumably some of the standard utilities) to access messages from various message catalogs, tailored to a user's **LC_MESSAGES** value.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2.5.3 Locale Definition Grammar

The grammar and lexical conventions in this subclause shall together describe the syntax for the locale definition source. The general conventions for this style of grammar are described in 2.1.2. Any discrepancies found between this grammar and other descriptions in this clause shall be resolved in favor of this grammar.

2.5.3.1 Locale Lexical Conventions

The lexical conventions for the locale definition grammar are described in this subclause.

The following tokens shall be processed (in addition to those string constants shown in the grammar):

`LOC_NAME` A string of characters representing the name of a locale.

`CHAR` Any single character.

`NUMBER` A decimal number, represented by one or more decimal digits.

`COLLSYMBOL` A symbolic name, enclosed between angle brackets. The string shall not duplicate any charmap symbol defined in the current charmap (if any), or a `COLLELEMENT` symbol.

`COLLELEMENT` A symbolic name, enclosed between angle brackets, which shall not duplicate either any charmap symbol or a `CHARSYMBOL` symbol.

`CHARSYMBOL` A symbolic name, enclosed between angle brackets, from the current charmap (if any).

`OCTAL_CHAR` One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an `escape_char` (normally a backslash) followed by two or more octal digits.

`HEX_CHAR` One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an `escape_char` followed by the constant `'x'` and two or more hexadecimal digits.

`DECIMAL_CHAR` One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an `escape_char` and followed by a `'d'` and two or more decimal digits.

`ELLIPSIS` The string "`...`".

`EXTENDED_REG_EXP`
An extended regular expression as defined in the grammar in 2.8.5.2.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

EOL The line termination character <newline>.

