Information Technology -

Specifications for Cultural Conventions

Work Item: JTC1.22.30.02

ISO/IEC 14652 WD5

Working Draft 5, September 1996

FOREWORD

The text comes primarily from the ISO/IEC 9945-2:1993 POSIX Shell and Utilities standard. Specifications that are specific to POSIX are removed form this Standard, so that other specifications can also apply.

EDITORS FOREWORD

This is working draft 5 for the cultural conventions specification standard. The draft has not been fully checked for errors in formatting, referencing and spelling errors, for which I apologise.

September 1996
Keld Jørn Simonsen

CONTENTS Page

1. SCOPE

Cultural conventions specifications are used to specify the subset of a user's information technology environment that depends on language and cultural conventions.

This Standard specifies a description file format for the specification of cultural conventions, a description file format for character sets, and a description file format for binding character names to ISO/IEC 10646 (11). The specification is compatible with POSIX locale specifications (10), and a locale conformant to POSIX specifications will also be conformant to the specifications in this Standard, while the reverse condition will not hold. The description files are intended to be used via Application Programming Interfaces.

2. NORMATIVE REFERENCES

This Standard incorporates by dated or undated reference, provisions from other publications. These normative references are cited at the appropriate places in the text and the publications are listed below. For dated references, subsequent amendments to or revisions of any of these publications apply to this European Prestandard only when incorporated in it by amendment or revision. For undated references the latest edition of the publication referred applies. Members of ISO and IEC maintain registers of currently valid standards.

(1) ISO 639 Code for the representation of names of languages

(2) ISO 646 Information technology - ISO 7-bit coded character set for information interchange

(3) ISO/IEC 2022 Information technology - Character code structure and extension techniques

(4) ISO 3166 Code for the representation of names of countries

(5) ISO 4217 Codes for the representation of currencies and funds

(6) ISO 8601 Data elements and interchange formats - Information interchange - Representation of dates and times

(7) ISO/IEC 8824 Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)

(8) ISO/IEC 8825 Information technology - Open System Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

(9) ISO/IEC 9899 Programming Language C.

(10) ISO/IEC 9945-2 Information technology - Portable Operating System Interface (POSIX) Part 2: Shell and Utilities

(11) ISO/IEC 10646 Information technology - Universal Multiple-Octet Coded Character Set (UCS)

3. DEFINITIONS AND NOTATIONS

For the purpose of this standard, the following definitions apply:

Affirmative response: An input string that matches one of the responses acceptable to the LC_MESSAGES category keyword "yesexpr", matching an extended regular expression in the current FDCC-set.

Byte: An individually addressable unit of data storage that is equal to or larger than an octet, used to store a character or a portion of a character.

A byte is composed of a contiguous sequence of bits, the number of which is implementation defined. The least significant bit is called the low-order bit; the most significant bit is called the high-order bit.

Character: A member of a set of elements used for the organization, control or representation of data.

Charmap: A text file describing a coded character set.

Coded character: A sequence of one or more bytes representing a single graphic symbol.

Character class: A named set of characters sharing an attribute associated with the name of the class.

Collating element: The smallest entity used to determine the logical ordering of strings.

See collating sequence. A collating element shall consist of either a single character, or two or more characters collating as a single entity. The value of the LC_COLLATE category in the current FDCC-set determines the current set of collating elements.

Collation: The logical ordering of strings according to defined precedence rules.

These rules identify a collation sequence between the collating elements, and such additional rules that can be used to order strings consisting of multiple collating elements.

Collating sequence: The relative order of collating elements as determined by the setting of the LC_LOCALE category in the current FDCC-set. (more in POSIX).

Cultural Convention: A data item for computer use that may vary dependent on language, territory, or other cultural circumstances.

Equivalence class: A set of collating elements with the same primary collation weight.

Elements in an equivalence class are typically elements that naturally group together, such as all accented letters based on the same letter.

The collation order of elements within an equivalence class is determined by the weights assigned on any subsequent levels after the primary weight.

FDCC-set: A Set of Formal Definitions of Cultural Conventions. The definition of the subset of a user's information technology environment that depends on language and cultural conventions. Note: the FDCC-set is a superset of the "locale" term in C and POSIX.

multicharacter collating element: A sequence of two or more characters that collate as an entity.

For example, in some languages two characters are sorted as one letter, this is the case for Danish and Norwegian "aa".

negative response: An input string that matches one of the responses acceptable to the LC_MESSAGES category keyword "noexpr", matching an extended regular expression in the current FDCC-set.

(NUL: A character with all bits set to zero.)

(Null string: see empty string)

printable character: One of the characters included in the "print" character classification of the LC_CTYPE category in the current FDCC-set.

Repertoiremap: A definition of a mapping between symbolic character names and characters for the repertoire of characters used in a FDCC-set, further described in 6.

Text File: A file that contains characters organized into one or more lines.

white space: A sequence of one or more characters that belong to the "space" class as defined via the LC_CTYPE category in the current FDCC-set.

Format description:

In this standard the syntax descriptions for statements is specified in the folowing way:

The format is given in a format string enclosed in double quotes, followed by a number of parameters, separated by a comma. The format of each parameter is given by an escape sequence as follows:

%s	specifies a string
%d	specifies an decimal integer
%c	specifies a character
%o	specifies an octal integer
%x	specifies a hexadecimal integer
%%	specifies a single %
\n	specifies an end-of-line

All other characters in the format string represent themselves.

The notation "..." is used to specify that repetition of the previous specification is optional, and this is done in both the format string and in the parameter list.

Ellipses

A series of characters can be represented by three adjacent periods representing an ellipsis symbol ("..."), or the symbols "...d..." or "...x..." representing respectively the symbolic decimal ellipsis symbol and the symbolic hexadecimal 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.

The symbolic ellipsises are only valid between symbolic character names. They shall be interpreted as all the symbolic names that can be generated by either incrementing the first symbolic names decimally or hexadecimally (corresponding to "...d..." or "...x..." respectively) until the symbolic character name is less or equal the second symbolic character name.

4. FDCC-set

A FDCC-set is the definition of the subset of a user's information technology 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 behaviour of components of the system. Category names correspond to the following environment variable names:

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

Other category names beginning with the 3 characters "LC_" are reserved for future standardization.

This standard defines a normative FDCC-set with values for each of the above categories.

4.1 FDCC-set Definition

FDCC-sets are described with the file format presented in this subclause. For the purposes of this standard, the file is referred to as the FDCC-set definition file or FDCC-set source file.

The FDCC-set definition file shall contain one or more FDCC-set category source definitions, and shall not contain more than one definition for the same FDCC-set 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. A category source definition shall contain either the definition of a category or a copy directive. In the event that some of the information for a FDCC-set category, as specified in this standard, is missing from the FDCC-set source definition, the behaviour of that category, if it is referenced, is unspecified.

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 FDCC-set element, or collating elements, or script symbols. In addition to the keywords defined in this standard, the source can contain implementation-defined keywords. Each keyword within a FDCC-set 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, or script symbols. 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, and the remainder of a line with a <comment char> occuring where a syntactic semicolon may occur, 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 "\".

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 FDCC-set 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:

(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 a charmap file to be used, and shall be replaced by a character value determined from the value associated with the symbolic name in the charmap file. 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 ä "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. Characters encoded as multible bytes 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 FDCC-set definition is created shall be specified, whether using symbolic 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.

The items (2), (3), (4) and (5) are depreciated and are retained for compatibility with the POSIX standard. FDCC-sets should be specified in a coded character set independent way, using symbolic names.

Editor’s note: In the following the "copy" and "reorder-after" keywords are under discussion, as the mechanisms to refer other specifications are not well defined. This could change if there was a registry standard for FDCC-sets. My thoughts are that a string identifying the specification to be referenced could have a specific format, with special entries for specifications to be retrieved over the network "net/" or in the filestore "usr/" and other strings referring to a registration in an international FDCC-set registry.

4.1.1 LC_CTYPE

The LC_CTYPE category defines character classification, case conversion, and other character attributes. In addition, ellipsises and symbolic ellipsises may be used to speify a list of characters.

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 FDCC-set 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, 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, shall automatically belong to this class, with implementation-defined character values.

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

digit — Define the characters to be classified as numeric digits. Only the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 shall be specified, and in ascending sequence by numerical value. If this keyword is not specified, the digits 0 through 9, shall automatically belong to this class, with implementation-defined character values.

space — Define characters to be classified as white-space characters. No character specified for the keywords upper, lower, alpha, dlgit, graph, or xdigit shall be specified. If this keyword is not specified, the characters <space>, <form-feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab>, shall automatically belong to this class, with implementation-defined character values. Any characters included in the class blank shall be automatically included.

cntrl — Define characters to be classified as control characters. No character specified for the keywords upper, lower, alpha, digit, punct, graph, print, or xdigit shall be specified.

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

graph	Define characters to be classified as printable characters, not including the <space> character. If this keyword is not specified, characters specified for the keywords upper, lower, alpha, digit, xdigit, and punct shall belong to this character class. No character specified for the keyword cntrl shall be specified.
print	Define characters to be classified as printable characters, including the <space> character. If this keyword is not provided, characters specified for the keywords upper, lower, alpha, digit, xdigit, punct, and the <space> character shall belong to this character class. No character specified for the keyword cntrl shall be specified.
xdigit	Define the characters to be classified as hexadecimal digits. Only the characters defined for the class digit shall be specified, in ascending sequence by numerical value, followed by one or more sets of six characters representing the hexadecimal digits 10 through 15, with each set in ascending order (for example A, B, C, D, E, F, a, b, c, d, e, f). If this keyword is not specified, the digits 0 through 9, the uppercase letters A through F, and the lowercase letters a through f, 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, 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, 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 1 shows the allowed character class combinations.

