

Proposal for C2Y WG N3942

Title: Annex F Introduction - Better Organize F.2.2, F.3 and F.10.1 +++
Author: CFP Study Group
Date: October 15, 2026
Proposal Category: Editorial + one minor technical
Reference: N3854

Rationale

Full support for signaling NaNs is what is defined by the **Recommended Practice** of F.2.2. Is this correct?

Very recent updates to C2Y has made references to C2Y's Table F.2 as being Table F.1. These incorrect references have been rectified. **Hopefully, we have corrected them all?**

References to quiet and signaling NaNs are now better organized and better explained in the quest for clarity. Paragraphs dealing with NaNs from F.3 and F.10.1 now appear in F.2.2. Also, while Annex F has already been enhanced to cater for decimal types, previously forgotten decimal infinities and NaNs are now included.

Multiple explanations of the same thing with slightly different words now only appear once. A case in point in that text which notes how a family of functions is described.

Variants of the same phrase are now made more consistent, the phrase "unless stated otherwise" and its nearly ten variants being a case in point.

More than one use of a word in the one sentence with different meanings in each case are avoided, the word **provide** being a case in point.

Unlike other ISO/IEC 60559 mandated operations seen in Table F.1, both

getDecimalRoundingDirection and **setDecimalRoundingDirection**

appeared in a paragraph in F.3 rather than Table F.1 like their counterparts:

getBinaryRoundingDirection and **setBinaryRoundingDirection**

That is now fixed and those functions now appear in the table.

F.3#1 implies that the specifications for **<math.h>** functions appears in Annex F. In fact, most specifications appear in the main body. It is now made far more obvious that only the specifications for rounding behavior, special cases and floating-point exceptions appear in Annex F.

The first and last sentences in F.3#20, as well as all of F.10.1#6 and F.10.1#15, effectively refer to the tables mentioned in paragraph F.3#1. These sentences are now merged into F.3#1 to make it more thorough and avoid the need to cross reference back to F.3#1 from elsewhere.

The first sentences in **F.3#20** deal with correct rounding (or not) issues of entries in Table F.4 (which immediately follows this clause). The second of these also mentions things which are really **Recommended Practice** but having those sentences in a paragraph with a separate heading noting this. Many pages away from Table F.4, the whole paragraph F.10.1#16 repeats those F.3#20 things (and more) but does so under a heading of **Recommended Practice**. In the interest of clarity and cohesion:

- The most important statement of the original F.3#20, i.e., that C does not mandate correct rounding for the functions of Table F.4, is retained as the sole statement in F.3#20;
- Issues to do with recommended practice for the functions of Table F.4 are moved to additional paragraphs under the heading of **Recommended Practice** which needs to be added;
- The first and last sentences in F.3#20 are already covered by the updated F.3#1 so they disappear; and
- All of paragraph F.10.1#16 has now been incorporated in F.3 (in what is F.3#2[012]) and can be deleted.

It is recommended that the new F.3#21 and F.3#22 paragraphs now follow Table F.4 in F.3."

The use in F.10.1 of the term ISO/IEC 60559 *rounding mode* is a misnomer. It is an ISO/IEC 60559 rounding direction attribute. The C2Y term *rounding direction* (5.3.5.3.3#23) is now used instead.

Within this Annex, and F.10 in particular, the phrases **<math.h>** *function* and *library function* appear. The former is the more common. The later, more generic, term appears three (3) times in F.10.1. Those three cases were changed to **<math.h>** *function* to avoid confusion.

Mention of whether the functions are odd or even is generally done where the function in question is discussed, i.e. where specifications appear. However, this is not the case with **atan2** and **atan2pi** which is unusual. This has been rectified.

Finally,

1. Some paragraphs which appear to discuss more than one concept have now been split for readability.
2. Various paragraphs have been rearranged to appear in more logical order.
3. Confusing pluralized text is made singular for clarity where appropriate.
4. The new rationalised rounding direction terminology is used throughout.
5. Open-ended domains are now defined explicitly.
6. Redundant and overlapping items have been merged.
7. Some sentences have been tightened.
8. Clunky wording is refined, e.g. F.10.1#4.
9. Punctuation is improved in places.