2.5.3.2 Locale Grammar

This subclause presents the grammar for the locale definition.

```
%token          LOC_NAME
%token          CHAR
%token          NUMBER
%token          COLLSYMBOL COLLELEMENT
%token          CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
%token          ELLIPSIS
%token          EXTENDED_REG_EXP
%token          EOL

%start  locale_definition

%%

locale_definition       : global_statements locale_categories
                        |                   locale_categories
                        ;

global_statements       : global_statements symbol_redefine
                        | symbol_redefine
                        ;

symbol_redefine         : '#escape_char'  CHAR EOL
                        | '#comment_char' CHAR EOL
                        ;

locale_categories       : locale_categories locale_category
                        | locale_category
                        ;

locale_category         : lc_ctype | lc_collate | lc_messages
                        | lc_monetary | lc_numeric | lc_time
                        ;

/*      The following grammar rules are common to all categories */

char_list               : char_list char_symbol
                        | char_symbol
                        ;

char_symbol             : CHAR | CHARSYMBOL
                        | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
                        ;

locale_name             : LOC_NAME
                        | '"' LOC_NAME '"'
                        ;

/*      The following is the LC_CTYPE category grammar */

lc_ctype                : ctype_hdr ctype_keywords         ctype_tlr
                        | ctype_hdr 'copy' locale_name EOL ctype_tlr
                        ;
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
ctype_hdr               : 'LC_CTYPE' EOL                                    2
                        ;                                                   2

ctype_keywords          : ctype_keywords ctype_keyword                      2
                        | ctype_keyword                                     2
                        ;                                                   1

ctype_keyword           : charclass_keyword charclass_list EOL              1
                        | charconv_keyword charconv_list EOL                1
                        ;                                                   1

charclass_keyword       : 'upper' | 'lower' | 'alpha' | 'digit'             1
                        | 'alnum' | 'xdigit' | 'space' | 'print'            1
                        | 'graph' | 'blank' | 'cntrl'                       1
                        ;                                                   1

charclass_list          : charclass_list ';' char_symbol                    2
                        | charclass_list ';' ELLIPSIS ';' char_symbol       1
                        | char_symbol                                       2
                        ;                                                   1

charconv_keyword        : 'toupper'                                         1
                        | 'tolower'                                         1
                        ;                                                   1

charconv_list           : charconv_list ';' charconv_entry                  2
                        | charconv_entry                                    2
                        ;                                                   1

charconv_entry          : '(' char_symbol ',' char_symbol ')'               1
                        ;                                                   1

ctype_tlr               : 'END' 'LC_CTYPE' EOL                              2
                        ;                                                   1

/*      The following is the LC_COLLATE category grammar */                 1

lc_collate              : collate_hdr collate_keywords      collate_tlr     2
                        | collate_hdr 'copy' locale_name EOL collate_tlr    2
                        ;                                                   2

collate_hdr             : 'LC_COLLATE' EOL                                  2
                        ;                                                   2

collate_keywords        :               order_statements                    2
                        | opt_statements order_statements                   2
                        ;                                                   1

opt_statements          : opt_statements collating_symbols                  2
                        | opt_statements collating_elements                 2
                        | collating_symbols                                 1
                        | collating_elements                                1
                        ;                                                   1

collating_symbols       : 'collating-symbol' COLLSYMBOL EOL                 1
                        ;                                                   1

collating_elements      : 'collating-element' COLLELEMENT                   1
                          'from' '"' char_list '"' EOL                      2
                        ;                                                   1
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
order_statements        : order_start collation_order order_end                 1
                        ;                                                       1

order_start             : 'order_start' EOL                                     1
                        | 'order_start' order_opts EOL                          1
                        ;                                                       1

order_opts              : order_opts ';' order_opt                              2
                        | order_opt                                             2
                        ;                                                       1

order_opt               : order_opt ',' opt_word                                2
                        | opt_word                                              2
                        ;                                                       1

opt_word                : 'forward' | 'backward' | 'position'                   2
                        ;                                                       1

collation_order         : collation_order collation_entry                       2
                        | collation_entry                                       2
                        ;                                                       1

collation_entry         : COLLSYMBOL EOL                                        1
                        | collation_element weight_list EOL                     1
                        | collation_element             EOL                     2
                        ;                                                       1

collation_element       : char_symbol                                           1
                        | COLLELEMENT                                           1
                        | ELLIPSIS                                              1
                        | 'UNDEFINED'                                           1
                        ;                                                       1

weight_list             : weight_list ';' weight_symbol                         2
                        | weight_list ';'                                       2
