-- --

IEEE P1003.2 Draft 11.2 – September 1991

`"%s,△%s△%d,△%d:%.2d\n"`, *<weekday>*, *<month>*, *<day>*, *<hour>*, *<min>*

To show π written to 5 decimal places:

`"pi△=△%.5f\n"`, *<value of π>*

To show an input file format consisting of five colon-separated fields:

`"%s:%s:%s:%s:%s\n"`, *<arg1>*, *<arg2>*, *<arg3>*, *<arg4>*, *<arg5>*

2.13 Configuration Values

2.13.1 Symbolic Limits

This clause lists magnitude limitations imposed by a specific implementation. The braces notation, {LIMIT}, is used in this standard to indicate these values, but the braces are not part of the name.

Table 2-16 – Utility Limit Minimum Values

Name	Description	Value
{POSIX2_BC_BASE_MAX}	The maximum *obase* value allowed by the `bc` utility.	99
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the `bc` utility.	2048
{POSIX2_BC_SCALE_MAX}	The maximum *scale* value allowed by the `bc` utility.	99
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the `bc` utility.	1000
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the LC_COLLATE `order` keyword in the locale definition file; see 2.5.2.2.3.	2
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the `expr` utility.	32
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing `<newline>`.	2048
{POSIX2_RE_DUP_MAX}	The maximum number of repeated occurrences of a regular expression permitted when using the interval notation `\{`*m*`,` *n*`\}`; see 2.8.3.3.	255
{POSIX2_VERSION}	This value indicates the version of the utilities in this standard that are provided by the implementation. It will change with each published version of this standard.	199???

The values specified in Table 2-16 represent the lowest values conforming implementations shall provide; and consequently, the largest values on which an application can rely without further enquiries, as described below. These values shall be accessible to applications via the `getconf` utility (see 4.26) and through the interfaces described in 7.8.2, [such as *sysconf* () in the C binding]. The literal names shown in the table apply only to the `getconf` utility; the high-level-language binding shall describe the exact form of each name to be used by the interfaces in that binding.

Implementations may provide more liberal, or less restrictive, values than shown in Table 2-16. These possibly more liberal values are accessible using the symbols in Table 2-17.

Table 2-17 – Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum *obase* value allowed by the `bc` utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the `bc` utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum *scale* value allowed by the `bc` utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the `bc` utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the LC_COLLATE `order` keyword in the locale definition file; see 2.5.2.2.3.	{POSIX2_COLL_WEIGHTS_MAX}
{EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the `expr` utility.	{POSIX2_EXPR_NEST_MAX}
{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing `<newline>`.	{POSIX2_LINE_MAX}
{RE_DUP_MAX}	The maximum number of repeated occurrences of a regular expression permitted when using the interval notation `\{`*m*`,` *n*`\}`; see 2.8.3.3.	{POSIX2_RE_DUP_MAX}

The functions in 7.8.2 [such as *sysconf* () in the C binding] or the `getconf` utility shall return the value of each symbol on each specific implementation. The value so retrieved shall be the largest, or most liberal, value that shall be available throughout the session lifetime, as determined at session creation. The literal names shown in the table apply only to the `getconf` utility; the high-level-language binding shall describe the exact form of each name to be used by the interfaces in that binding.

All numerical limits defined by POSIX.1 {8}, such as {PATH_MAX}, also apply to this standard. (See POSIX.1 {8} 2.8.) All the utilities defined by this standard are implicitly limited by these values, unless otherwise noted in the utility descriptions.

It is not guaranteed that the application can in fact push a value to the implementation's specified limit in any given case, or at all, as a lack of virtual memory or other resources may prevent this. The limit value indicates only that the implementation does not specifically impose any arbitrary, more restrictive limit.

2.13.1.1 Symbolic Limits Rationale. *(This subclause is not a part of P1003.2)*

This clause grew out of an idea that originated in POSIX.1 {8}, in the form of *sysconf*() and *pathconf*(). (In fact, the same person wrote the original text for both standards.) The idea is that a Strictly Conforming POSIX.2 Application can be written to use the most restrictive values that a minimal system can provide, but it shouldn't have to. The values shown in Table 2-17 represent compromises so that some vendors can use historically-limited versions of UNIX system utilities. They are the highest values that Strictly Conforming POSIX.2 Applications or Conforming POSIX.2 Applications can assume, given no other information.

