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

Interpretation Request #54
Topic: extern int errno **Relevant Sections:** 2.4 **Classification:** No change

On page 23, section 2.4, on line 502, the POSIX standard specifies the attributes that are required for the 'errno' variable. The lines states: extern int errno;

This is expressed as a specific 'C' declaration. I believe that this is overly specific, and only represents a short hand notation. Other areas within this standard do not use this format, as in the 'stat' structure in section 5.6.1.

I believe that the description should be spelled out as being an expression representing an assignable integer variable with global scope or expands to a modifiable lvalue of type int

This will permit the declaration that is specified, while also permitting other implementations which satisfy the same need. To get the values that are returned in this variable, <errno.h> must be included, and that is typically where the 'errno' variable is typically defined.

This same topic has also come up in the POSIX.4a working group and is mentioned in the latest drafts in section 2.5.2. They propose specific alteration of the standard; I propose an interpretation that permits flexiability in the declaration and may also solve the problem the P1003.4 group is attempting to solve.

Attached is a note that I received from one of the members of the P1003.1 Working Group. It gives a different perspective on the problem and the history of the text in the standard. (Paul Wanish)

Text of Note
The POSIX C bindings should use exactly the ANSI C definition of errno, which says that errno "expands to a modifiable lvalue of type int." This ALLOWS an implementation to do something like:

#define errno (*__errno_val)
extern int *__errno_val;

but also allows the more traditional definition of just "extern int errno". The main difference between POSIX.1-1990 and ANSI C is that POSIX explicitly allows an application to contain the statement: extern int errno;

I believe that this would even be allowed if the application didn't include <errno.h>. This is different than ANSI C, and means:
(1) A straight ANSI C application still is OK under POSIX.1.
(2) An implementation that sticks with ANSI C is OK under POSIX.1.
(3) An application that includes "extern int errno" is OK under POSIX.1 but not ANSI C.
(4) An implementation that defines errno as a preprocessor define is OK under ANSI C but not POSIX.1.

This definition was put into POSIX.1-1988 primarily to accomodate Common Usage C implementations and existing applications. There existed (and still exist?) many historical applications that didn't bother to include <errno.h>, but just defined errno themselves. This was common practice, and the committee could not reach a concensus on requiring them to change. (Also, frankly, I don't think that anyone had any idea of why it might be a good idea to define things the ANSI way, at least in the U**X world.) The "no substantive change" rule precluded this being changed in -1990.

However, time has passed. ANSI compilers are edging out older compilers, and the disadvantages of the POSIX definition of errno is becoming clear. It's time to change it.

Interpretation Response
IEEE Std 1003.1-1990 specifies that extern int errno; is the correct declaration for errno. This is not a shorthand for a different meaning.

Rationale for Interpretation
The standard means exactly what it says. This issue has been resolved in the manner suggested by the requester in IEEE Draft Standard P1003.1a, which is now in ballot.