IEEE P1003.2 Draft 11.2 − September 1991

2.4 Character Set

Table 2-3 – Character Set and Symbolic Names

Symbolic Name	Glyph	Symbolic Name	Glyph	Symbolic Name	Glyph
<NUL>		<colon>	:	<circumflex>	^
<alert>		<semicolon>	;	<circumflex-accent>	^
<backspace>		<less-than-sign>	<	<underscore>	_
<tab>		<equals-sign>	=	<low-line>	_
<newline>		<greater-than-sign>	>	<grave-accent>	`
<vertical-tab>		<question-mark>	?	<a>	a
<form-feed>		<commercial-at>	@	<b>	b
<carriage-return>		<A>	A	<c>	c
<space>		<B>	B	<d>	d
<exclamation-mark>	!	<C>	C	<e>	e
<quotation-mark>	"	<D>	D	<f>	f
<number-sign>	#	<E>	E	<g>	g
<dollar-sign>	$	<F>	F	<h>	h
<percent-sign>	%	<G>	G	<i>	i
<ampersand>	&	<H>	H	<j>	j
<apostrophe>	'	<I>	I	<k>	k
<left-parenthesis>	(	<J>	J	<l>	l
<right-parenthesis>	)	<K>	K	<m>	m
<asterisk>	*	<L>	L	<n>	n
<plus-sign>	+	<M>	M	<o>	o
<comma>	,	<N>	N	<p>	p
<hyphen>	-	<O>	O	<q>	q
<hyphen-minus>	-	<P>	P	<r>	r
<period>	.	<Q>	Q	<s>	s
<full-stop>	.	<R>	R	<t>	t
<slash>	/	<S>	S	<u>	u
<solidus>	/	<T>	T	<v>	v
<zero>	0	<U>	U	<w>	w
<one>	1	<V>	V	<x>	x
<two>	2	<W>	W	<y>	y
<three>	3	<X>	X	<z>	z
<four>	4	<Y>	Y	<left-brace>	{
<five>	5	<Z>	Z	<left-curly-bracket>	{
<six>	6	<left-square-bracket>	[	<vertical-line>	|
<seven>	7	<backslash>	\	<right-brace>	}
<eight>	8	<reverse-solidus>	\	<right-curly-bracket>	}
<nine>	9	<right-square-bracket>	]	<tilde>	~

Conforming implementations shall support one or more coded character sets. Each supported coded character set shall include the *portable character set* specified in Table 2-3. The table defines the characters in the portable character set and the corresponding symbolic character names used to identify each character in a character set description file. The names are chosen to correspond closely with character names defined in other international standards. The table contains more than one symbolic character name for characters whose traditional name differs from the chosen name.

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

(1) If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, the results achieved by an application accessing those locales are unspecified.

(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.

(3) A null character, NUL, which has all bits set to zero, shall be in the set of characters.

Conforming implementations shall support certain character and character set attributes, as defined in 2.5.1.

2.4.1 Character Set Description File

Implementations shall provide a character set description file for at least one coded character set supported by the implementation. These files are referred to elsewhere in this standard as *charmap* files. It is implementation defined whether or not users or applications can provide additional character set description files. If such a capability is supported, the system documentation shall describe the rules for the creation of such files.

Each character set description file shall define characteristics for the coded character set and the encoding for the characters specified in Table 2-3, and may define encoding for additional characters supported by the implementation. Other information about the coded character set may 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 2-3 shall be included in the file and shall be mapped to a unique encoding value (except for those symbolic names that are shown with identical glyphs). If the control characters commonly associated with the symbolic names in Table 2-4 are supported by the implementation, the symbolic names and their corresponding encoding values shall be included in the file. Some of the values associated with the symbolic names in this table also may be contained in Table 2-3.

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.

`<code_set_name>` The name of the coded character set for which the character set description file is defined. The characters of the name shall be taken from the set of characters with

Table 2-4 – Control Character Set

<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>	1
<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>	1
<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>	1
<CAN>	<DEL>	<ETB>	<IS1>	<RS>	<SYN>	1
<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>	1
<DC1>	<EM>	<FF>	<IS3>	<SO>	<VT>	1

visible glyphs defined in Table 2-3. 1

`<mb_cur_max>` The maximum number of bytes in a multibyte character. This shall default to 1.