However, by using `getconf` or *sysconf*(), the elegant application can tailor itself to the more liberal values on some of the specific instances of specific implementations.

There is no explicitly-stated requirement that an implementation provide finite limits for any of these numeric values; the implementation is free to provide essentially unbounded capabilities (where it makes sense), stopping only at reasonable points such as {ULONG_MAX} (from the C Standard {7} via POSIX.1 {8}). Therefore, applications desiring to tailor themselves to the values on a particular implementation need to be ready for possibly huge values; it may not be a good idea to blindly allocate a buffer for an input line based on the value of {LINE_MAX}, for instance. However, unlike POSIX.1 {8}, there is no set of limits in this standard that return a special indication meaning ''unbounded.'' The implementation should always return an actual number, even if the number is very large.

The statement

> ''It is not guaranteed that the application ...

is an indication that many of these limits are designed to ensure that implementors design their utilities without arbitrary constraints related to unimaginative programming. There are certainly conditions under which combinations of options can cause failures that would not render an implementation nonconforming. For example, {EXPR_NEST_MAX} and {ARG_MAX} could collide when expressions are large; combinations of {BC_SCALE_MAX} and {BC_DIM_MAX} could exceed virtual memory.

In POSIX.2, the notion of a limit being guaranteed for the process lifetime, as it is in POSIX.1 {8}, is not as useful to a shell script. The `getconf` utility is probably a process itself, so the guarantee would be valueless. Therefore, POSIX.2 requires the guarantee to be for the session lifetime. This will mean that many vendors will either return very conservative values or possibly implement `getconf` as a built-in.

It may seem confusing to have limits that apply only to a single utility grouped into one global clause. However, the alternative, which would be to disperse them out into their utility description clauses, would cause great difficulty when *sysconf*() and `getconf` were described. Therefore, the working group chose the global approach.

Each language binding could provide symbol names that are slightly different than are shown here. For example, the C binding prefixes the symbols with a leading underscore.

The following comments describe selection criteria for the symbols and their values.

{ARG_MAX}
: This is defined by POSIX.1 {8}. Unfortunately, it is very difficult for a portable application to deal with this value, as it does not know how much of its argument space is being consumed by the user's environment variables.

{BC_BASE_MAX}
{BC_DIM_MAX}
{BC_SCALE_MAX}
: These were originally one value, {BC_SCALE_MAX}, but it was unreasonable to link all three concepts into one limit.

{CHILD_MAX}
: This is defined by POSIX.1 {8}.

{CUT_FIELD_MAX}
: This value was removed from an earlier draft. It represented the maximum length of the *list* argument to the `cut -c` or `-f` options. Since the length is now unspecified, the utility should have to deal with arbitrarily long lists, as long as {ARG_MAX} is not exceeded.

{CUT_LINE_MAX}
: This value was removed from an earlier draft. Historical `cut`s have had input line limits of 1024; this removal therefore mandates that a conforming `cut` shall process files with lines of unlimited length. 1

{DEPTH_MAX}
: This directory-traversing depth limit (which at one time applied to `rm` and `find`) was removed from an earlier draft for two major reasons:

(1) It could be a security problem if utilities searching for files could not descend below a published depth; this would be a semi-reliable means of hiding files from the administrator.

(2) There is no reason a reasonable implementation should have to limit itself in this way.

{ED_FILE_MAX}

This value was removed from an earlier draft. Historical `ed`s have had very small file limits; since {ED_FILE_MAX} is no longer specified, implementations have to document the limits as described in 2.11. It is recommended that implementations set much more reasonable file size limits as they modify `ed` to deal with other features required by POSIX.2.

{ED_LINE_MAX}

This value was removed from an earlier draft. Historical `ed`s have had small input line limits; this removal therefore mandates that a conforming `ed` shall process files with lines of length {LINE_MAX}.