                        | weight_symbol                                         2
                        ;                                                       1

weight_symbol           : char_symbol                                           2
                        | COLLSYMBOL                                            1
                        | '"' char_list '"'                                     1
                        | ELLIPSIS                                              1
                        | 'IGNORE'                                              1
                        ;                                                       1

order_end               : 'order_end' EOL                                       1
                        ;                                                       1

collate_tlr             : 'END' 'LC_COLLATE' EOL                                2
                        ;                                                       1

/*      The following is the LC_MESSAGES category grammar */                   1

lc_messages             : messages_hdr messages_keywords      messages_tlr      2
                        | messages_hdr 'copy' locale_name EOL messages_tlr      2
                        ;                                                       2

messages_hdr            : 'LC_MESSAGES' EOL                                     2
                        ;                                                       2
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
messages_keywords       : messages_keywords messages_keyword
                        | messages_keyword
                        ;

messages_keyword        : 'yesexpr' '"' EXTENDED_REG_EXP '"' EOL
                        | 'noexpr'  '"' EXTENDED_REG_EXP '"' EOL
                        ;

messages_tlr            : 'END' 'LC_MESSAGES' EOL
                        ;

/*      The following is the LC_MONETARY category grammar */

lc_monetary             : monetary_hdr monetary_keywords       monetary_tlr
                        | monetary_hdr 'copy' locale_name EOL  monetary_tlr
                        ;

monetary_hdr            : 'LC_MONETARY' EOL
                        ;

monetary_keywords       : monetary_keywords monetary_keyword
                        | monetary_keyword
                        ;

monetary_keyword        : mon_keyword_string mon_string EOL
                        | mon_keyword_char NUMBER EOL
                        | mon_keyword_char '-1'   EOL
                        | mon_keyword_grouping mon_group_list EOL
                        ;

mon_keyword_string      : 'int_curr_symbol' | 'currency_symbol'
                        | 'mon_decimal_point' | 'mon_thousands_sep'
                        | 'positive_sign' | 'negative_sign'
                        ;

mon_string              : '"' char_list '"'
                        | '""'
                        ;

mon_keyword_char        : 'int_frac_digits' | 'frac_digits'
                        | 'p_cs_precedes' | 'p_sep_by_space'
                        | 'n_cs_precedes' | 'n_sep_by_space'
                        | 'p_sign_posn' | 'n_sign_posn'
                        ;


mon_keyword_grouping    : 'mon_grouping'
                        ;

mon_group_list          : NUMBER
                        | mon_group_list ';' NUMBER
                        ;

monetary_tlr            : 'END' 'LC_MONETARY' EOL
                        ;

/*      The following is the LC_NUMERIC category grammar */

lc_numeric              : numeric_hdr numeric_keywords       numeric_tlr
                        | numeric_hdr 'copy' locale_name EOL numeric_tlr
                        ;
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
numeric_hdr             : 'LC_NUMERIC' EOL                                      2
                        ;                                                       2

numeric_keywords        : numeric_keywords numeric_keyword                      2
                        | numeric_keyword                                       2
                        ;                                                       1

numeric_keyword         : num_keyword_string num_string EOL                     1
                        | num_keyword_grouping num_group_list EOL               1
                        ;                                                       1

num_keyword_string      : 'decimal_point'                                       1
                        | 'thousands_sep'                                       1
                        ;                                                       1

num_string              : '"' char_list '"'                                     1
                        | '""'                                                  1
                        ;                                                       1

num_keyword_grouping    : 'num_grouping'                                        1
                        ;                                                       1

num_group_list          : NUMBER                                                2
                        | num_group_list ';' NUMBER                             2
                        ;                                                       1


numeric_tlr             : 'END' 'LC_NUMERIC' EOL                                2
                        ;                                                       1

/*      The following is the LC_TIME category grammar */                        1

lc_time                 : time_hdr time_keywords          time_tlr              2
                        | time_hdr 'copy' locale_name EOL time_tlr              2
                        ;                                                       1

time_hdr                : 'LC_TIME' EOL                                         2
                        ;                                                       1

time_keywords           : time_keywords time_keyword                            2
                        | time_keyword                                          2
                        ;                                                       1

time_keyword            : time_keyword_name time_list EOL                       2
                        | time_keyword_fmt time_string EOL                      1
                        | time_keyword_opt time_list EOL                        1
                        ;                                                       1

time_keyword_name       : 'abday' | 'day' | 'abmon' | 'mon'                      2
                        ;                                                       1

time_keyword_fmt        : 'd_t_fmt' | 'd_fmt' | 't_fmt' | 'am_pm' | 't_fmt_ampm' 1
                        ;                                                       1

time_keyword_opt        : 'era' | 'era_year' | 'era_d_fmt' | 'alt_digits'        1
                        ;                                                       1

time_list               : time_list ';' time_string                             2
                        | time_string                                           2
                        ;                                                       1

time_string             : '"' char_list '"'                                     1
                        ;                                                       1
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
time_tlr                : 'END' 'LC_TIME' EOL
                        ;
```