`<mb_cur_min>` An unsigned positive integer value that shall define the minimum number of bytes in a character for the encoded character set. The value shall be less than or equal to `mb_cur_max`. If not specified, the minimum number shall be equal to `mb_cur_max`.

`<escape_char>` The escape character used to indicate that the characters following shall be interpreted in a special way, as defined 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.

`<comment_char>` The character, that when placed in column 1 of a charmap line, is used to indicate that the line shall be ignored. The default character shall be the number-sign (#).

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 `comment_char` in the first column shall be 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 either of two forms:

`"%s %s %s\n"`, *<symbolic-name>* , *<encoding>* , *<comments>*

or

`"%s. . .%s %s %s\n"`, *<symbolic-name>* , *<symbolic-name>* , *<encoding>* , *<comments>*

In the first format, the line in the character set mapping definition defines a single symbolic name and a corresponding encoding. A symbolic name is one or more characters from the set shown with visible glyphs in Table 2-3, enclosed between angle brackets. A character following an escape character shall be interpreted as itself; for example, the sequence ‘`<\\\>>`’ represents the symbolic name ‘`\>`’

enclosed between angle brackets.

In the second format, the line in the character set mapping definition defines a range of one or more symbolic names. In this form, the symbolic names shall consist of zero or more nonnumeric characters from the set shown with visible glyphs in Table 2-3, followed by an integer formed by one or more decimal digits. The characters preceding the integer shall be identical in the two symbolic names, and the integer formed by the digits in the second symbolic name shall be equal to or greater than the integer formed by the digits in the first name. This shall be interpreted as a series of symbolic names formed from the common part and each of the integers between the first and the second integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

A character set mapping definition line shall exist for all symbolic names specified in Table 2-3, and shall define the coded character value that corresponds with the character glyph indicated in the table, or the coded character value that corresponds with the control character symbolic name. If the control characters commonly associated with the symbolic names in Table 2-4 are supported by the implementation, the symbolic name and the corresponding encoding value shall be included in the file. Additional unique symbolic names may be included. A coded character value can be represented by more than one symbolic name.

The encoding part shall be expressed as one (for single-byte character values) or more concatenated decimal, octal, or hexadecimal constants in the following formats:

"%cd%d", *<escape_char>* , *<decimal byte value>*

"%cx%x", *<escape_char>* , *<hexadecimal byte value>*

"%c%o", *<escape_char>* , *<octal byte value>*

Decimal constants shall be represented by two or three decimal digits, preceded by the escape character and the lowercase letter d; for example, \d05, \d97, or \d143. Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape character and the lowercase letter x; for example, \x05, \x61, or \x8f. Octal constants shall be represented by two or three octal digits, preceded by the escape character; for example, \05, \141, or \217. In a portable charmap file, each constant shall represent an 8-bit byte. Implementations supporting other byte sizes may allow constants to represent values larger than those that can be represented in 8-bit bytes, and to allow additional digits in constants. When constants are concatenated for multibyte character values, they shall be of the same type, and interpreted in byte order from left to right. The manner in which constants are represented in the character is implementation defined. Omitting bytes from a multibyte character definition produces undefined results.

In lines defining ranges of symbolic names, the encoded value is the value for the first symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic names defined by the range shall have encoding values in increasing order. For example, the line

```
    <j0101>...<j0104>       \d129\d254
```

shall be interpreted as

```
    <j0101>                 \d129\d254
    <j0102>                 \d129\d255
    <j0103>                 \d130\d0
    <j0104>                 \d130\d1
```

The comment is optional.

For the interpretation of the dollar-sign and the number-sign, see 2.2.2.37 and 2.2.2.93.

2.4.2 Character Set Rationale. *(This subclause is not a part of P1003.2)*

The portable character set is listed in full so there is no dependency on the ISO/IEC 646 {1} (or historically ASCII) encoded character set, although the set is identical to the characters defined in the International Reference Version of ISO/IEC 646 {1}.

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 using the `localedef` utility's `−f` option (although in the case of just one file on the system, `−f` is not useful).

The statement about invariance in code sets for the portable character set is worded as it is to avoid precluding implementations where multiple incompatible code sets are available (say, ASCII and EBCDIC). The standard utilities cannot be expected to produce predictable results if they access portable characters that vary on the same implementation.

The character set description file provides:

— 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’’ `localedef` source files for all code sets that share the portable character set (such as the ISO 8859 family or IBM Extended ASCII).

