JTC1/SC22/WG14: N#### (N2823 revision 2)

Title:	Freestanding C and IEC 60559 conformance scope reduction proposal
Author:	CFP
Date:	2022-01-20

Version 1:	• Original paper (N2823)
Version 2:	• Added in more motivation based on feedback from WG14 • Structural changes (Paper header, change log, etc.) • Removed parts voted into C23 • Shift to using FENV_ACCESS for the main problem

1. Introduction:

The incorporation of TS-18661 parts 1-3 into C23 brought in more requirements for freestanding C if the freestanding C implementation also conformed to the IEC 60559 parts brought in by the TS. This was indicated through defining some feature test macros (__STDC_IEC_60559_BFP__ and__STDC_IEC_60559_DFP__). Without defining those macros, the original C freestanding support remained. It is only when defining the macros that the implementation has extra functionality it needs to support. But by doing so, some people believe that unintended features are also pulled into being required for freestanding, even though they believe the intent was not to do so.

Latest revision description:

This is an update for N2823 as requested by WG14 in the 2021/11 meeting.

All changes requested by WG14 were made. They include: More text for motivation and rationale, removal of what was voted in already, and addition of relative diffs (from/to aspects of the suggested changes to the standard).

2. Motivation

Some members of the WG21 committee seem to be concerned with the expansion in scope in freestanding environments in C23 due to the changes brought in from the integration of TS 18661. This paper tries to address the concerns brought forward in WG14 reflector messages 19444 and 19445.

In reflector message 19445, a request was made to have the strtod* functions allow using exceptions to indicate error conditions. This would allow not providing thread local errno if the implementation so desires.

In reflector message 19444, concerns about requiring errno were brought up. Also having locale sensitive issues like whitespace and decimal separators for the numeric conversion functions brings in locales to the freestanding space. The addition of errno requires thread local storage, and the locale issue reduces portability.

The addition of thread local storage is a problem in general since freestanding applications often do not want to have "global" storage overhead or cannot have it. Locales have a similar issue in that most freestanding applications do not deal with locales (and hence the associated storage needed for them).

The general floating-point environment (described in 7.6) also has thread storage duration and hence requires overhead in terms of "global" storage. This was not directly mentioned in the reflector messages but is a conceivable concern that needs to be addressed.

3. Counter claims and workarounds

Since the addition of errno (along with the floating point environment) and locales are often problematic in freestanding implementations, those implementations can simply not define __STDC_IEC_60559_BFP__ and __STDC_IEC_60559_DFP__.

The argument often made is that that limits the portability of code in implementations that want to support IEEE floating point for freestanding, but can't due to the requirement for thread local storage and locale issues. This is problematic in at least a couple of cases. First, the problem of portability of locale support is already present in hosted implementations so this is not a new problem in general (though it can be for freestanding that wants to support the new floating point specifications). Second, in freestanding implementations, often threads are not supported anyways so the problem of having thread local storage decays into having extra global storage. While this does reduce the problem scope, it is still something that a number of freestanding implementations could consider a burden as global storage is often expensive or unavailable.

Reflector message 19444 had an underspecified recommendation to separate out language and library parts to allow the library to be supported only with new macro definitions which would end up not requiring thread local storage and locale support if the library macros were not defined. While this would work, the point of having the language and library parts present was to conform to the IEC standard. Separating them out would seem to miss the point of having the features there in the first place.

Although this was not brought forward by WG14 or WG21, another alternative would be to not have IEC 60559 binding at all for freestanding C as there may not be any demand for this case. CFP however believes it is needed so it was part of the original TS's and now in the C standard.

4. Proposed changes description

Freestanding programs are not required to accept strictly conforming programs that use features of <errno.h> or <locale.h>. This means such programs do not need to allow access to the variable errno or the locale, even if they provide functions whose specification references errno or locales. That should alleviate the concerns brought forward from the reflector.

The floating point environment is another problematic area that was not mentioned in the reflector. Unless the floating-point environment access is turned on (or is on by default) through the FENV_ACCEESS pragma there is no need for thread local storage for the floating-point environment. Since it can be turned off with user code or it is off as part of the default state, no changes need to be made to the standard to address the concerns brought up in the reflector as well as additional concerns that flow from those initial concerns.

The other problem brought forward during the analysis of the concerns above included the requirement for errno to be used for error reporting in the string conversion functions. Since some freestanding implementations may not have errno or exceptions, there should be a way to allow those implementations to still support the functions. Allowing no form of error reporting at all can be used in freestanding implementations to allow the functions to be used, though with the drawback that there is no way to portably determine if an error occurred during the conversion. This is proposed as an additional independent change that should allow existing code that checks the individual error reporting mechanisms to continue to work.