Note-to-Editor

- See recommendation about F.3#21.
 - Paragraph renumbering due to a deleted paragraph may be left to the editor;
 - New footnotes need to be numbered which means renumbering of footnotes;
 - Tables have been added so one needs to be renumbered and references to it fixed;
 - Table headings which need to change appear below for clarity
- **Table F.1 heading-only changes in blue**

~~Table F.1 - Operation Binding - mathematical operations~~

Table F.1 - Bindings for ISO/IEC 60559 Mandated Mathematical Operations

- **Table F.2 (now F.4) heading-only changes in blue**

~~Table F.2 - ISO/IEC 60559 operation to C function~~

Table F.4 - Bindings for ISO/IEC 60559 Recommended Mathematical Operations

F.1#4 Introduction

..... mentioned in the subclauses just listed prior are understood to also refer to is understood to also refer to the rounding direction mode

F.2.2 - Infinities and NaNs - #5 is NEW - so RENUMBER the rest

- 1 As negative infinity ($-\infty$) and positive infinity ($+\infty$) are representable in ISO/IEC 60559 formats, all real numbers lie within the range of representable values (5.3.5.3.3), The **INFINITY** and **DEC_INFINITY** macros in **<float.h>** provide designations for ISO/IEC 60559 infinities.
- 2 This annex requires full support for quiet NaNs as specified by ISO/IEC 60559, but does not require full support for signaling NaNs as specified by ISO/IEC 60559. The behavior of signaling NaNs depends on whether or not an implementation provides full support for signaling NaNs. Where it does, that behavior should adhere to the recommended practice noted in subsequent paragraphs in this clause. Where it does not, that behavior is implementation-defined, i.e., treated as either an ISO/IEC 60559 quiet NaN or an ISO/IEC 60559 signaling NaN.³⁹⁴⁾ The **NAN** and **INFINITY** **DEC_NAN** macros in **<float.h>** and the **nan** family of functions in **<math.h>** provide designations for ISO/IEC 60559 quiet NaNs and infinities. The **FLT_SNAN**, **DBL_SNAN**, and **LDBL_SNAN** **LDBL_SNAN**, **DEC32_SNAN**, **DEC64_SNAN**, and **DEC128_SNAN**, macros in **<float.h>** provide designations for ISO/IEC 60559 signaling NaNs.
- 3 ~~This annex does not require the full support for signaling NaNs specified in ISO/IEC 60559.~~ This annex uses the term NaN, unless explicitly qualified, to denote any of the representable quiet NaNs. ~~Where specification of signaling NaNs is not provided, the behavior of signaling NaNs is implementation-defined (either treated as an ISO/IEC 60559 quiet NaN or treated as an ISO/IEC 60559 signaling NaN).~~³⁹⁴⁾
- 4 Any operator or **<math.h>** function returning a floating-point result that raises an "invalid" floating-point exception ~~if delivering a floating type result~~, shall return a quiet NaN unless ~~explicitly specified:~~ stated otherwise.
- 5 Any operator or **<math.h>** function with a quiet NaN input returning a floating-point result shall return a quiet NaN and raise no floating-point exception unless stated otherwise. *[NEW - comes from F.10.1#14]*
- 6 To support signaling NaNs as specified in by ISO/IEC 60559, an implementation should adhere to the following recommended practice.

Recommended Practice

- 7 Any operator or **<math.h>** function or macro with a signaling NaN input ~~, unless explicitly specified otherwise,~~ raises an "invalid" floating-point exception ~~- unless stated otherwise.~~
- 8 **NOTE** ~~Some functions do not propagate quiet NaN arguments. For example, **hypot**(*x*, *y*) returns infinity if *x* or *y* is infinite and the other is a quiet NaN.~~ Even for those functions that have special treatment for quiet NaN input, such as those that do not propagate that quiet NaN, the preceding paragraph is still recommended practice for signaling NaN input. A case in point is **hypot**(*x*, *y*). If either of *x* or *y* is a quiet NaN and the other is infinite, no floating-point exception is raised and ∞ is returned. But, if instead either of *x* or *y* is a signaling NaN and the other infinite, ~~The~~ the recommended practice ~~in this subclause specifies that such functions (and others) is to raise the "invalid" floating-point exception if an argument is a signaling NaN, which also implies they return a quiet NaN in these cases.~~ is to raise an "invalid" floating-point exception and implicitly, to return a quiet NaN.
- 9 The **<fenv.h>** header defines the macro **FE_SNANS_ALWAYS_SIGNAL** if and only if the implementation follows the recommended practice in this subclause. If defined, **FE_SNANS_ALWAYS_SIGNAL** expands to the integer literal 1.
- 10 If a function with one or more quiet NaN arguments returns a quiet NaN result, the result should be the same as one of those quiet NaN arguments (after possible type conversion), except perhaps for the sign. *[New - comes from F.10.1#17]*