Table 1: Valid Character Class Combinations

In	Can also belong to										
Class	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
upper	-	-	A	x	x	x	x	A	A	-	x

lower	-	-	A	x	x	x	x	A	A	-	x
alpha	-	-	-	x	x	x	x	A	A	-	x
digit	x	x	x	-	x	x	x	A	A	A	x
space	x	x	x	x	-	-	*	*	*	x	-
cntrl	x	x	x	x	-	-	x	x	x	x	-
punct	x	x	x	x	-	x	-	A	A	x	-
graph	-	-	-	-	-	x	-	-	A	-	-
print	-	-	-	-	-	x	-	-	-	-	-
xdigit	-	-	-	-	x	x	x	A	A	-	x
blank	x	x	x	x	A	-	*	*	*	x	-

NOTES:
Note 1: Explanation of codes:
A Automatically included; see text
- Permitted
x Mutually exclusive
* See note 2

Note 2: The <space> character, which is part of the space and blank class, 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.

Character string transformation

The following may be used to transform or transliterate strings. The transliteration may for example be from the Cyrillic script to the Latin script. Transliteration is often language dependent, and the language to be transliterated to is identified with the FDCC-set, which may also be used to identify a specific language to be transliterated from. Transformation of an incoming character string to a character string in a FDCC-set can be specificed with the following keywords and transformation statements.

transform_start	The "transform_start" keyword is followed by one or more transformation statements assigning character transformation values to transformating elements, and include statements copying transformation specifications from other FDCC-sets.
transform_end	The end of the transformation statements.
include	The name of the FDCC-set in text form and culture to transform from and the repertoiremap for the FDCC-set to be used for the definition of the transformation statements. Other transformation statements may follow to replace specification of the copied FDCC-set. This keyword is optional.
default_missing	defines one or more characters to be used if no transformation statement can be applied to a input <transformation-identifier>.

Transformation statements

The "transform_start" keyword may be followed by transformation identifier entries. The syntax for the transformation identifier entries is:

```
"%s %s;%s;...;%s\n",<transformation-identifier>,<transformation-string>,
   <transformation-string>,...
```

Each <transformation-identifier> shall consist of one or more characters (in any of the forms defined in 4.1). The longest matching string is the one selected for transliteration.

The order the transformation-strings is defined in, defines the precedence of transformations. The first transformation-string that satisfies the transformation (by for example having characters that are all in the coded character set that is transformed into and having the desired string length) is chosen.

If more than one transformation statement is given for a given <transformation-identifier> this is an error, unless the c-option is given on the localedef utility - then a warning is given and the last transformation statement is assumed.

Example:

```
<ae> <a:>;<e*>;<a><e>;"<e>"
<s>  <s*>;<s=>
<K><O>  <KO>
```

The first line defines a number of transformations for the LATIN LETTER AE, including into LATIN LETTER A WITH DIAERESIS, GREEK LETTER EPSILON, the two Latin letters A and E, and finally the LATIN LETTER E.

The second line defines transformation of the LATIN LETTER S into GREEK LETTER SIGMA, and CYRILLIC LETTER ES.

The 3rd line transforms the two Latin letters K and O into the Japanese Hiragana character KO

include keyword

The "include" keyword specifies a set of transformation statements in text form to be included in the current transformation.

The syntax of the "include" statement is:

```
"include %s;%s\n",<FDCC-set>,<repertoiremap>
```

<FDCC-set> is a string identifying the FDCC-set to be included from.

<repertoiremap> is a string identifying the repertoiremap used in the FDCC-set being included, and is used to map character specifications from the specified FDCC-set into the

current FDCC-set.