Finally, allowing exceptions to be used, as opposed to purely relying on errno as an error indicator would allow the same error handling mechanisms for the floating-point string conversion functions as the rest of the math functions. The last set of changes attempts to do this.

5. Proposed changes text

5.1 Proposed change #1 - allow no error reporting for freestanding implementations:

Change 7.12#20 from:

(... math_errhandling ...) expands to an expression that has type int and the value **MATH_ERRNO**, **MATH_ERREXCEPT**, or the bitwise OR of both.

to:

(... math_errhandling ...) expands to an expression that has type int and the value **MATH_ERRNO**, **MATH_ERREXCEPT**, ~~or~~ the bitwise OR of both, or **0**; the value shall not be **0** in a hosted implementation.

5.2 Proposed change #2 (independent of proposed change #1) – Allow exceptions for error reporting:

In N2596 change the strto{d,f,ld} functions in 7.22.1.5#10 from:

The functions return the converted value, if any. If no conversion could be performed, zero is returned. If the correct value overflows and default rounding is in effect (7.12.1), plus or minus HUGE_VAL, HUGE_VALF, or HUGE_VALL is returned (according to the return type and sign of the value), and the value of the macro ERANGE is stored in errno. If the result underflows (7.12.1), the functions return a value whose magnitude is no greater than the smallest normalized positive number in the return type; whether errno acquires the value ERANGE is implementation-defined.

to:

The functions return the converted value, if any. If no conversion could be performed, zero is returned.

If the correct value overflows and default rounding is in effect (7.12.1), plus or minus HUGE_VAL, HUGE_VALF, or HUGE_VALL is returned (according to the return type and sign of the value); if the integer expression **math_errhandling & MATH_ERRNO** is nonzero, the integer expression **errno** acquires the value **ERANGE**; if the integer expression **math_errhandling & MATH_ERREXCEPT** is nonzero, the "overflow" floating-point exception is raised.

If the result underflows (7.12.1), the functions return a value whose magnitude is no greater than the smallest normalized positive number in the return type; if the integer expression **math_errhandling & MATH_ERRNO** is nonzero, whether **errno** acquires the value **ERANGE** is implementation defined; if the integer expression **math_errhandling & MATH_ERREXCEPT** is nonzero, whether the "underflow" floating-point exception is raised is implementation-defined.

For the analogous wide character functions:

Change 7.29.4.1.1#10 from:

The functions return the converted value, if any. If no conversion could be performed, zero is returned. If the correct value overflows and default rounding is in effect (7.12.1), plus or minus **HUGE_VAL**, **HUGE_VALF**, or **HUGE_VALL** is returned (according to the return type and sign of the value), and the value of the macro **ERANGE** is stored in **errno**. If the result underflows (7.12.1), the functions return a value whose magnitude is no greater than the smallest normalized positive number in the return type; whether **errno** acquires the value **ERANGE** is implementation-defined.

to:

The functions return the converted value, if any. If no conversion could be performed, zero is returned.

If the correct value overflows and default rounding is in effect (7.12.1), plus or minus **HUGE_VAL**, **HUGE_VALF**, or **HUGE_VALL** is returned (according to the return type and sign of the value); if the integer expression **math_errhandling & MATH_ERRNO** is nonzero, the integer expression **errno** aquires the value **ERANGE**; if the integer expression **math_errhandling & MATH_ERREXCEPT** is nonzero, the "overflow" floating-point exception is raised.

If the result underflows (7.12.1), the functions return a value whose magnitude is no greater than the smallest normalized positive number in the return type; if the integer expression **math_errhandling & MATH_ERRNO** is nonzero, whether **errno** acquires the value **ERANGE** is implementation defined; if the integer expression **math_errhandling & MATH_ERREXCEPT** is nonzero, whether the "underflow" floating-point exception is raised is implementation-defined.

Change 4#7 as follows from:

The strictly conforming programs that shall be accepted by a conforming freestanding implementation that defines __STDC_IEC_60559_BFP__ or __STDC_IEC_60559_DFP__ may also use features in the contents of the standard headers <fenv.h> and and <math.h> and the numeric conversion functions (7.22.1) of the standard header <stdlib.h>. All identifiers that are reserved when <stdlib.h> is included in a hosted implementation are reserved when it is included in a freestanding implementation.

to:

The strictly conforming programs that shall be accepted by a conforming freestanding implementation that defines __STDC_IEC_60559_BFP__ or __STDC_IEC_60559_DFP__ may also use features in the contents of the standard headers <fenv.h> and <math.h> and the strto* floating-point numeric conversion functions (7.22.1) of the standard header <stdlib.h>, provided the program does not set the state of the **FENV_ACCESS** pragma to "**ON**". All identifiers that are reserved when <stdlib.h> is included in a hosted implementation are reserved when it is included in a freestanding implementation.