F.3#1 Operations

[See Rationale]

The C operators, functions, ~~and~~ function-like ~~macros~~, macros and constant expressions in Table F.1 provide operations mandated by ISO/IEC 60559 and the C functions of Table F.4 provide operations recommended specified by ISO/IEC 60559. ~~as shown in Table F.1. In the table, C functions are represented by the function name without a type suffix. Specifications for the C facilities are provided in the listed clauses. The C specifications are intended to match ISO/IEC 60559, unless stated otherwise.~~ Subsequent subclauses in this appendix contain C specifications about the rounding behavior, special cases, and floating-point exceptions of these various C facilities. Unless stated otherwise, any such C specification matches ISO/IEC 60559 since all of the aforementioned operations are completely specified by ISO/IEC 60559. Any reference to the principal function of a family of functions - i.e., that function within a family without a type suffix - in any entry from Table F.1, Table F.4, or any of those subclauses, implies or infers all functions within that family.

• Changes to Table F.1 in blue

.....		
convertToIntegerTiesToEven convertToIntegerTowardZero convertToIntegerTowardPositive convertToIntegerTowardNegative	cast of the result of fromfp or ufromfp	7.12.10.10, F.10.7.10
convertToIntegerTiesToAway	cast of the result of fromfp or ufromfp , lround , llround	7.12.10.10, F.10.7.10 7.12.10.7, F.10.7.7
convertToIntegerTiesToEven convertToIntegerExactTowardZero convertToIntegerExactTowardPositive convertToIntegerExactTowardNegative convertToIntegerExactTiesToAway	cast of the result of fromfpx or ufromfpx	7.12.10.11, F.10.7.11
.....		
radix (of a real floating type)	FLT_RADIX	5.3.5.3.3
radix (of a decimal floating type)	DEC_RADIX	5.3.5.3.4
.....		
getBinaryRoundingDirection	fegetround	7.6.6.3
setBinaryRoundingDirection	fesetround	7.6.6.6
getDecimalRoundingDirection	fe_dec_getround	7.6.6.4
setDecimalRoundingDirection	fe_dec_setround	7.6.6.7
.....		

F.3#6 Operations

~~The~~ Each of the C classification macros **fpclassify**, **iscanonical**, **isfinite**, **isinf**, **isnan**, **isnormal**, **issignaling**, **issubnormal**, **iszero**, and **signbit**, provides the ISO/IEC 60559 operation indicated in Table F.1 ~~provided their arguments are only if the macro's argument is~~ in the format of its semantic type. ~~Then these~~ These macros raise no floating-point exception ~~s~~, even if ~~an~~ the argument is a signaling NaN.

F.3#7 Operations

[Qualify the use of NaN to mention both quiet NaNs and signaling NaNs]

..... This applies to ~~all values, including NaNs~~, any argument value, including a quiet NaN or a signaling NaN whose sign bit is

F.3#13 Operations

..... an extra rounding error. The functions that round ~~their~~ result to narrower type

F.3#10 Operations

The For arithmetic with standard floating types, Table F.2 specifies the mapping among C rounding direction modes (5.3.5.3.3), ISO/IEC 60559's rounding direction attributes, and the macros (7.6) **FE_DOWNWARD**, **FE_TONEAREST**, **FE_TONEARESTFROMZERO**, **FE_TOWARDZERO**, and **FE_UPWARD**, which are used in conjunction with the **fegetround** and **fesetround** functions and the **FENV_ROUND** pragma. ,represent the ISO/IEC 60559 rounding-direction-attributes roundTowardNegative, roundTiesToEven, roundTiesToAway, roundTowardZero, and roundTowardPositive, respectively, for binary floating-point arithmetic.

Table F.2: ISO/IEC 60559 rounding for arithmetic with standard floating type

Rounding Direction (Mode)	Rounding Direction Attribute	Rounding Direction Macro
toward positive infinity	roundTowardPositive	FE_UPWARD
toward negative infinity	roundTowardNegative	FE_DOWNWARD
toward zero	roundTowardZero	FE_TOWARDZERO
to nearest with ties to even	roundTiesToEven	FE_TONEAREST
to nearest with ties away from zero	roundTiesToAway	FE_TONEARESTFROMZERO

Support for the ISO/IEC 60559 roundTiesToAway rounding direction attribute for binary floating-point arithmetic, and hence for the **FE_TONEARESTFROMZERO** macro, is optional.