The "i18n" FDCC-set for the LC_CTYPE is defied as follows:

```
LC_CTYPE

 digit   <0>;<1>;<2>;<3>;<4>;<5>;<6>;<7>;<8>;<9>
 xdigit  <0>;<1>;<2>;<3>;<4>;<5>;<6>;<7>;<8>;<9>;/
<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
 blank   <SP>;<HT>;<NS>
 space   <SP>;<LF>;<VT>;<FF>;<CR>;<HT>;<NS>
 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>;<A!>;<A’>;<A/>>;<A?>;/
<A:>;<AA>;<AE>;<C,>;<E!>;<E’>;/
<E/>>;<E:>;<I!>;<I’>;<I/>>;<I:>;/
<D->;<N?>;<O!>;<O’>;<O/>>;<O?>;/
<O:>;<O//>;<U!>;<U’>;<U/>>;<U:>;/
<Y’>;<TH>;<A->;<A(>;<A;>;<C’>;/
<C/>>;<C.>;<C<>;<D<>;<D//>;<E->;/
<E(>;<E.>;<E;>;<E<>;<G/>>;<G(>;/
<G.>;<G,>;<H/>>;<H//>;<I?>;<I->;/
<I;>;<I.>;<IJ>;<J/>>;<K,>;<L’>;/
<L,>;<L<>;<L.>;<L//>;<N’>;<N,>;/
<N<>;<NG>;<O->;<O(>;<O">;<OE>;/
<R’>;<R,>;<R<>;<S’>;<S/>>;<S,>;/
<S<>;<T,>;<T<>;<T//>;<U?>;<U->;/
<U(>;<U0>;<U">;<U;>;<W/>>;<Y/>>;/
<Y:>;<Z’>;<Z.>;<Z<>;<ED>;<A1>;/
<A7>;<A3>;<G//>;<G<>;<K<>;<O;>;/
<O1>;<EZ>;<AA’>;<AE’>;<O//’>;<A%>;/
<E%>;<Y%>;<I%>;<O%>;<U%>;<W%>;/
<A*>;<B*>;<G*>;<D*>;<E*>;<Z*>;/
<Y*>;<H*>;<I*>;<K*>;<L*>;<M*>;/
<N*>;<C*>;<O*>;<P*>;<R*>;<S*>;/
<T*>;<U*>;<F*>;<X*>;<Q*>;<W*>;/
<J*>;<V*>;<IO>;<D%>;<G%>;<IE>;/
<DS>;<II>;<YI>;<J%>;<LJ>;<NJ>;/
<Ts>;<KJ>;<V%>;<DZ>;<A=>;<B=>;/
<V=>;<G=>;<D=>;<E=>;<Z%>;<Z=>;/
<I=>;<J=>;<K=>;<L=>;<M=>;<N=>;/
<O=>;<P=>;<R=>;<S=>;<T=>;<U=>;/
<F=>;<H=>;<C=>;<C%>;<S%>;<Sc>;/
<=">;<Y=>;<%">;<JE>;<JU>;<JA>;/
<W!>;<W’>;<W:>
```

```
 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>;<ss>;<a!>;<a’>;<a/>>;/
<a?>;<a:>;<aa>;<ae>;<c,>;<e!>;/
<e’>;<e/>>;<e:>;<i!>;<i’>;<i/>>;/
<i:>;<d->;<n?>;<o!>;<o’>;<o/>>;/
<o?>;<o:>;<o//>;<u!>;<u’>;<u/>>;/
<u:>;<y’>;<th>;<y:>;<a->;<a(>;/
<a;>;<c’>;<c/>>;<c.>;<c<>;<d<>;/
<d//>;<e->;<e(>;<e.>;<e;>;<e<>;/
<g/>>;<g(>;<g.>;<g,>;<h/>>;<h//>;/
<i?>;<i->;<i;>;<i.>;<ij>;<j/>>;/
<k,>;<kk>;<l’>;<l,>;<l<>;<l.>;/
<l//>;<n’>;<n,>;<n<>;<’n>;<ng>;/
<o->;<o(>;<o">;<oe>;<r’>;<r,>;/
<r<>;<s’>;<s/>>;<s,>;<s<>;<t,>;/
<t<>;<t//>;<u?>;<u->;<u(>;<u0>;/
<u">;<u;>;<w/>>;<y/>>;<z’>;<z.>;/
<z<>;<a1>;<a7>;<a3>;<g//>;<g<>;/
<k<>;<o;>;<o1>;<ez>;<aa’>;<ae’>;/
<o//’>;<i3>;<a%>;<e%>;<y%>;<i%>;/
<u3>;<a*>;<b*>;<g*>;<d*>;<e*>;/
<z*>;<y*>;<h*>;<i*>;<k*>;<l*>;/
<m*>;<n*>;<c*>;<o*>;<p*>;<r*>;/
<*s>;<s*>;<t*>;<u*>;<f*>;<x*>;/
<q*>;<w*>;<j*>;<v*>;<o%>;<u%>;/
<w%>;<a=>;<b=>;<v=>;<g=>;<d=>;/
<e=>;<z%>;<z=>;<i=>;<j=>;<k=>;/
<l=>;<m=>;<n=>;<o=>;<p=>;<r=>;/
<s=>;<t=>;<u=>;<f=>;<h=>;<c=>;/
<c%>;<s%>;<sc>;<=’>;<y=>;<%’>;/
<je>;<ju>;<ja>;<io>;<d%>;<g%>;/
<ie>;<ds>;<ii>;<yi>;<j%>;<lj>;/
<nj>;<ts>;<kj>;<v%>;<dz>;<w!>;/
<w’>;<w:>;<nS>;<sB>;<ed>;<a8>;/
<b8>;<c8>;<d8>;<e8>;<f8>;<g8>;/
<h8>;<i8>;<j8>;<k8>;<l8>;<m8>;/
<n8>;<o8>;<p8>;<q8>;<r8>;<s8>;/
<t8>;<u8>;<v8>;<w8>;<x8>;<y8>;/
<z8>;<fi>;<fl>

 alpha   <A>;<B>;<C>;<D>;<E>;<F>;/
<G>;<H>;<I>;<J>;<K>;<L>;/
<M>;<N>;<O>;<P>;<Q>;<R>;/
<S>;<T>;<U>;<V>;<W>;<X>;/
```

```
<Y>;<Z>;<a>;<b>;<c>;<d>;/
<e>;<f>;<g>;<h>;<i>;<j>;/
<k>;<l>;<m>;<n>;<o>;<p>;/
<q>;<r>;<s>;<t>;<u>;<v>;/
<w>;<x>;<y>;<z>;<-->;<A!>;/
<A’>;<A/>>;<A?>;<A:>;<AA>;<AE>;/
<C,>;<E!>;<E’>;<E/>>;<E:>;<I!>;/
<I’>;<I/>>;<I:>;<D->;<N?>;<O!>;/
<O’>;<O/>>;<O?>;<O:>;<O//>;<U!>;/
<U’>;<U/>>;<U:>;<Y’>;<TH>;<ss>;/
<a!>;<a’>;<a/>>;<a?>;<a:>;<aa>;/
<ae>;<c,>;<e!>;<e’>;<e/>>;<e:>;/
<i!>;<i’>;<i/>>;<i:>;<d->;<n?>;/
<o!>;<o’>;<o/>>;<o?>;<o:>;<o//>;/
<u!>;<u’>;<u/>>;<u:>;<y’>;<th>;/
<y:>;<A->;<a->;<A(>;<a(>;<A;>;/
<a;>;<C’>;<c’>;<C/>>;<c/>>;<C.>;/
<c.>;<C<>;<c<>;<D<>;<d<>;<D//>;/
<d//>;<E->;<e->;<E(>;<e(>;<E.>;/
<e.>;<E;>;<e;>;<E<>;<e<>;<G/>>;/
<g/>>;<G(>;<g(>;<G.>;<g.>;<G,>;/
<g,>;<H/>>;<h/>>;<H//>;<h//>;<I?>;/
<i?>;<I->;<i->;<I;>;<i;>;<I.>;/
<i.>;<IJ>;<ij>;<J/>>;<j/>>;<K,>;/
<k,>;<kk>;<L’>;<l’>;<L,>;<l,>;/
<L<>;<l<>;<L.>;<l.>;<L//>;<l//>;/
<N’>;<n’>;<N,>;<n,>;<N<>;<n<>;/
<’n>;<NG>;<ng>;<O->;<o->;<O(>;/
<o(>;<O">;<o">;<OE>;<oe>;<R’>;/
<r’>;<R,>;<r,>;<R<>;<r<>;<S’>;/
<s’>;<S/>>;<s/>>;<S,>;<s,>;<S<>;/
<s<>;<T,>;<t,>;<T<>;<t<>;<T//>;/
<t//>;<U?>;<u?>;<U->;<u->;<U(>;/
<u(>;<U0>;<u0>;<U">;<u">;<U;>;/
<u;>;<W/>>;<w/>>;<Y/>>;<y/>>;<Y:>;/
<Z’>;<z’>;<Z.>;<z.>;<Z<>;<z<>;/
<ED>;<A1>;<a1>;<A7>;<a7>;<A3>;/
<a3>;<G//>;<g//>;<G<>;<g<>;<K<>;/
<k<>;<O;>;<o;>;<O1>;<o1>;<EZ>;/
<ez>;<AA’>;<aa’>;<AE’>;<ae’>;<O//’>;/
<o//’>;<;S>;<A%>;<E%>;<Y%>;<I%>;/
<O%>;<U%>;<W%>;<i3>;<A*>;<B*>;/
<G*>;<D*>;<E*>;<Z*>;<Y*>;<H*>;/
<I*>;<K*>;<L*>;<M*>;<N*>;<C*>;/
<O*>;<P*>;<R*>;<S*>;<T*>;<U*>;/
<F*>;<X*>;<Q*>;<W*>;<J*>;<V*>;/
<a%>;<e%>;<y%>;<i%>;<u3>;<a*>;/
```

```
<b*>;<g*>;<d*>;<e*>;<z*>;<y*>;/
<h*>;<i*>;<k*>;<l*>;<m*>;<n*>;/
<c*>;<o*>;<p*>;<r*>;<*s>;<s*>;/
<t*>;<u*>;<f*>;<x*>;<q*>;<w*>;/
<j*>;<v*>;<o%>;<u%>;<w%>;<IO>;/
<D%>;<G%>;<IE>;<DS>;<II>;<YI>;/
<J%>;<LJ>;<NJ>;<Ts>;<KJ>;<V%>;/
<DZ>;<A=>;<B=>;<V=>;<G=>;<D=>;/
<E=>;<Z%>;<Z=>;<I=>;<J=>;<K=>;/
<L=>;<M=>;<N=>;<O=>;<P=>;<R=>;/
<S=>;<T=>;<U=>;<F=>;<H=>;<C=>;/
<C%>;<S%>;<Sc>;<=">;<Y=>;<%">;/
<JE>;<JU>;<JA>;<a=>;<b=>;<v=>;/
<g=>;<d=>;<e=>;<z%>;<z=>;<i=>;/
<j=>;<k=>;<l=>;<m=>;<n=>;<o=>;/
<p=>;<r=>;<s=>;<t=>;<u=>;<f=>;/
<h=>;<c=>;<c%>;<s%>;<sc>;<='>;/
<y=>;<%'>;<je>;<ju>;<ja>;<io>;/
<d%>;<g%>;<ie>;<ds>;<ii>;<yi>;/
<j%>;<lj>;<nj>;<ts>;<kj>;<v%>;/
<dz>;<W!>;<w!>;<W'>;<w'>;<W:>;/
<w:>;<nS>;<ed>;<a8>;<b8>;<c8>;/
<d8>;<e8>;<f8>;<g8>;<h8>;<i8>;/
<j8>;<k8>;<l8>;<m8>;<n8>;<o8>;/
<p8>;<q8>;<r8>;<s8>;<t8>;<u8>;/
<v8>;<w8>;<x8>;<y8>;<z8>;<fi>;/
<fl>

 cntrl    <NU>;<SH>;<SX>;<EX>;<ET>;<EQ>;/
<AK>;<BL>;<BS>;<HT>;<LF>;<VT>;/
<FF>;<CR>;<SO>;<SI>;<DL>;<D1>;/
<D2>;<D3>;<D4>;<NK>;<SY>;<EB>;/
<CN>;<EM>;<SB>;<EC>;<FS>;<GS>;/
<RS>;<US>;<DT>;<PA>;<HO>;<BH>;/
<NH>;<IN>;<NL>;<SA>;<ES>;<HS>;/
<HJ>;<VS>;<PD>;<PU>;<RI>;<S2>;/
<S3>;<DC>;<P1>;<P2>;<TS>;<CC>;/
<MW>;<SG>;<EG>;<SS>;<GC>;<SC>;/
<CI>;<ST>;<OC>;<PM>;<AC>

 punct    <!>;<">;<Nb>;<DO>;/
<%>;<&>;<'>;<(>;<)>;<*>;/
<+>;<,>;<->;<.>;<//>;<:>;/
<;>;<<>;<=>;</>>;<?>;<At>;/
<<(>;<////>;<)/>>;<'/>>;<_>;<'!>;/
<(!>;<!!>;<!)>;<'?>;/
<!I>;<Ct>;<Pd>;<Cu>;<Ye>;<BB>;/
```

```
<SE>;<’:>;<Co>;<-a>;<<<>;<NO>;/
<Rg>;<DG>;<+->;<2S>;<3S>;<’’>;/
<My>;<PI>;<.M>;<’,>;<1S>;<-o>;/
</>/>>;<14>;<12>;<34>;<?I>;<*X>;/
<-:>;<’<>;<’(>;<’.>;<’0>;<’;>;/
<’">;<’%>;<-N>;<-M>;<’6>;<’9>;/
<.9>;<9’>;<"6>;<"9>;<:9>;<//->;/
<//=>;<%0>;<1">;<<1>;</>1>;<’->;/
<//f>;<Li>;<Pt>;<N0>;<TM>;<Om>;/
<18>;<38>;<58>;<78>;<<->;<-!>;/
<-/>>;<-v>;<==>;<dP>;<//0>;<*P>;/
<+Z>;<Ob>;<Sb>;<RT>;<00>;<AN>;/
<(U>;<In>;<?1>;<?2>;<!=>;<=3>;/
<=<>;</>=>;<.3>;<NI>;<Iu>;<Il>;/
<hh>;<HH>;<vv>;<VV>;<dr>;<dR>;/
<Dr>;<DR>;<dl>;<dL>;<Dl>;<LD>;/
<ur>;<uR>;<Ur>;<UR>;<ul>;<uL>;/
<Ul>;<UL>;<vr>;<vR>;<Vr>;<VR>;/
<vl>;<vL>;<Vl>;<VL>;<dh>;<dH>;/
<Dh>;<DH>;<uh>;<uH>;<Uh>;<UH>;/
<vh>;<vH>;<Vh>;<VH>;<TB>;<LB>;/
<FB>;<lB>;<RB>;<.S>;<:S>;<?S>;/
<fS>;<sB>;<Db>;<Md>;<UB>;<"!>;/
<"’>;<"/>>;<"?>;<"->;<"(>;<".>;/
<":>;<"0>;<"">;<"<>;<",>;<";>;/
<"_>;<Fl>

 tolower (<A->,<a->); (<A>,<a>);/
(<A:>,<a!>); (<A(>,<a:>);/
(<A;>,<a(>); (<A1>,<a;>);/
(<A3>,<a1>); (<A7>,<a3>);/
(<AE>,<a7>); (<AE’>,<ae>);/
(<B>,<ae’>); (<C>,<b>);/
(<C,>,<c>); (<C’>,<c,>);/
(<C/>>,<c’>); (<C.>,<c/>>);/
(<C<>,<c.>); (<D//>,<c<>);/
(<E->,<e->); (<E>,<e>);/
(<E:>,<e!>); (<E(>,<e:>);/
(<E.>,<e(>); (<E;>,<e.>);/
(<E<>,<e;>); (<ED>,<e<>);/
(<EZ>,<ed>); (<F>,<ez>);/
(<J>,<j>); (<J/>>,<j/>>);/
(<K>,<k>); (<K,>,<k,>);/
(<K<>,<k<>); (<L//>,<l//>);/
(<L>,<l>); (<L’>,<l’>);/
(<L,>,<l,>); (<L<>,<l<>);/
(<L.>,<l.>); (<M>,<m>);/
```

```
(<N>,<n>); (<N?>,<n?>);/
(<N’>,<n’>); (<N,>,<n,>);/
(<N<>,<n<>); (<O//>,<o//>);/
(<O->,<o->); (<O>,<o>);/
(<O!>,<o!>); (<O’>,<o’>);/
(<O/>>,<o/>>); (<O?>,<o?>);/
(<O:>,<o:>); (<O(>,<o(>);/
(<O">,<o">); (<O;>,<o;>);/
(<O1>,<o1>); (<OE>,<oe>);/
(<Q>,<p>); (<R>,<q>);/
(<U->,<u->); (<U>,<u>);/
(<U!>,<u!>); (<U’>,<u’>);/
(<U:>,<u/>>); (<U?>,<u:>);/
(<U(>,<u?>); (<U">,<u(>);/
(<U;>,<u">); (<U0>,<u;>);/
(<V>,<u0>); (<W>,<v>);/
(<X>,<d//>); (<Y>,<g//>);/
(<Y’>,<y>); (<Y/>>,<w/>>);/
(<Y:>,<y/>>); (<Z>,<w:>);/
(<Z’>,<z>); (<Z.>,<z’>);/
(<Z<>,<z<>); (<AA>,<aa>);/
(<AA’>,<aa’>); (<A*>,<a%>);/
(<B*>,<a*>); (<C*>,<b*>);/
(<D*>,<c*>); (<E*>,<e%>);/
(<F*>,<e*>); (<I*>,<j*>);/
(<K*>,<k*>); (<L*>,<l*>);/
(<M*>,<m*>); (<N*>,<n*>);/
(<O*>,<o%>); (<P*>,<o*>);/
(<Q*>,<p*>); (<R*>,<q*>);/
(<U*>,<u*>); (<V*>,<u%>);/
(<W*>,<v*>); (<Y*>,<y%>);/
(<Z*>,<w%>); (<=">,<z*>);/
(<%">,<=’>); (<A=>,<%’>);/
(<B=>,<a=>); (<C%>,<b=>);/
(<D=>,<c%>); (<E=>,<e=>);/
(<J=>,<j=>); (<JA>,<j%>);/
(<JE>,<ja>); (<JU>,<je>);/
(<K=>,<ju>); (<L=>,<k=>);/
(<LJ>,<l=>); (<M=>,<lj>);/
(<N=>,<m=>); (<NJ>,<n=>);/
(<O=>,<nj>); (<P=>,<o=>);/
(<R=>,<p=>); (<U=>,<u=>);/
(<V=>,<v=>); (<Y=>,<g=>);/
(<Z%>,<dz>); (<Z=>,<z%>)

 toupper (<a->,<A->); (<a>,<A>);/
(<a!>,<A:>); (<a:>,<A(>);/
```

