IEEE Standards Interpretations for IEEE Std 1003.2™-1992 IEEE Standard for Information Technology--Portable Operating System Interfaces (POSIX®)--Part 2: Shell and Utilities

Interpretation Request #142
Topic: printf **Relevant Sections:** 4.50.7

In section 4.50.7, page 396, lines 8241-8243, the standard states, "The argument operands shall be treated as strings if the corresponding conversion character is b, c, or s; otherwise, it shall be evaluated as a C constant, as described by the C Standard {7}, with the following extensions:" In section 4.50.7, page 396, lines 8248-8252, the standard states, "If an argument operand cannot be completely converted into an internal value appropriate to the corresponding conversion specification, a diagnostic message shall be written to standard error and the utility shall not exit with a zero exit status, but shall continue processing any remaining operands and shall write the value accumulated at the time the error was detected to standard output." Q0. Do the words "cannot be completely converted into an internal value appropriate to the corresponding conversion" preclude extensions? In other words, could an implementation allow operands to be C constant expressions? For example, could pprintf "%d\n" 3+4 yield 7?

If you read this first paragraph as a requirement that the implementation is required to handle C constants, and the second paragraph as describing what an implementation must do when it cannot interpret the operand, then this would be legal. If you read the first paragraph as only C constants can be given, then this would have to be an error. I strongly recommend the first interpretation. There are several other questions that these sentences don't seem to answer: Q1. Does "otherwise" in the first paragraph refer only >to conversion characters specified by this standard. Can an implementation add a conversion character that does not process its operands as C constants? Q2. Does an implementation that does not support floating >point conversions need to handle floating point constants? Q3. Can an implementation handle integer constants larger >than MAXINT as an extension? Q4. What does it mean by the value accumulat-

ed at the >time that the error was detected? The standard doesn't specify the order of processing the characters. For example, with %d, the value 123x456 could output 123, 456, 1230000, or any other value depending on how the value is converted.

Interpretation Response
Q0. The standard does not speak to this issue, and no conformance distinction can be made between alternative implementations based on this. Implementations are free to evaluate in an implementation defined manner as an extension to the standard. Q1. "Otherwise", refers to other conversion characters specified in this standard. An implementation can add additional characters as an extension. Q2. No, an implementation that does not support floating point conversions is not required to handle floating point constants. Q3. Page 95 Section 2.9.1 requires that integer values be treated as C signed long data types, therefore LONG_MAX is the limit not MAXINT for %d. For %u it is an unsigned value (ULONG_MAX). Q4. The standard does not speak to this issue and as such no conformance distinction can be made between alternative implementations based on this.

Rationale for Interpretation
None.