IEEE Standards Interpretations for IEEE Std 1003.1™-2001 IEEE Standard for Information Technology - Portable Operating System Interface (POSIX®)

Interpretation Request #3

Topic: Defect in XSH dlsym **Relevant Sections:** dlsym

The "dlsym()" function is defined as returning "the address of a symbol". According to the Example, "dlsym ()can be used to access either function or data objects". The return type of dlsym is a "void *". In order to actually call a function using the pointer returned by "dlsym()", the pointer must be converted from a "void *" type to a pointer to function type. However, the ISO C Standard says that converting a "void *" type to a pointer to function type results in undefined behavior. At least one major compiler reports a warning when attempting such a conversion, even with an explicit cast.

I would like a new function added that returns the address of a function object symbol and whose return type is a function pointer. For example: fptr_t dlsym_f(void *restrict handle, const char *restrict name); Standard C requires that a pointer to a function of one type may be converted to a pointer to a function of another type and back again, and the result shall compare equal to the original pointer. Therefore, "fptr_t" can be defined as any pointer to function type. The "dlsym()" function could continue to be used for data objects. As an extension, implementations that allow converting from "void *" to pointer to function types could continue to allow using "dlsym()" in addition to "dlsym_f()" for function objects. This would allow backwards compatability, while offering a more portable option for new code.

Interpretation Response

The standard clearly states the requirements for dlsym(), and conforming implementations must conform to this. The dlsym() function is marked as part of the X/Open System Interfaces Extension (by the XSI margin marking). Systems conforming to the X/Open Systems Interfaces Extension are indeed required to be able to convert between function pointers and void * data pointers without losing data. This is most clearly stat-

ed in the description of the va_arg() macro in the description of (XBD, P310, L11109-11119) where the XSI shading on L11119 indicates that implementations supporting the X/Open System Interfaces Extension are required to be able to handle pointers to any type of object, not just pointers to data. The XSI margin marking description clearly states that requirements like this are extensions to the requirements of the C Standard. In this case, the 1999 C Standard does indeed require a warning to be issued for the function call shown in the dlsym() examples section on XSH P259, L8566. An equivalent form of this call: *(void **)(&fptr) = dlsym(handle, "my_function"); does not generate compiler warnings and will work correctly on all systems supporting the X/Open System Interfaces Extension.

Rationale for Interpretation
See above response.

Notes to the Editor (not part of this interpretation)
In a TC or revision of the standard make the following changes: 1. XSH P259, L8566 (EXAMPLES): Change from: fptr = (int (*)(int))dlsym(handle, "my_function"); to: *(void **)(&fptr) = dlsym(handle, "my_function"); 2. XSH P260, L8590 (RATIONALE): Change from: None. to: The C Standard does not require that pointers to functions can be cast back and forth to pointers to data. Indeed, the C Standard does not require that an object of type void * can hold a pointer to a function. Systems supporting the X/Open System Interfaces Extension, however, do require that an object of type void * can hold a pointer to a function. The result of converting a pointer to a function into a pointer to another data type (except void *) is still undefined, however. Note that compilers conforming to the C Standard are required to generate a warning if a conversion from a void * pointer to a function pointer is attempted as in: fptr = (int (*)(int))dlsym(handle, "my_function");