(<a(>,<A;>); (<a;>,<A1>);/
(<a1>,<A3>); (<a3>,<A7>);/
(<a7>,<AE>); (<ae>,<AE’>);/
(<ae’>,<B>); (<b>,<C>);/
(<c>,<C,>); (<c,>,<C’>);/
(<c’>,<C/>>); (<c/>>,<C.>);/
(<c.>,<C<>); (<c<>,<D//>);/
(<e->,<E->); (<e>,<E>);/
(<e!>,<E:>); (<e:>,<E(>);/
(<e(>,<E.>); (<e.>,<E;>);/
(<e;>,<E<>); (<e<>,<ED>);/
(<ed>,<EZ>); (<ez>,<F>);/
(<j>,<J>); (<j/>>,<J/>>);/
(<k>,<K>); (<k,>,<K,>);/
(<k<>,<K<>); (<l//>,<L//>);/
(<l>,<L>); (<l’>,<L’>);/
(<l,>,<L,>); (<l<>,<L<>);/
(<l.>,<L.>); (<m>,<M>);/
(<n>,<N>); (<n?>,<N?>);/
(<n’>,<N’>); (<n,>,<N,>);/
(<n<>,<N<>); (<o//>,<O//>);/
(<o->,<O->); (<o>,<O>);/
(<o!>,<O!>); (<o’>,<O’>);/
(<o/>>,<O/>>); (<o?>,<O?>);/
(<o:>,<O:>); (<o(>,<O(>);/
(<o">,<O">); (<o;>,<O;>);/
(<o1>,<O1>); (<oe>,<OE>);/
(<p>,<Q>); (<q>,<R>);/
(<u->,<U->); (<u>,<U>);/
(<u!>,<U!>); (<u’>,<U’>);/
(<u/>>,<U:>); (<u:>,<U?>);/
(<u?>,<U(>); (<u(>,<U">);/
(<u">,<U;>); (<u;>,<U0>);/
(<u0>,<V>); (<v>,<W>);/
(<d//>,<X>); (<g//>,<Y>);/
(<y>,<Y’>); (<w/>>,<Y/>>);/
(<y/>>,<Y:>); (<w:>,<Z>);/
(<z>,<Z’>); (<z’>,<Z.>);/
(<z<>,<Z<>); (<aa>,<AA>);/
(<aa’>,<AA’>); (<a%>,<A*>);/
(<a*>,<B*>); (<b*>,<C*>);/
(<c*>,<D*>); (<e%>,<E*>);/
(<e*>,<F*>); (<j*>,<I*>);/
(<k*>,<K*>); (<l*>,<L*>);/
(<m*>,<M*>); (<n*>,<N*>);/
(<o%>,<O*>); (<o*>,<P*>);/
(<p*>,<Q*>); (<q*>,<R*>);/

```
(<u*>,<U*>); (<u%>,<V*>);/
(<v*>,<W*>); (<y%>,<Y*>);/
(<w%>,<Z*>); (<z*>,<=">);/
(<=’>,<%">); (<%’>,<A=>);/
(<a=>,<B=>); (<b=>,<C%>);/
(<c%>,<D=>); (<e=>,<E=>);/
(<j=>,<J=>); (<j%>,<JA>);/
(<ja>,<JE>); (<je>,<JU>);/
(<ju>,<K=>); (<k=>,<L=>);/
(<l=>,<LJ>); (<lj>,<M=>);/
(<m=>,<N=>); (<n=>,<NJ>);/
(<nj>,<O=>); (<o=>,<P=>);/
(<p=>,<R=>); (<u=>,<U=>);/
(<v=>,<V=>); (<g=>,<Y=>);/
(<dz>,<Z%>); (<z%>,<Z=>)
 END LC_CTYPE
```

4.1.2 LC_COLLATE

A collation sequence definition defines the relative order between collating elements (characters and multicharacter collating elements) in the FDCC-set. 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 FDCC-set 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.
(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 FDCC-set to be used as the source for the definition of this category. If this keyword is specified, only the "reorder-after", "reorder-end", "reorder-scripts" and "reorder-scripts-end" keywords may also be specified. The FDCC-set shall be copied in source form.
script	Define a script symbol representing a set of collation order statements. This keyword is optional.
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.
reorder-after	Redefine collating rules. Specify after which collating element the redefinition of collation order shall take order. This statement is followed by one or more collation order statements, reassigning character collation values and collation weights to collating elements.
reorder-end	Specify the end of the "reorder-after" collating order statements.
reorder-script	Redefine the order of scripts. This statement is followed by one or more script symbols, reassigning character collation values and collation weights to collating elements.
reorder-script-end	Specify the end of the "reorder-scripts" script order statements.

4.2.2.1 "script" keyword

This keyword shall be used to define symbols for use in script related statements; suc as the "order_start", "reorder-scripts" and "reorder-scripts-end" keywords. The syntax is

"script %s\n", <script-symbol>

The <script-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 <script-symbol> defined via this keyword is only recognized with the LC_COLLATE category.

Example:
script <LATIN>
script <ARABIC>

4.1.2.2 "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.

Example:
collating-element <ch> from <c><h>
collating-element <e-acute> from <acute><e>
collating-element <aa> from aa

4.2.2.3 "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 <CAPITAL>
collating-symbol <HIGH>

4.1.2.4 "order_start" keyword

The order_start keyword shall precede collation order entries and also defines the number of weights for this collation sequence definition, the collation script name 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 first operand may be a <script-symbol> surrounded by "<" and ">" and the set of collating statements following the "order_start" keyword until the "order_end" keyword are identified with this <script_symbol>. The remaining 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-IGNOREd elements in the strings. The string containing a non-IGNOREd element after the fewest IGNOREd collating elements from the start of the compare shall collate first. If both strings contain a non-IGNOREd character in the same relative position, the collating values assigned to the elements shall determine the ordering. In case of equality, subsequent non-IGNOREd characters shall be considered in the same manner.