F.3#17 Operations

The integer literal 10 provides the radix operation defined in ISO/IEC 60559 for decimal floating-point arithmetic.

••• **TODO** - Note how **printf** and related functions handle a quiet NaN or signaling NaN argument.

F.3.#18 Operations

[Move routines to Table F.1 and rounding direction mode discussion to F.3#17]

For arithmetic with decimal floating types, Table F.3 specifies the mapping among C rounding direction modes (5.3.5.3.3), ISO/IEC 60559 rounding direction attributes, and the macros (7.6) which are used in conjunction with the **fe_dec_getround** and **fe_dec_setround** functions and the **FENV_DEC_ROUND** pragma.

Table F.3: ISO/IEC 60559 rounding for arithmetic with decimal floating types

Rounding Direction (Mode)	Rounding Direction Attribute	Rounding Direction Macro
toward positive infinity	roundTowardPositive	FE_DEC_UPWARD
toward negative infinity	roundTowardNegative	FE_DEC_DOWNWARD
toward zero	roundTowardZero	FE_DEC_TOWARDZERO
to nearest with ties to even	roundTiesToEven	FE_DEC_TONEAREST
to nearest with ties away from zero	roundTiesToAway	FE_DEC_TONEARESTFROMZERO

The **fe_dec_getround** (7.6.6.4) and **fe_dec_setround** (7.6.6.7) functions provide the **getDecimalRoundingDirection** and **setDecimalRoundingDirection** operations defined in ISO/IEC 60559 for decimal floating-point arithmetic. The macros (7.6) **FE_DEC_DOWNWARD**, **FE_DEC_TONEAREST**, **FE_DEC_TONEARESTFROMZERO**, **FE_DEC_TOWARDZERO**, and **FE_DEC_UPWARD** which are used in conjunction with the **fe_dec_getround** and **fe_dec_setround** functions and the **FENV_DEC_ROUND** pragma, represent the ISO/IEC 60559 rounding-direction-attributes roundTowardNegative, roundTiesToEven, roundTiesToAway, roundTowardZero, and roundTowardPositive, respectively, for decimal floating-point arithmetic.

F.3#20 Operations

The C functions in Table F.2 correspond to mathematical operations recommended by ISO/IEC 60559. However, correct rounding, which ISO/IEC 60559 specifies for its operations, is not required for the C functions in the table. The C functions in Table F.4 are not required to be correctly rounded. 7.35.9 (potentially) reserves **cr_**—prefixed names for functions fully matching the ISO/IEC 60559 mathematical operations. In the table, the C functions are represented by the function name without a type suffix.

F.3#21 Operations - NEW

Recommended Practice

As ISO/IEC 60559 preserves useful mathematical properties such as symmetry, monotonicity, and periodicity for any function **f** in Table F.4 by specifying its correct rounding, a C implementation of that same function which is not correctly rounded should still preserve as many of these properties as possible.

F.3#22 Operations - NEW

For any function **f** in Table F.4, 7.35.9 (potentially) reserves a **cr_**-prefixed name variant: **cr_f**. If a C implementation provides this function **cr_f**, it should deliver a guaranteed correctly rounded result for the ISO/IEC 60559 operation corresponding to **f**.

F.5#5 - Conversions ...

Functions such as **strtod** that convert character sequences to floating types honor the applicable rounding direction **mode**. Hence, ~~if the rounding direction can be upward or downward:~~ the implementation cannot convert a minus-signed sequence by arithmetically negating the converted unsigned sequence ~~should that applicable rounding direction mode be either of "toward positive infinity" or "toward negative infinity".~~

F.10.1#1

...for ISO/IEC 60559 implementations ~~and enhance the definitions in <math.h>.~~

F.10.1#3 General

For each single-argument function **f** in **<math.h>** whose mathematical counterpart is symmetric (even), **f(x)** is **f(-x)** for all ~~rounding modes and~~ **x** in the (valid) domain of the ~~function.~~ function regardless of the applicable rounding direction mode. For each single-argument function **f** in **<math.h>** whose mathematical counterpart is antisymmetric (odd), **f(x)** is **-f(x)** for ~~the ISO/IEC 60559 rounding modes roundTiesToEven, roundTiesToAway, and roundTowardZero, and~~ all **x** in the (valid) domain of the ~~function.~~ function only when the applicable rounding direction mode is one of "to nearest with ties to even", "to nearest with ties away from zero" or "toward zero". Similarly for each dual-argument function **f** in **<math.h>** whose mathematical counterpart is antisymmetric (odd) in its first argument, **f(x,y)** is **-f(-x, y)** for all **x** and **y** in the (valid) domain of the function only when the applicable rounding direction mode is one of "to nearest with ties to even", "to nearest with ties away from zero" or "toward zero". ~~The atan2 and atan2pi functions are odd in their first argument.~~

