IEEE Standards Interpretation for IEEE Std 1003.1™-1990 IEEE Standard for Information Technology--Portable Operating System Interfaces (POSIX®)

Interpretation Request #27
Topic: _POSIX_VDISABLE symbol **Relevant Sections:** 2.9.4

A question has been raised with regard to the requirements POSIX.1 places on the form of the value defined for the _POSIX_VDISABLE symbol. Does the Standard require that _POSIX_VDISABLE be a preprocessor number? For example, an implementation might use the value (unsigned char)255, which the C preprocessor won't compare to -1.

_POSIX_VDISABLE is listed as an "Execution-Time Symbolic Constant". Must it also be usable in numerical comparisons in the preprocessor? The constants that are guaranteed to be usable at compile time are listed separately, as "Compile- Time Symbolic Constants".

My reading is that though it would be nice if one could use this value at compile time, and though the authors might have intended that it be usable this way, the standard does not guarantee it. From POSIX.1 (2.9.4, page 38, lines 1129 ff.):

The constants in Table 2-11 may be used by the application, at execution time, to determine which optional facilities are present and what actions shall be taken by the implementation ...

Under the implementation example described above, the proper way to use a constant from table 2.11 is to use #ifdef to see whether it's defined in <unistd.h>, but to do a numerical comparison only at run time.

Interpretation Response
The standard does not require that _POSIX_VDISABLE be a preprocessor number. The standard does not require that _POSIX_VDISABLE be usable in numeric comparisons in

the preprocessor.

Rationale for Interpretation
The standard makes no requirement that the constant _POSIX_VDISABLE be a preprocessor number. The requirements relating this constant in section 2.9.4 relate only to use at execution time.

It is understandable why an application might like to be able to use _POSIX_VDISABLE as a preprocessor constant. The wording in section 2.9.4:

If any of the constants in Table 2-11 are defined to have value -1 in the header .... can suggest, on casual reading, code like the following to minimize size and optimize efficiency for each implementation:

```
#ifdef _POSIX_VDISABLE
#if _POSIX_VDISABLE == -1
      /* code that assumes no vdisable capability */
#else
     /* code that assumes vdisable capability */
#endif
#else
     /* code that uses pathconf() to determine vdisable capability */
#endif
```

However, there is no wording in the standard to actually back up that suggestion, and silence on the part of the standard means no requirement.

There are reasons why an implementor might want to define a value that is not a preprocessor number, such as including a type cast to avoid problems in comparing the value to a member of the c_cc array member of a termios struct (which is constrained by the standard to be an unsigned integer type). Since no wording in the standard prohibits this, it is implicitly permitted.

Thus, rather than the above fragment, an implementation could include code like:

```
#ifdef _POSIX_VDISABLE
      if (_POSIX_VDISABLE == -1) {
          /* code that assumes no vdisable capability */
      } else {
          /* code that assumes vdisable capability */
      }
#else
        /* code that uses pathconf() to determine vdisable capability */
#endif
```

Of course it is generally simplest, though potentially less efficient, to just write the code that uses pathconf().