The directives forward and backward are mutually exclusive.

Examples:
order_start forward;backward
order_start <CYRILLIC>;forward;forward

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

4.1.2.5 Collation Order

The order_start keyword shall be followed by collating statements. The syntax for the collating statements 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 4.1), 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. Symbolic elipses may be used as the ellipses symbol, but generating symbolic character names, and thus have a better chance of portability between implementations.

The symbol UNDEFINED shall be interpreted as including all coded character set values not specified explicitly or via the ellipsis or one of the symbolic elipsises symbols. Such characters shall be 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 here), <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.

A <comment character> occurring where the delimiter ";" may occur, terminates the collating statement.

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

For example, the collation statement

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

is equal to

```
<a>
```

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

The collation order as defined in this clause defines the interpretation of bracket expressions in regular expressions (see POSIX-2 clause 2.8.3.2).

Example:

```
order_start     forward;backward
UNDEFINED       IGNORE;IGNORE
<LOW>
<space>         <LOW>; <space>
...             <LOW>;
<a>             <a>;<a>
<a-acute>       <a>;<a-acute>
```

```
<a-grave>       <a>;<a-grave>
<A>             <a>;<A>
<A-acute>       <a>;<A-acute>
<A-grave>       <a>;<A-grave>
<Ch>            <Ch>;<Ch>
<Ch>            <Ch>;<Ch>
<s>             <s>;<s>
<eszet>         <s><s>;<eszet><eszet>
...             <HIGH>;
                <HIGH>
order_end
```

This example is interpreted as follows:

(1) The UNDEFINED means that all characters not specified in this definition (explicitly or via the ellipsis) shall be ignored for collation purposes; for regular expression purposes they are ordered first.
(2) All characters between <space> and <a> shall have the same primary equivalence class and individual secondary weights based on their ordinal encoded values.
(3) All characters based on the upper or lowercase character "a" belong to the same primary equivalence class.
(4) The multicharacter collating element <c><h> is represented by the collating symbol <ch> and belongs to the same primary equivalence class as the multicharacter collating element <C><h>.
(5) Note that it is not possible to use the collating element <ss> as a weight and expect it to be expanded to the string "ss". When used as a weight, any collating-element represents the relative order assigned to it in the character collation sequence, not the string from which it was derived (compare with <ch>).

4.1.2.6 "order_end" keyword

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

4.1.2.7 "reorder-after" keyword

The "reorder-after" keyword shall be used to specify a modification to a copied collation specification of an existing FDCC-set. There can be more than one "reorder-after" statement in a collating specification. The syntax shall be:

    "reorder-after %s\n",<collating-symbol>

The <collating-symbol> operand shall be a symbolic name, enclosed between angle brackets, and shall be present in the source FDCC-set copied via the "copy" keyword. The "reorder-after" statement is followed by one or more collation statements as described in the "Collating Order" clause (4.1.2.5), with the exception that the ellipsis symbol (...) shall not be used.

Each collation statement reassigns character collation values and collation weights to collating elements existing in the copied collation specification, by removing the collating statement from the copied specification, and inserting the collating element in the collating sequence with the new collation weights after the preceding collating element of the "reorder-after" specification, the first collating element in the collation sequence being the <collating-symbol> specified on the "reorder-after" statement.

A "reorder-after" specification is terminated by another "reorder-after" specification or the "reorder-end" statement.

4.1.2.8 "reorder-end" keyword

The "reorder-end" keyword shall specify the end of a list of collating statements, initiated by the "reorder-after" keyword.

4.1.2.9 Example of "reorder-after":

```
reorder-after <y8>
<U:>          <Y>;<U:>;<CAPITAL>
<u:>          <Y>;<U:>;<SMALL>
reorder-after <z8>
<AE>          <AE>;<NONE>;<CAPITAL>
<ae>          <AE>;<NONE>;<SMALL>
<A:>          <AE>;<DIAERESIS>;<CAPITAL>
<a:>          <AE>;<DIAERESIS>;<SMALL>
<O/>          <O/>;<NONE>;<CAPITAL>
<o/>          <O/>;<NONE>;<SMALL>
<AA>          <AA>;<NONE>;<CAPITAL>
<aa>          <AA>;<NONE>;<SMALL>
reorder-end
```

The example is interpreted as follows:

1. The collating element <U:> is removed from the copied collating sequence and inserted after <y8> in the collating sequence with the new weights. The collating element <u:> is removed from the copied collating sequence and inserted in the resulting collation sequence after <U:> with the new weights.

2. The second "reorder-after" statement terminates the first list of reordering collation identifier entries, and initiates a second list, rearranging the order and weights for the <AE>, <ae>, <A:>, <a:>, <O/>, and <o/> collating elements after the <z8> collating symbol in the copied specification.

3. The "reorder-end" statement terminates the second list of reordering entries.

4. Thus for the original sequence

   ... (U u Ü ü) V v W w X x Y y Z z

   this example reordering gives

   ... U u V v W w X x (Y y Ü ü) Z z (Æ æ Ä ä) Ø ø Å å

4.1.2.10 "reorder-scripts" keyword

The "reorder-scripts" keyword shall be used to specify a modification to a copied collation specification of an existing FDCC-set. The "reorder-scripts" statement is followed by one or more statements consisting of script reordering statements.

4.1.2.11 script reordering statements

The script reordering statements rearranges the set of collating entries and changes sorting rules for the set of collating entries identified by a script symbol in a preceding "order_start" statement. Each script reorder statement has the syntax:

"%s %s;...%s\n", <script-symbol>, <sort-rules>, <sort-rules> ...

The <script-symbol> identifies the set of collating entries.

The <sort-rules> are as described for the "order_start" keyword. Specified <sort-rules> replace the specification for the ordering given on the "order_start" statement identified
by the <script-symbol>. The <sort-rules> are optional and <sort-rules> not to be changed may be given by empty specifications.

The order of the script reordering statements rearranges the assignment of collation entries for the sets of collation entries identified by the <script-symbols> to the order that the <script-symbols> occur after the "reorder-scripts" statement.

The script reordering statements are terminated by a "reorder-scripts-end" statement.

Examples:
<LATIN> forward;forward
<ARABIC> forward;forward

4.1.2.12 "reorder-scripts-end" keyword

The "reorder-scripts-end" keyword shall specify the end of a list of script symbols, initiated by the "reorder-scripts" keyword.

The "i18n" FDCC-set LC_COLLATE category is defined in ISO/IEC 14651.

4.1.3 LC_MONETARY

The LC_MONETARY category defines 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 FDCC-set 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 (5) (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 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.
int_frac_digits	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using int_curr_symbol.
frac_digits	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using currency_symbol.
p_cs_precedes	An integer set to 1 if the currency_symbol or int_curr_symbol precedes the value for a nonnegative formatted monetary quantity, and set to 0 if the symbol succeeds the value.
p_sep_by_space	An integer set to 0 if no space separates the currency_symbol or int_curr_symbol from the value for a nonnegative formatted monetary quantity, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
n_cs_precedes	An integer set to 1 if the currency_symbol or int_curr_symbol precedes the value for a negative formatted monetary quantity, and set to 0 if the symbol succeeds the value.
n_sep_by_space	An integer set to 0 if no space separates the currency_symbol or int_curr_symbol from the value for a negative formatted monetary quantity, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
int_p_cs_precedes	An integer set to 1 if the int_curr_symbol or int_curr_symbol precedes the value for a nonnegative formatted monetary quantity, and set to 0 if the symbol succeeds the value.
int_p_sep_by_space	An integer set to 0 if no space separates the int_curr_symbol or int_curr_symbol from the value for a nonnegative formatted monetary quantity, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
int_n_cs_precedes	An integer set to 1 if the int_curr_symbol or int_curr_symbol precedes the value for a negative formatted monetary quantity, and set to 0 if the symbol succeeds the value.
int_n_sep_by_space	An integer set to 0 if no space separates the int_curr_symbol or int_curr_symbol from the value for a negative formatted monetary quantity, set to 1 if a space separates the symbol from the value, and set to 2 if a space separates the symbol and the sign string, if adjacent.
p_sign_posn	An integer set to a value indicating the positioning of the positive_sign for a nonnegative 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.
5 A space separates the symbol and the sign string, if adjacent.

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.
5 A space separates the symbol and the sign string, if adjacent.

The "i18n" FDCC-set is defined as follows for the LC_MONETARY category.

```
LC_MONETARY
int_curr_symbol      "<S><D><R><SP>"
currency_symbol      "<Cu>"
mon_decimal_point    "<,>"
mon_thousands_sep    "<.>"
mon_grouping         3;3
positive_sign        ""
negative_sign        "<->"
int_frac_digits      2
frac_digits          2
p_cs_precedes        1
p_sep_by_space       0
n_cs_precedes        1
n_sep_by_space       0
p_sign_posn          1
n_sign_posn          1
END LC_MONETARY
```

4.1.4 LC_NUMERIC

The LC_NUMERIC category defines 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 FDCC-set 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 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 non-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.

The i18n" FDCC-set is for the LC_NUMERIC category (in accordance with ISO 1000):

```
LC_NUMERIC
decimal_point       "<,>"
thousands_sep       "<.>"
grouping            "3;0"
END LC_NUMERIC
```

4.1.5 LC_TIME

The following mandatory keywords shall be recognized:

copy	Specify the name of an existing FDCC-set 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, referenced by the %a field descriptor. The operand shall consist of seven semicolon-separated strings. The first string shall be the abbreviated name 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, referenced by 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, referenced by 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, referenced by 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, referenced by 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.
d_fmt	Define the appropriate date representation, referenced by 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.
t_fmt	Define the appropriate time representation, referenced by 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.

am_pm	Define the appropriate representation of the ante meridiem and post meridiem strings, referenced by the %p field descriptor. The operand shall consist of two strings, separated by a semicolon. The first string shall represent the antemeridiem designation, the last string the postmeridiem designation.
t_fmt_ampm	Define the appropriate time representation in the 12-hour clock format with am_pm, referenced by 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 FDCC-set.