— 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 2-3. For example, if an implementation defines ISO 8859-1 {5} as the primary code set, and ISO 8859-2 {6} 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 members of the portable character set follow the ISO 8859 {5} and the ISO/IEC 10646 {B11} standards. However, several commonly used UNIX system names occur as synonyms in the list:

— The traditional UNIX system names are used for control characters.

— The word ‘‘slash’’ is in addition to ‘‘solidus.’’

— The word ‘‘backslash’’ is in addition to ‘‘reverse-solidus.’’

— The word ‘‘hyphen’’ in addition to ‘‘hyphen-minus.’’

— The word ‘‘period’’ in addition to ‘‘full-stop.’’

— For the digits, the word ‘‘digit’’ is eliminated.

— For letters, the words ‘‘Latin Capital Letter’’ and ‘‘Latin Small Letter’’ are eliminated.

— The words ‘‘left-brace’’ and ‘‘right-brace’’ in addition to ‘‘left-curly-bracket’’ and ‘‘right-curly-bracket.’’

— 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 the letter ell).

The names for the control characters in Table 2-4 were taken from ISO 4873 {4}.

The charmap file was introduced to resolve problems with the portability of, especially, `localedef` sources. This standard assumes that the portable character set is constant across all locales, but does not prohibit implementations from supporting two incompatible codings, such as both ASCII and EBCDIC. Such ‘‘dual-support’’ implementations should have all charmaps and `localedef` sources encoded using one portable character set, in effect ‘‘cross-compiling’’ for the other environment. Naturally, charmaps (and `localedef` sources) are only portable without transformation between systems using the same encodings for the portable character set. They can, however, be transformed between two sets using only a subset of the actual characters (the portable 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 code set. The charmap provides the capability to define a common locale definition for multiple code sets (the same `localedef` source can be used for code sets with different extended characters; the ability in the charmap to define ‘‘empty’’ names allows for characters missing in certain code sets).

In addition, several implementors have expressed an interest in using the charmap concept to provide the information required for support of multiple character sets. Examples of such information is encoding mechanism, string parsing rules, default font information, etc. Such extensions are not described here.

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. Historical interfaces, such as the shell command language and `awk`, have not been modified to accommodate this type of terminal.) The `<comment_char>` declaration was added at the request of the international community to eliminate the potential confusion between the number sign and the pound sign.

The octal number notation with no leading zero required was selected to match 1
those of `awk` and `tr` and is consistent with that used by `localedef`. To avoid 1
confusion between an octal constant and the backreferences used in `localedef` 1
source, the octal, hexadecimal, and decimal constants must contain at least two 1
digits. As single-digit constants are relatively rare, this should not impose any 1
significant hardship. Each of the constants includes ‘‘two or more’’ digits to 1
account for systems in which the byte size is larger than eight bits. For example, 1
a Unicode system that has defined 16-bit bytes may require six octal, four hexade- 1
cimal, and five decimal digits. 1

The decimal notation is supported because some newer international standards define character values in decimal, rather than in the old column/row notation.

The charmap identifies the coded character sets supported by an implementation. At least one charmap must be provided, but no implementation is required to provide more than one. Likewise, implementations can allow users to generate new charmaps (for instance for a new version of the 8859 family of coded character sets), but does not have to do so. If users are allowed to create new charmaps, the system documentation must describe the rules that apply (for instance: ‘‘only coded character sets that are supersets of ISO/IEC 646 {1} IRV, no multibyte characters, etc.’’)

2.5 Locale

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

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

Conforming implementations shall provide the standard utilities and the inter- 1
faces in Annex B (if that option is supported) with the capability to modify their 1
behavior based on the current locale, as defined in the Environment Variables 1
subclause for each utility and interface. 1

Locales other than those supplied by the implementation can be created via the `localedef` utility (see 4.35), provided that the {POSIX2_LOCALEDEF} symbol is defined on the system; see 2.13.2. Otherwise, only the implementation-provided locale(s) can be used. The input to the utility is described in 2.5.2. The value that shall be used to specify a locale when using environment variables shall be the string specified as the *name* operand to the `localedef` utility when the locale was created. The strings `"C"` and `"POSIX"` are reserved as identifiers for the POSIX Locale (see 2.5.1.) When the value of a locale environment variable begins with a slash (`/`), it shall be interpreted as the pathname of the locale definition. If the value of the locale value does not begin with a slash, the mechanism used to locate the locale is implementation defined.

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