{COLL_WEIGHTS_MAX}

The weights assigned to `order` can be considered as ''passes'' through the collation algorithm.

{EXPR_NEST_MAX}

The value for expression nesting was borrowed from the C Standard {7}.

{FIND_DEPTH_MAX}

This was removed from an earlier draft in favor of a common value, {DEPTH_MAX}.

{FIND_FILESYS_MAX}

This was removed from an earlier draft. It indicated the limit of the number of file systems that `find` could traverse in its search. It was dropped because this standard does not really acknowledge the historical nature of separate file systems.

{FIND_NEWER_MAX}

This value, which allowed `find` to limit the number of `-newer` operands it processed, was deleted from an earlier draft. It was felt to be a vestige of a particular implementation with an incorrect programming algorithm that should not limit applications.

{JOIN_LINE_MAX}

This value was removed from an earlier draft. Historical `join`s have had input line limits of 1024; this removal therefore mandates that a conforming `join` shall process files with lines of length {LINE_MAX}.

{LINE_MAX}

This is a global limit that affects all utilities, unless otherwise noted. The {MAX_CANON} value from POSIX.1 {8} may further limit input lines from terminals. The {LINE_MAX} value was the subject of much debate and is a compromise between those who wished unlimited lines and those who understood that many historical utilities were written with fixed buffers. Frequently, utility writers selected the UNIX system constant BUFSIZ to allocate these buffers; therefore, some utilities were limited to 512 bytes for I/O lines, while others achieved 4096 or greater.

It should be noted that {LINE_MAX} applies only to input line length; there is no requirement in the standard that limits the length of output lines. Utilities such as `awk`, `sed`, and `paste` could theoretically construct lines longer than any of the input lines they received, depending on the options used or the instructions from the application. They are not required to truncate their output to {LINE_MAX}. It is the responsibility of the application to deal with this. If the output of one of those utilities is to be piped into another of the standard utilities, line lengths restrictions will have to be considered; the `fold` utility, among others, could be used to ensure that only reasonable line lengths reach utilities or applications.

{LINK_MAX}
This is defined by POSIX.1 {8}.

{LP_LINE_MAX}
This value was removed from an earlier draft. Since so little is being required for the details of the `lp` utility, it made little sense to specify how long its output lines are. Thus, implementations of `lp` will be expected to deal with lines up to {LINE_MAX}, but whether those lines print sensibly on every device is unspecified.

{MAX_CANON}
This is defined by POSIX.1 {8}.

{MAX_INPUT}
This is defined by POSIX.1 {8}.

{NAME_MAX}
This is defined by POSIX.1 {8}.

{NGROUPS_MAX}
This is defined by POSIX.1 {8}.

{OPEN_MAX}
This is defined by POSIX.1 {8}.

{PATH_MAX}
This is defined by POSIX.1 {8}.

{PIPE_BUF}
This is defined by POSIX.1 {8}.

{RM_DEPTH_MAX}
This was removed from an earlier draft in favor of a common value, {DEPTH_MAX}.

{RE_DUP_MAX}
The value selected is consistent with historical practice.

{SED_PATTERN_MAX}
This symbolic value, the size of the `sed` pattern space, was replaced by a specific value in the `sed` description. It is unlikely that any real application would ever need to access this value symbolically.

{SORT_LINE_MAX}
This was removed from an earlier draft. Now that `cut` and `fold` can handle unlimited-length input lines, a special long input line limit for `sort` is not needed.

There are different limits associated with command lines and input to utilities, depending on the method of invocation. In the case of a C program *exec*-ing a utility, {ARG_MAX} is the underlying limit. In the case of the shell reading a script and *exec*-ing a utility, {LINE_MAX} limits the length of lines the shell is required to process and {ARG_MAX} will still be a limit. If a user is entering a command on a terminal to the shell, requesting that it invoke the utility, {MAX_INPUT} may restrict the length of the line that can be given to the shell to a value below {LINE_MAX}.