It is implementation defined whether the following optional keywords shall be recognized. If they are not supported, but present in a description file, 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.

4.1.5.1 Date Field Descriptors

The LC_TIME category defines the interpretation of a number of field descriptors (supported by the POSIX date utility, and intended for support in other interfaces). The field descriptors are also avialable in the definitions with the following LC_TIME keywords: d_t_fmt, d_fmt, t_fmt, t_fmt_ampm, era, and era_d_fmt. A field descriptor may not be used with the LC_TIME keywords defining it.

Table 4 - Escape sequences for the date field

%a	FDCC-set's abbreviated weekday name.
%A	FDCC-set's full weekday name.
%b	FDCC-set's abbreviated month name.
%B	FDCC-set's full month name.
%c	FDCC-set's appropriate date and time representation.
%C	Century (a year divided by 100 and truncated to integer) as decimal number (00-99).
%d	Day of the month as a decimal number (01-31).
%D	Date in the format mm/dd/yy.
%e	Day of the month as a decimal number (1-31 in at two-digit field with leading <space> fill).
%F	is replaced by the date in the format YYYY-MM-DD (ISO 8601 format)
%h	A synonym for %b.
%H	Hour (24-hour clock) as a decimal number (00-23).
%I	Hour (12-hour clock) as a decimal number (01-12).
%j	Day of the year as a decimal number (001-366).
%m	Month as a decimal number (01-12).
%M	Minute as a decimal number (00-59).
%n	A <newline> character.
%p	FDCC-set's equivalent of either AM or PM.
%r	12-hour clock time (01-12) using the AM/PM notation.

%S Seconds as a decimal number (00-61).
%t A <tab> character.
%T 24-hour clock time in the format HH:MM:SS.
%U Week number of the year (Sunday as the first day of the week) as a decimal number (00-53).
%w Weekday as a decimal number (0(Sunday)-6).
%W Week number of the year (Monday as the first day of the week) as a decimal number (00-53).
%x FDCC-set's appropriate date representation.
%X FDCC-set's appropriate time representation.
%y Year (offset from %C) as a decimal number (00-99).
%Y Year with century as a decimal number.
%Z Time-zone name, or no characters if no time zone is determinable.
%% A <percent-sign> character.

4.1.5.2 Modified Field Descriptors

Some field descriptors can be modified by the E and O modifier characters to indicate a different format or specification as specified in the LC_TIME FDCC-set description. If the corresponding keyword (see era, era_year, era_d_fmt, and alt_digits) is not specified or not supported for the current FDCC-set, the unmodified field descriptor value shall be used.

%Ec FDCC-set's alternate date and time representation.
%EC The name of the base year (period) in the FDCC-set's alternate representation.
%Ex FDCC-set's alternate date representation.
%Ey Offset from %EC (year only) in the FDCC-set's alternate representation.
%EY Full alternate year representation.
%Od Day of month using the FDCC-set's alternate numeric symbols.
%Oe Day of month using the FDCC-set's alternate numeric symbols.
%OH Hour (24-hour clock) using the FDCC-set's alternate numeric symbols.
%OI Hour (12-hour clock) using the FDCC-set's alternate numeric symbols.
%Om Month using the FDCC-set's alternate numeric symbols.
%OM Minutes using the FDCC-set's alternate numeric symbols.
%OS Seconds using the FDCC-set's alternate numeric symbols.
%OU Week number of the year (Sunday as the first day of the week) using the FDCC-set's alternate numeric symbols.
%Ow Weekday as number in the FDCC-set's alternate representation (Sunday=0).
%OW Week number of the year (Monday as the first day of the week) using the FDCC-set's alternate numeric symbols.
%Oy Year (offset from %C) in alternate representation.

The "i18n" FDCC-set is for the LC_TIME category (following ISO 8601):

```
LC_TIME
abday     "<7>;<1>;<2>;<3>;<4>;<5>;<6>"
day    "<7>;<1>;<2>;<3>;<4>;<5>;<6>"
abmon     "<1>;<2>;<3>;<4>;<5>;<6>;<7>;<8>;<9>;<1><0>;<1><1>;<1><2>"
mon       "<1>;<2>;<3>;<4>;<5>;<6>;<7>;<8>;<9>;<1><0>;<1><1>;<1><2>"
d_t_fmt  "<%><Y><-><%><m><-><%><d><T><%><T><SP><%><Z>"
d_fmt    "<%><Y><-><%><m><-><%><d>"
t_fmt     "<%><T>"
am_pm   "";""
t_fmt_ampm  ""
END LC_TIME
```

4.1.6 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 ISO/IEC 9945-2 (10) clause 2.8.4. The following keywords shall be recognized:

copy — Specify the name of an existing FDCC-set 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.

The "i18n" FDCC-set is for the LC_MESSAGES category:

```
LC_MESSAGES
yesexpr     "<+>"
noexpr      "<->"
END LC_MESSAGES
```

5. CHARMAP

Conforming implementations shall support the portable character set specified in Table 3. The table defines characters.

This standard places only the following requirements on the encoded values of the characters in the portable character set:

(1) The encoded values associated with each member of the portable character set shall be invariant across all FDCC-sets supported by the implementation.

(2) The encoded values associated with the digits ’0’ to ’9’ shall be such that the value of each character after ’0’ shall be one greater than the value of the previous character.

Conforming implementations shall support certain character and character set attributes, as defined in 5.1.

5.1 Character Set Description File

A character set description file shall exist for each coded character set supported by an implementation. This file is referred elsewhere in this standard as a charmap file.

Each character set description file shall define characteristics for the coded character set and the encoding for the characters specified in Table 3, and may define encoding for additional characters supported by the implementation. Other information about the coded character set can also be in the file. Coded character set character values shall be defined using symbolic character names followed by character encoding values.

Each symbolic name specified in Table 3 shall be included in the file and shall be mapped to a unique character value. In addition, the symbolic names in Table 4 shall be included in the file, but mapping of these symbolic names to character values is optional.

Table 4 - Control Character Set

<ACK>	<DC4>	<ESC>	<IS3>	<SO>
<CAN>	<DEL>	<ETB>	<IS4>	<SOH>
<DC1>	<DLE>	<ETX>	<NAK>	<STX>
<DC2>	<EM>	<IS1>	<NUL>	<SUB>
<DC3>	<EOT>	<IS2>	<SI>	<SYN>

The following declarations can precede the character definitions. Each shall consist of the symbol shown in the following list, starting in column 1, including the surrounding brackets, followed by one of more <blank>s, followed by the value to be assigned to the symbol. If any of the declarations are included, they shall be specified in the order shown in the following list:

<mb_cur_max>	Maximum number of bytes in a multibyte character. This shall default to 1.
<escape_char>	The escape character used for expressing numeric values, as described later in this subclause. This shall default to backslash (\), which is the character glyph used in all the following text and examples, unless otherwise noted.

The character set mapping definitions shall be all the lines immediately following an identifier line containing the string CHARMAP starting in column 1, and preceding a trailer line containing the string END CHARMAP starting in column 1. Empty lines and lines containing a number sign in the first column shall be treated as comment lines and ignored. Each noncomment line of the character set mapping definition (i.e., between the CHARMAP and END CHARMAP lines of the file) shall be in one of the following formats. The ellipsis symbol (...) stands for one or more byte representations of the same type as the one preceding it in the line.

"%s %c ...\n", <symbolic-name>, <value>, ...

"%s %c0%o ...\n", <symbolic-name>, <escape_char>, <value>, ...

"%s %cx%x ...\n", <symbolic-name>, <escape_char>, <value>, ...

"%s\n", <symbolic-name>

Each noncomment line of the character set mapping definition shall start with the symbolic name, immediately preceded by a <less-than> character and immediately followed by a <greater-than> character. Symbolic names shall only contain characters from the set shown with a visible glyph in Table 3. The <greater-than> character or the escape character can be included as part of the symbolic name by specifying it twice; for example, the sequence "<\\>>>" represents the symbolic name "\>". Symbolic names shall be unique and each symbolic name shall uniquely identify one value. Symbolic names may be included to identify values that can overlap with each other or with the values of the symbolic names shown in Table 3. It is possible to specify symbolic names for which no encoding exists in the encoded character set, by not specifying a value.

If the character represented by the symbolic name has an encoding value in the character set being defined, then the <symbolic name> shall be separated from the encoding value by a field separator (a <blank>). The encoding value shall be expressed as one or more octal or hexadecimal constants, each representing a byte, or as the character value itself. Octal constants shall be represented as one, two, or three octal digits, preceded by the escape character; for example, \141. Hexadecimal constants shall be represented by two hexadecimal digits, preceded by a backslash and a lowercase x; for example, \x61 or \x8f. Multibyte character encodings shall be represented by concatenating two or more octal or hexadecimal representations. For example, a two-byte value could be represented as "\xab\x21" or "\153\041"

6 REPERTOIREMAP

FDCC-set and Charmap sources may be specified in a coded character set independent way, using symbolic character names. The relation between the symbolic character names and characters may be specified via a Repertoiremap file, which defines the repertoire of characters defined for a FDCC-set, and the symbolic character names and corresponding abstract character (by a reference to ISO/IEC 10646 (11)).

