-- --

IEEE P1003.2 Draft 11.2 – September 1991

B.10 C Binding for Get POSIX Configurable Variables

B.10.1 C Binding for Get String-Valued Configurable Variables

Function: *confstr* ()

B.10.1.1 Synopsis

```
#include <unistd.h>
size_t confstr(int name , char *buf, size_t len );
```

B.10.1.2 Description

The *confstr* () function provides a method for applications to get configuration-defined string values. Its use and purpose are similar to the *sysconf* () function defined in POSIX.1 {8}, but it is used where string values rather than numeric values are returned.

The *name* argument represents the system variable to be queried. The implementation shall support all of the *name* values shown in Table B-18, which are defined in `<unistd.h>`. It may support others.

Table B-18 – `confstr()` *name* Values

***name* Value**	**String returned by `confstr()`**
_CS_PATH	A value for the **PATH** environment variable that finds all standard utilities.

If *len* is not zero, and if *name* has a configuration-defined value, *confstr* () shall copy that value into the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes, including the terminating null, then *confstr* () shall truncate the string to *len*−1 bytes and null-terminate the result. The application can detect that the string was truncated by comparing the value returned by *confstr* () with *len*.

If *len* is zero and *buf* is **NULL**, then *confstr* () still shall return the integer value as defined below, but shall not return a string. If *len* is zero but *buf* is not **NULL**, the result is unspecified.

B.10.1.3 Returns

If *name* does not have a configuration-defined value, *confstr* () shall return zero and leave *errno* unchanged.

If *name* has a configuration-defined value, the *confstr* () function shall return the size of buffer that would be needed to hold the entire configuration-defined value.

If this return value is greater than *len*, the string returned in *buf* has been truncated.

B.10.1.4 Errors

If any of the following conditions occur, *confstr*() shall return zero and set *errno* to the corresponding value:

[EINVAL] The value of the *name* argument is invalid.

B.10.1.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

An application can distinguish between an invalid *name* parameter value and one that corresponds to a configurable variable that has no configuration-defined value by checking if *errno* has been modified. This mirrors the behavior of *sysconf*() in POSIX.1 {8}.

The original need for this function was to provide a way of finding the configuration-defined default value for the environment variable **PATH**. Since **PATH** can be modified by the user to include directories that could contain utilities replacing POSIX.2 standard utilities, applications need a way to determine the system-supplied **PATH** environment variable value that contains the correct search path for the POSIX.2 standard utilities.

An application could use `confstr(name,NULL,(size_t) 0)` to find out how big a buffer is needed for the string value, *malloc*() a buffer to hold the string, and call *confstr*() again to get the string. Alternately, it could allocate a fixed, static buffer that is big enough to hold most answers (512 bytes, maybe, or 1024), but then *malloc*() a larger buffer if it finds that this is too small.

History of Decisions Made

In Draft 7, these values and *sysconf*() values defined in POSIX.1 {8} were obtained using a function named *posixconf*(). However, that routine was dropped in favor of *csysconf*(). There did not seem to be any reason to provide the redundant interface to POSIX.1 {8} functions, nor to return values as strings when numeric values are really what are needed. *csysconf*() could be extended to return strings for other related standards or features.

In Draft 9, *csysconf*() has been replaced by *confstr*(). The name was changed because too many people were confused by the name; they thought that the 'c' referred to the C language, rather than characters (as distinct from integers). The *confstr*() function also copies the returned string into a buffer supplied by the application instead of returning a pointer to a string. This allows a cleaner interface in some implementations (lightweight processes were mentioned), and resolves questions about when the application must copy the string returned.

B.10.2 C Binding for Get Numeric-Valued Configurable Variables

Functions: *sysconf* (), *pathconf* (), *fpathconf* ()

A system that supports the C Language Bindings Option shall support the C language bindings defined in POSIX.1 {8} for the *sysconf* (), *pathconf* (), and *fpathconf* () functions. Of the *name* values defined in POSIX.1 {8}, only those that correspond to numeric-valued configuration values listed in Table 7-1, are required by POSIX.2. In addition, the *sysconf* () function shall support the *name* values in Table B-19, defined in `<unistd.h>`, to provide values for values in 2.13.1.

Table B-19 – C Bindings for Numeric-Valued Configurable Variables

Symbolic Limit	*name* Value
{BC_BASE_MAX}	_SC_BC_BASE_MAX
{BC_DIM_MAX}	_SC_BC_DIM_MAX
{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
{BC_STRING_MAX}	_SC_BC_STRING_MAX
{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
{LINE_MAX}	_SC_LINE_MAX
{RE_DUP_MAX}	_SC_RE_DUP_MAX
{POSIX2_VERSION}	_SC_2_VERSION
{POSIX2_C_DEV}	_SC_2_C_DEV
{POSIX2_FORT_DEV}	_SC_2_FORT_DEV
{POSIX2_FORT_RUN}	_SC_2_FORT_RUN
{POSIX2_LOCALEDEF}	_SC_2_LOCALEDEF
{POSIX2_SW_DEV}	_SC_2_SW_DEV

B.10.3 Rationale. *(This subclause is not a part of P1003.2)*

In Draft 9, the *name* values corresponding to the _POSIX2_* symbolic limits were changed to more closely follow the convention used in POSIX.1 {8}. In POSIX.1 {8}, for example, the *name* value for {_POSIX_VERSION} is _SC_VERSION. The POSIX.2 *name* value for {_POSIX2_C_DEV} (actually, it was {_POSIX_C_DEV} in Draft 8) was _SC_POSIX_C_DEV, and is now _SC_2_C_DEV.

If `sysconf(_SC_2_VERSION)` is not equal to the value of the {_POSIX2_VERSION} symbolic constant (see B.2.2), the utilities available via *system* () or *popen* () might not behave as described in this standard. This would mean that the application is not running in an environment that conforms to POSIX.2. Some applications might be able to deal with this, others might not. However, the interfaces defined in Annex B shall continue to operate as specified, even if `sysconf(_SC_2_VERSION)` reports that the utilities no longer perform as specified.

B.11 C Binding for Locale Control

The C binding to the services described in 7.9 shall be the *setlocale* () function defined in POSIX.1 {8} 8.1.2. In addition to the category values defined in POSIX.1 {8}, *setlocale* () shall also accept the value LC_MESSAGES, which shall be defined in `<locale.h>`.

B.11.1 C Binding for Locale Control Rationale. *(This subclause is not a part of P1003.2)*

The order in which the various locale categories are processed by *setlocale* () is not specified by POSIX.1 {8}, so the place for LC_MESSAGES in that order is also unspecified.