2.5.4 Locale Definition Example. *(This subclause is not a part of P1003.2)*

The following is an example of a locale definition file that could be used as input to the `localedef` utility. It assumes that the utility is executed with the `-f` option, naming a *charmap* file with (at least) the following content:

```
CHARMAP
<space>      \x20
<dollar>     \x24
<A>          \101
<a>          \141
<A-acute>    \346
<a-acute>    \365
<A-grave>    \300
<a-grave>    \366
<b>          \142
<C>          \103
<c>          \143
<c-cedilla>  \347
<d>          \x64
<H>          \110
<h>          \150
<eszet>      \xb7
<s>          \x73
<z>          \x7a
END CHARMAP
```

It should not be taken as complete or to represent any actual locale, but only to illustrate the syntax.

A further set of examples is offered as part of Annex F.

```
#
LC_CTYPE
lower   <a>;<b>;<c>;<c-cedilla>;<d>;...;<z>
upper   A;B;C;Ç;...;Z
space   \x20;\x09;\x0a;\x0b;\x0c;\x0d
blank   \040;\011
toupper (<a>,<A>);(b,B);(c,C);(ç,Ç);(d,D);(z,Z)
END LC_CTYPE
#
LC_COLLATE
#
# The following example of collation is based on the proposed
# Canadian standard Z243.4.1-1990, "Canadian Alphanumeric
# Ordering Standard For Character sets of CSA Z234.4 Standard".
# (Other parts of this example locale definition file do not
# purport to relate to Canada, or to any other real culture.)
# The proposed standard defines a 4-weight collation, such that
# in the first pass, characters are compared without regard to
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
# case or accents; in second pass, backwards compare without
# regard to case; in the third pass, forward compare without
# regard to diacriticals.  In the 3 first passes, non-alphabetic
# characters are ignored; in the fourth pass, only special
# characters are considered, such that "The string that has a
# special character in the lowest position comes first.  If two
# strings have a special character in the same position, the
# collation value of the special character determines ordering.
#
# Only a subset of the character set is used here; mostly to
# illustrate the set-up.
#
#
collating-symbol <LOW_VALUE>
collating-symbol <LOWER-CASE>
collating-symbol <SUBSCRIPT-LOWER>
collating-symbol <SUPERSCRIPT-LOWER>
collating-symbol <UPPER-CASE>
collating-symbol <NO-ACCENT>
collating-symbol <PECULIAR>
collating-symbol <LIGATURE>
collating-symbol <ACUTE>
collating-symbol <GRAVE>
# Further collating-symbols follow.
#
# Properly, the standard does not include any multi-character
# collating elements; the one below is added for completeness.
#
collating_element <ch> from <c><h>
collating_element <CH> from <C><H>
collating_element <Ch> from <C><h>
#
order_start forward;backward;forward;forward,position
#
# Collating symbols are specified first in the sequence to allocate
# basic collation values to them, lower that than of any character.

<LOW_VALUE>
<LOWER-CASE>
<SUBSCRIPT-LOWER>
<SUPERSCRIPT-LOWER>
<UPPER-CASE>
<NO-ACCENT>
<PECULIAR>
<LIGATURE>
<ACUTE>
<GRAVE>
<RING-ABOVE>
<DIAERESIS>
<TILDE>
# Further collating symbols are given a basic collating value here.
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
#
# Here follows special characters.
<space>     IGNORE;IGNORE;IGNORE;<space>
# Other special characters follow here.
#
# Here comes the regular characters.
<a>         <a>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
<A>         <a>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
<a-acute>   <a>;<ACUTE>;<LOWER-CASE>;IGNORE
<A-acute>   <a>;<ACUTE>;<UPPER-CASE>;IGNORE
<a-grave>   <a>;<GRAVE>;<LOWER-CASE>;IGNORE
<A-grave>   <a>;<GRAVE>;<UPPER-CASE>;IGNORE
<ae>        <a><e>;<LIGATURE><LIGATURE>;<LOWER-CASE><LOWER-CASE>;IGNORE
<AE>        <a><e>;<LIGATURE><LIGATURE>;<UPPER-CASE><UPPER-CASE>;IGNORE
<b>         <b>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
<B>         <b>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
<c>         <c>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
<C>         <c>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
<ch>        <ch>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
<Ch>        <ch>;<NO-ACCENT>;<PECULIAR>;IGNORE
<CH>        <ch>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
#
# As an example, the strings "Bach" and "bach" could be encoded (for
# compare purposes) as:
# "Bach"  <b>;<a>;<ch>;<LOW_VALUE>;<NO_ACCENT>;<NO_ACCENT>;\                2
#           <NO_ACCENT>;<LOW_VALUE>;<UPPER>;<LOWER>;<LOWER>;<NULL>          2
# "bach"  <b>;<a>;<ch>;<LOW_VALUE>;<NO_ACCENT>;<NO_ACCENT>;\                2
#           <NO_ACCENT>;<LOW_VALUE>;<LOWER>;<LOWER>;<LOWER>;<NULL>          2
#
# The two strings are equal in pass 1 and 2, but differ in pass 3.
#
# Further characters follow.
#
UNDEFINED   IGNORE;IGNORE;IGNORE;IGNORE
#
order_end
#
END LC_COLLATE
#
LC_MONETARY
int_curr_symbol     "USD "
currency_symbol     "$"
mon_decimal_point   "."
mon_grouping        3;0
positive_sign       ""
negative_sign       "-"
p_cs_precedes       1
n_sign_posn         0
END LC_MONETARY
#
LC_NUMERIC
copy "US_en.ASCII"                                                        1
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
3232  END LC_NUMERIC
3233  #
3234  LC_TIME
3235  abday     "Sun";"Mon";"Tue";"Wed";"Thu";"Fri";"Sat"
3236  #
3237  day       "Sunday";"Monday";"Tuesday";"Wednesday";\
3238            "Thursday";"Friday";"Saturday"
3239  #
3240  abmon     "Jan";"Feb";"Mar";"Apr";"May";"Jun";\
3241            "Jul";"Aug";"Sep";"Oct";"Nov";"Dec"
3242  #
3243  mon       "January";"February";"March";"April";\
3244            "May";"June";"July";"August";"September";\
3245            "October";"November";"December"
3246  #
3247  d_t_fmt   "%a %b %d %T %Z %Y\n"
3248  END LC_TIME
3249  #
3250  LC_MESSAGES
3251  yesexpr   "^([yY][[:alpha:]]*)|(OK)"                    1
3252  #
3253  noexpr    "^[nN][[:alpha:]]*"                           1
3254  END LC_MESSAGES
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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,

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.