The repertoire mapping is defined by specifying the symbolic character name and the ISO/IEC 10646 canonical encoding in hexadecimal form (with a preceding 'U') and optionally the long ISO/IEC 10646 character name. It is recommended to use, whenever possible, symbolic names for characters specified in ISO/IEC 9945-2 (10) Annex G. The symbolic character name and the ISO/IEC 10646 canonical encoding are each surrounded by angle brackets <>, and the fields shall be separated by one or more spaces or tabs on a line. If a right angle bracket or an escape character is used within a symbolic name, it shall be preceded by the escape character. The escape character can be redefined from the default reverse solidus (\) with the first line of the Repertoiremap containing the string "escape_char" followed by one or more spaces or tabs and then the escape character.

7 CONFORMANCE

A conforming implementation need to honour the follwing requirement:

COLL_WEIGHTS_MAX is minmum 7

Annex A. Differences from the POSIX standard (informative)

This standard is based on the locale and charmap specifications in the POSIX standard, and it intends to be backwards compatible, so that what is comformant to the POSIX standard is also conformant to this standard.

A number of enhancements have been done and a number of restrictions have been lifted in comparison to the POSIX standard:

1. Dependence on specific meaning of the character NUL as termination of a string (from the C standard) has been removed, to cater for other programming languages.

2. Dollar and number signs cannot have alternate representations.

3. A description of a "repertoiremap" definition file was added to facilitate descriptions of FDCC-sets without charmaps, and also to provide binding from a FDCC-set using one set of character names to charmaps using another naming set.

4. The specific POSIX locale has been replaced with the "i18n" FDCC-set.

5. Transliteration support has been added in the LC_CTYPE category.

6. Terminology has been aligned with ISO/IEC TR 11017, especially the POSIX term "locale" has been changed to "FDCC-set".

7. A date escape format %F has been added for ISO 8601 dates.

8. Added to LC_MONETARY to accomdate differences between locala and international formats:
int_p_cs_precedes
int_p-sep_by_space
int_n_cs_precedes
int_n_sep_by_space

9. script symbols have been added via the "script" keyword in the LC_COLLATE category.

10. The "order_start" keyword has got an optional script-symbol identifier

11. The keywords "reorder-scripts" and "reorder-scripts_end" have been introduce to reorder scripts.

12. Symbolic elipsises (both decimal and hexadecimal) has been introduced generally as a notation.

Annex B. Rationale (informative)

B.1 FDCC-set Rationale

The description of FDCC-sets is based on work performed in the UniForum Technical Committee Subcommittee on Internationalisation and on POSIX. Wherever appropriate, keywords were taken from the C Standard (9) or the POSIX-2 standard (10). The C and POSIX term "locale" has been changed into the term "FDCC-set" from ISO/IEC TR 11017 to align with that specification.

POSIX compiles locale sources into object files. 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 (ISO 646) 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.

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, an ISO/IEC 10646 system that has defined 16-bit bytes may require six octal, four hexadecimal, and five decimal digits, for some coded characters.

As an international (ISO/IEC) standard this standard should follow the ISO/IEC guidelines, including the ISO/IEC TR 10176. This TR has a rule that characters outside the invariant part of ISO/IEC 646 should not be used in portable specifications. The backslash and the number-sign character are not in the invariant part. As far as general usage of these symbols, they are covered by the "grandfather clause", but for newly defined interfaces, ISO has requested that specifications provide alternate representations, and this standard then follows POSIX for backward compatibility. 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.

B.1.1 LC_CTYPE Rationale

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. This standard 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 (9) character-class definition, the category does not support multicharacter elements. For 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 this standard 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.

B.1.2 LC_COLLATE Rationale.

The LC_COLLATE category governs the collation order in the FDCC-set, and thus the processing of the C Standard (9) strxfrm() and strcoll() functions, as well as a number of POSIX-2 (10) 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) 4 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 may 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 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 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.

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 ideograph can have different meanings and different pronunciations). Such collation, in general, falls 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 this standard 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. 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-IGNOREd characters. As an example, consider the two strings "o-ring" and "or-ing". Assuming the hyphen is IGNOREd 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 IGNOREd, and in the normal case the two strings would again compare equal. By taking position into account, the first collates before the second.

B.1.2.1 "reorder-after" rationale

Much work has already been done on FDCC-sets, and making them quite general. CEN/TC304 has on its programme of work the specification of a multilingual ordering for Europe, and also the collection of collating sequences of the different countries in Europe in a registry of cultural conventions; one of the formal specification techniques for this is using the POSIX standard (ISO/IEC 9945-2:1993). The POSIX standard introduces a copy command for all categories of the POSIX locale. This is good for many purposes and it ensures that two FDCC-sets are equivalent for this category. A further step in building on previous FDCC-set work is defined in this standard.

Collating sequences often vary a bit from country to country, and from language to language, but generally much of the collating sequence is the same. For instance the Danish sequence is for the most part the same as the German, English or French collation, but for about a dozen letters it differs. The same can be said for Swedish or Hungarian: generally the Latin collating sequence is the same, but a few characters are different.

With the advent of the quite general, coded character set independent locales like the example Danish locales in the POSIX Shell and Utilities standard (ISO/IEC 9945-2:1993) annex G, and the European multilingual POSIX collating specification, it will prove to be convenient if the few differences could be specified just as changes to an existing one. Using the "reorder-after" construct will also help improve the overview of what the changes really are for implementers and other users.

An example of the use of the "reorder-after" construct is the following. A default European ordering for the Latin alphabet may be adequate for Danish, with the exception of the collation rules for the letters Ü, ü, Æ, æ, Ä, ä, Ø, ø, Å and å. By applying the "reorder-after" construct, the Danish specification can be made more easily by copying and reordering the existing European specification, rather than specifying collation parameters for all Latin letters (with or without diacritics). There is no obligation for Denmark to take this approach, but the normative annex E provides the mechanism for doing so if it is deemed desirable.

B.1.2.2 awk script for "reorder-after" construct

A script has been written in the "awk" language defined in the POSIX standard ISO/IEC 9945-2 to implement the "reorder-after" construct:

```
BEGIN { comment = "%"; back[0]= follow[0] = 0; }
/LC_COLLATE/ { coll=1 }
/END LC_COLLATE/ { coll=0; for (lnr= 1; lnr; lnr= follow[lnr]) print c-
ont[lnr] }

{ if (coll == 0) print $0  ;
   else { if ($1 == "copy")   {
                  file = $2
                  while (getline < file )
                  if ( $1 == "LC_COLLATE" ) copy_lc = 1
                  else if ( $1 == "END" && $2 == "LC_COLLATE" ) copy_lc =0
                  else if (copy_lc) {
                      lnr++
                      follow[lnr-1] = lnr; back [ lnr ] = lnr-1
                      cont[lnr] = $0; symb[ $1 ] = lnr
                  }
                  close (file )
              }
              else if ($1 == "reorder-after") { ra=1 ; after = symb [ $2 ] }
              else if ($1 == "reorder-end") ra = 0
              else {
                  lnr++
                  if (ra) follow [ lnr ] = follow [ after ]
                  if (ra) back [ follow [ after ] ] = lnr
                  follow[after] = lnr; back [ lnr ] = after
                  cont[lnr] = $0
                  if ( ra && $1 != comment && $1 != "" )  {
                      old = symb [ $1 ];
                      follow [ back [ old ] ] = follow [ old ];
                      back [ follow [ old ] ] = back [ old ];
                      symb[ $1 ] = lnr;
                  }
                  after = lnr
              }
   }
}
```