F.10.1#4 General

~~Special cases for functions in <math.h> are covered directly or indirectly by ISO/IEC 60559. The functions that ISO/IEC 60559 specifies directly are identified in F.3. The other <math.h> functions treat infinities, NaNs, signed zeros, subnormals, and (provided the state of the FENV_ACCESS pragma is "on") the floating-point status flags in a manner consistent with ISO/IEC 60559. A special case for a <math.h> function in Table F.1 or Table F.4 is specified directly by ISO/IEC 60559. A special case for any other <math.h> function is covered indirectly by ISO/IEC 60559, i.e., it treats in a manner consistent with ISO/IEC 60559: infinities, NaNs, signed zeros, subnormals, and, provided the state of the FENV_ACCESS pragma is "on", the floating-point status flags.~~

F.10.1#5 General

~~The expression **math_errhandling & MATH_ERREXCEPT** shall evaluate to a nonzero value.~~ A special case of a **<math.h>** function applies to a particular sub-domain of a parameter of that function. In subsequent sub-clauses, any such sub-domain with an inclusive infinite upper or lower bound is rewritten with that infinite bound implied, i.e., given a floating-point parameter **x** and a constant **C**:

- $x > C$ is equivalent to $C < x \leq +\infty$,
- $x \geq C$ is equivalent to $C \leq x \leq +\infty$,
- $x < C$ is equivalent to $-\infty \leq x < C$,
- $x \leq C$ is equivalent to $-\infty \leq x \leq C$.

F.10.1#6 General

[What is here already refers to F.3#1 so we really should put it there]

[Recover the paragraph that we deleted in the previous paragraph]

~~The functions bound to operations in ISO/IEC 60559 (F.3) are fully specified by ISO/IEC 60559, including rounding behaviors and floating-point exceptions.~~

The expression `math_errhandling & MATH_ERREXCEPT` shall evaluate to a nonzero value.

F.10.1#10 General

Whether or when ~~library~~ `<math.h>` functions not listed in Table F.1 F.4 raise the

F.10.1#11 General

Whether or when ~~library~~ `<math.h>` functions not listed in Table F.1 F.4 raise a

F.10.1#12 General

As implied by F.8.7, ~~library~~ functions in the standard libraries, including `<math.h>`, do not raise

F.10.1#13 General

Whether ~~or when the~~ `<math.h>` functions not listed in Table F.1 F.4 honor the applicable rounding direction mode is implementation-defined, unless ~~explicitly specified stated~~ otherwise.

F.10.1#14 General

[1st sentence -> F.2.2#5 - 2nd sentence comes from F.10.1#15]

~~Functions with a NaN argument return a NaN result and raise no floating-point exception, except where explicitly stated otherwise. In subsequent subclauses, no mention is made of floating-point exceptions when none are raised unless stated otherwise.~~

F.10.1#15 General

[1st sentence -> F.10.1#1]

[The 2nd sentence is already mandated by updated F.3#1]

~~The specifications in the following subclauses append to the definitions in `<math.h>`. For families of functions, the specifications apply to all the functions even though only the principal function is shown.~~

[This paragraph now deals solely with issues with a $\pm$ symbol in a special case]

Unless ~~specified stated~~ otherwise, where the $\pm$ symbol occurs in both an argument and the result of subsequent clauses, ~~the result has the same sign as the argument~~ it implies two cases: one with each $\pm$ symbol replaced with a plus (+) symbol in both argument and result, and the other with each $\pm$ symbol replaced with a minus (−) symbol in both argument and result.

F.10.1#16 - General - Recommended Practice

[1st sentence -> F.3#1, the rest merged with F.3#20]

~~ISO/IEC 60559 specifies correct rounding for the operations in Table F.4 recommended by ISO/IEC 60559 and thereby preserves useful mathematical properties such as symmetry, monotonicity, and periodicity. The corresponding functions with (potentially) reserved `er_`-prefixed names (7.35.9) do the same. The C functions in the table, however, are not required to be correctly rounded although implementations should still preserve as many of these useful mathematical properties as possible.~~

F.10.1#17 - General - Recommended Practice

[Moved to F.2.2#10]

~~If a function with one or more NaN arguments returns a NaN result, the result should be the same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.~~