2.13.2 Symbolic Constants for Portability Specifications

Table 2-18 – Optional Facility Configuration Values

Name	Description
{POSIX2_C_BIND}	The C language development facilities in Annex A support the C Language Bindings Option (see Annex B).
{POSIX2_C_DEV}	The system supports the C Language Development Utilities Option (see Annex A).
{POSIX2_FORT_DEV}	The system supports the FORTRAN Development Utilities Option (see Annex C).
{POSIX2_FORT_RUN}	The system supports the FORTRAN Runtime Utilities Option (see Annex C).
{POSIX2_LOCALEDEF}	The system supports the creation of locales as described in 4.35.
{POSIX2_SW_DEV}	The system supports the Software Development Utilities Option (see Section 6).

Table 2-18 lists symbols that can be used by the application to determine which optional facilities are present on the implementation. The functions defined in 7.8.2 [such as *sysconf*()] or the `getconf` utility can be used to retrieve the value of each symbol on each specific implementation. The literal names shown in the table apply only to the `getconf` utility; the high-level-language binding shall describe the exact form of each name to be used by the interfaces in that binding.

Each of these symbols shall be considered valid names by the implementation. Each shall be defined on the system with a value of 1 if the corresponding option is supported; otherwise, the symbol shall be undefined.

2.13.2.1 Symbolic Constants for Portability Specifications Rationale. *(This subclause is not a part of P1003.2)*

When an option is supported, `getconf` returns a value of 1. For example, when C development is supported:

```
if [ "$(getconf POSIX2_C_DEV)" -eq 1 ]; then
        echo C supported
fi
```

The *sysconf* () function in the C binding would return 1.

The following comments describe selection criteria for the symbols and their values.

{POSIX2_C_BIND}
{POSIX2_C_DEV}
{POSIX2_FORT_DEV}
{POSIX2_SW_DEV}

These were renamed from _POSIX_* in Draft 9 after it was pointed out that each of the POSIX standards should keep generally in its own namespace.

It is possible for some (usually privileged) operations to remove utilities that support these options, or otherwise render these options unsupported. The header files, the *sysconf* () function, or the `getconf` utility will not necessarily detect such actions, in which case they should not be considered as rendering the implementation nonconforming. A test suite should not attempt tests like:

```
rm /usr/bin/c89
getconf POSIX2_C_DEV
```

{_POSIX_LOCALEDEF}

This symbol was introduced to allow implementations to restrict supported locales to only those supplied by the implementation.

Section 3: Shell Command Language

The shell is a command language interpreter. This section describes the syntax of that command language as it is used by the `sh` utility and the functions in 7.1 [such as *system* () and *popen* () in the C binding].

The shell operates according to the following general overview of operations. The specific details are included in the cited clauses and subclauses of this section. The shell:

(1) Reads its input from a file (see `sh` in 4.56), from the `-c` option, or from one of the functions in 7.1. If the first line of a file of shell commands starts with the characters #!, the results are unspecified.

(2) Breaks the input into tokens: words and operators. (See 3.3.)

(3) Parses the input into simple (3.9.1) and compound (3.9.4) commands.

(4) Performs various expansions (separately) on different parts of each command, resulting in a list of pathnames and fields to be treated as a command and arguments (3.6).

(5) Performs redirection (3.7) and removes redirection operators and their operands from the parameter list.

(6) Executes a function (3.9.5), built-in (3.14), executable file, or script, giv-
ing the name of the command (or, in the case of a function within a 1
script, the name of the script) as the ''zero'th'' argument and the remain- 1
ing words and fields as parameters (3.9.1.1).

(7) Optionally waits for the command to complete and collects the exit status (3.8.2).

3.0.1 Shell Command Language Rationale. *(This subclause is not a part of P1003.2)*

The System V shell was selected as the starting point for this standard. The BSD C-shell was excluded from consideration, for the following reasons:

(1) Most historically portable shell scripts assume the Version 7 ''Bourne'' shell, from which the System V shell is derived.

(2) The majority of tutorial materials on shell programming assume the System V shell.

Despite the selection of the System V shell, the developers of the standard did not limit the possibilities for a shell command language that was upward-compatible.