B.1.2.3 Sample FDCC-set specification for Danish

```
escape_char /
comment_char %

% Danish language FDCC-set for Denmark
% Source: Danish Standards Association
% Address: Kollegievej 6,
%   DK-2920 Charlottenlund, Danmark
% Contact: Keld Simonsen
% Email: Keld.Simonsen@dkuug.dk
% Tel: +45 - 39966101
% Fax: +45 - 39966202
% Language: da
% Territory: DK
% Revision: 4.0
% Date: 1996-09-11
% Application: general
% Users: general
% Repertoiremap: mnemonic,DS
% Charset: ISO_8859-1:1987
% Distribution and use is free, also
% for commercial purposes.

% The ordering algorithm is in accordance
% with Danish Standard DS 377 (1980)
% and the Danish Orthography Dictionary
% (Retskrivningsordbogen, 1986).
% It is also in accordance with
% Greenlandic orthography.

LC_COLLATE
collating-element <A-A> from "<A><A>"
collating-element <A-a> from "<A><a>"
collating-element <a-A> from "<a><A>"
collating-element <a-a> from "<a><a>"
copy en_DK
reorder-after <CAPITAL>
<CAPITAL>
<CAPITAL-SMALL>
<SMALL-CAPITAL>
<SMALL>
reorder-after <q8>
<kk>            <Q>;<SPECIAL>;<SMALL>;IGNORE
reorder-after <t8>
<TH>            "<T><H>";"<TH><TH>";"<CAPITAL><CAPITAL>";IGNORE
```

```
<th>            "<T><H>";"<TH><TH>";"<SMALL><SMALL>";IGNORE
reorder-after <y8>
% <U:> and <U"> are treated as <Y> in Danish
<U:>            <Y>;<U:>;<CAPITAL>;IGNORE
<u:>            <Y>;<U:>;<SMALL>;IGNORE
<U">            <Y>;<U">;<CAPITAL>;IGNORE
<u">            <Y>;<U">;<SMALL>;IGNORE
reorder-after <z8>
% <AE> is a separate letter in Danish
<AE>            <AE>;<NONE>;<CAPITAL>;IGNORE
<ae>            <AE>;<NONE>;<SMALL>;IGNORE
<AE'>           <AE>;<ACUTE>;<CAPITAL>;IGNORE
<ae'>           <AE>;<ACUTE>;<SMALL>;IGNORE
<A3>            <AE>;<MACRON>;<CAPITAL>;IGNORE
<a3>            <AE>;<MACRON>;<SMALL>;IGNORE
<A:>            <AE>;<SPECIAL>;<CAPITAL>;IGNORE
<a:>            <AE>;<SPECIAL>;<SMALL>;IGNORE
% <O//> is a separate letter in Danish
<O//>           <O//>;<NONE>;<CAPITAL>;IGNORE
<o//>           <O//>;<NONE>;<SMALL>;IGNORE
<O//'>          <O//>;<ACUTE>;<CAPITAL>;IGNORE
<o//'>          <O//>;<ACUTE>;<SMALL>;IGNORE
<O:>            <O//>;<DIAERESIS>;<CAPITAL>;IGNORE
<o:>            <O//>;<DIAERESIS>;<SMALL>;IGNORE
<O">            <O//>;<DOUBLE-ACUTE>;<CAPITAL>;IGNORE
<o">            <O//>;<DOUBLE-ACUTE>;<SMALL>;IGNORE
% <AA> is a separate letter in Danish
<AA>            <AA>;<NONE>;<CAPITAL>;IGNORE
<aa>            <AA>;<NONE>;<SMALL>;IGNORE
<A-A>           <AA>;<A-A>;<CAPITAL>;IGNORE
<A-a>           <AA>;<A-A>;<CAPITAL-SMALL>;IGNORE
<a-A>           <AA>;<A-A>;<SMALL-CAPITAL>;IGNORE
<a-a>           <AA>;<A-A>;<SMALL>;IGNORE
<AA'>           <AA>;<AA'>;<CAPITAL>;IGNORE
<aa'>           <AA>;<AA'>;<SMALL>;IGNORE
reorder-end
END LC_COLLATE

LC_CTYPE
copy "en_DK"
END LC_CTYPE

LC_MONETARY
int_curr_symbol        "<D><K><K><SP>"
currency_symbol        "<k><r>"
mon_decimal_point      "<,>"
mon_thousands_sep      "<.>"
```

```
mon_grouping          3;3
positive_sign         ""
negative_sign         "<->"
int_frac_digits       2
frac_digits           2
p_cs_precedes         1
p_sep_by_space        2
n_cs_precedes         1
n_sep_by_space        2
p_sign_posn           4
n_sign_posn           4
END LC_MONETARY

LC_NUMERIC
decimal_point         "<,>"
thousands_sep         "<.>"
grouping              3;3
END LC_NUMERIC

LC_TIME
abday     "<s><o//><n>";"<m><a><n>";/
          "<t><i><r>";"<o><n><s>";/
          "<t><o><r>";"<f><r><e>";/
          "<l><o//><r>"
day       "<s><o//><n><d><a><g>";/
          "<m><a><n><d><a><g>";/
          "<t><i><r><s><d><a><g>";/
          "<o><n><s><d><a><g>";/
          "<t><o><r><s><d><a><g>";/
          "<f><r><e><d><a><g>";/
          "<l><o//><r><d><a><g>"
abmon     "<j><a><n>";"<f><e><b>";/
          "<m><a><r>";"<a><p><r>";/
          "<m><a><j>";"<j><u><n>";/
          "<j><u><l>";"<a><u><g>";/
          "<s><e><p>";"<o><k><t>";/
          "<n><o><v>";"<d><e><c>"
mon       "<j><a><n><u><a><r>";/
          "<f><e><b><r><u><a><r>";/
          "<m><a><r><t><s>";/
          "<a><p><r><i><l>";/
          "<m><a><j>";/
          "<j><u><n><i>";/
          "<j><u><l><i>";/
          "<a><u><g><u><s><t>";/
          "<s><e><p><t><e><m><b><e><r>";/
          "<o><k><t><o><b><e><r>";/
```

```
           "<n><o><v><e><m><b><e><r>";/
           "<d><e><c><e><m><b><e><r>"
d_t_fmt     "<%><a><SP><%><d><SP><%><b><SP><%><Y><SP><%><T><SP-
><%><Z>"
d_fmt       "<%><d><-><%><m><-><%><Y>"
t_fmt       "<%><T>"
am_pm       "";""
t_fmt_ampm  ""
END LC_TIME

LC_MESSAGES
yesexpr     "<<(><1><J><j><Y><y><)/>><.><*>"
noexpr      "<<(><0><N><n><)/>><.><*>"
END LC_MESSAGES
```

B.1.3 LC_MONETARY Rationale.

The currency symbol does not appear in LC_MONETARY because it is not defined in the C Standard's (9) C locale. The C Standard (9) limits the size of decimal points and thousands delimiters to single-byte values. In FDCC-sets based on multibyte coded character sets this cannot be enforced, obviously; this standard does not prohibit such characters, but makes the behaviour unspecified (in the text "In contexts where other standards . . . ").

The grouping specification is based on, but not identical to, the C Standard (9). The "-1" signals that no further grouping shall be performed, the equivalent of (CHAR_MAX) in the C Standard (9)).

The FDCC-set definition is an extension of the C Standard (9) 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:

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

1	+1.25 $	+1.25 $ +1.25$
2	1.25$ +	1.25 $+ 1.25$+
3	1.25+ $	1.25 +$ 1.25+$
4	1.25$ +	1.25 $+ 1.25$+
5 ???????		

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 (9) 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 (9).

Mon_grouping	Formatted Value	C 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.

B.1.4 LC_NUMERIC Rationale.

See the rationale for LC_MONETARY (4.1.3.1) for a description of the behaviour of grouping.

B.1.5 LC_TIME Rationale.

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 FDCC-sets. 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, day, 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 FDCC-set, 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. 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.

B.1.6 LC_MESSAGES Rationale.

The LC_MESSAGES category is described in 1.8.1 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 FDCC-set definition file, is not specified by this version of this standard. The POSIX 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.

B.2 Character Set Rationale.

This standard poses no requirement that multiple character sets or code sets be supported, leaving this as a marketing differentiation for implementors. Although multiple charmap files are supported, it is the responsibility of the implementation to provide the file(s); if only one is provided, only that one will be accessible.

The character set description file provides:

(1) the capability to describe character set attributes (such as collation order or character classes) independent of character set encoding, and using only the characters in the portable character set. This makes it possible to create ‘‘generic’’ FDCC-set source files for all code sets that share the portable character set (such as the ISO 8859 family or IBM Extended ASCII).

(2) standardized symbolic names for all characters in the portable character set, making it possible to refer to any such character regardless of encoding.

Implementations are free to describe more than one code set in a character set description file, as long as only one encoding exists for the characters in Table 3. For example, if an implementation defines ISO 8859-1 as the primary code set, and ISO 8859-2 as an alternate set, with each character from the alternate code set preceded in data by a shift code, a character set description file could contain a complete description of the primary set and those characters from the secondary that are not identical, the encoding of the latter including the shift code.

Implementations are free to choose their own symbolic names, as long as the names identified by this standard are also defined; this provides support for already existing ‘‘character names.’’

The names selected for the portable character set follows the ISO 8859 standard, with the following exceptions:

(1) The traditional UNIX system names are used for control characters.

(2) The word ‘‘slash’’ is preferred over ‘‘solidus.’’

(3) The word ‘‘backslash’’ is preferred over ‘‘reverse-virgule.’’

(4) The word ‘‘underscore’’ is preferred over ‘‘low-line.’’

(5) The word ‘‘percent’’ is preferred over ‘‘percent-sign.’’

(6) The names of the digits are preferred over the numbers, to avoid possible confusion between ‘‘0’’ and ‘‘O’’, and between ‘‘1’’ and ‘‘l’’ (one and ell).

The names for the control characters in Table 4 were taken from ISO 4873.

The charmap file was introduced to resolve problems with the portability of, especially, FDCC-set sources. While the portable character set (in Table 3) is a constant across all FDCC-sets for a particular implementation, this is not true for the extended character set. However, the particular coded character set used for an application or an implementation does not necessarily imply different characteristics or collation: on the contrary, these attributes should in many cases be identical, regardless of codeset. The charmap provides the capability to define a common FDCC-set definition for multiple codesets (the same FDCC-set source can be used for codesets with different extended characters; the ability in the charmap to define ‘‘empty’’ names allows for characters missing in certain codesets).

In addition, some implementors have expressed an interest in using the charmap file to define certain other characteristics of codesets, such as the <mb_cur_max> value for the particular codeset. (Note that <mb_cur_max> has to be equal to or lower than the C Standard (9) {MB_LEN_MAX}, which is the implementation limit). Such extensions are not described here; but may be added in a later revision of this standard.

The <escape_char> declaration was added at the request of the international community to ease the creation of portable charmap files on terminals not implementing the default backslash escape. (This approach was adopted because this is a new interface invented by POSIX-2 (10). Historical interfaces, such as the shell command language and awk, have not been modified to accommodate this type of terminal.)

The octal number notation was selected to match those of POSIX awk and tr utilities and is consistent with that used by the POSIX localedef utility.

The charmap capability implements a facility available at some X/Open compatible implementations. Its prime virtue is to support ‘‘generic’’ collation sequence source definitions. An implementor or an applications developer can produce a template definition that can be used to produce several codeset-dependent ‘‘compiled’’ FDCC-set definitions. The facility also removes any dependency in many source definitions on characters outside the character set defined in this clause.

As a reference, the following is the portable filename character set from 1003.1:

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

Annex C. INDEX

Not available yet.