ISO/IEC 60559 Floating-Point Support Annex for C++

Number: P4212R0
Date: 2026-05-11
Audience: SG6
Reply to: Guy Davidson <gd@6it.dev>

Revision History

- R0: Initial proposal with design and wording sketch

Abstract

This paper proposes a new normative Annex specifying support for floating-point arithmetic as defined by [559]. The C standard provides a similar annex, but it is not directly suitable for C++ due to differences in language semantics, constant evaluation, templates, and the standard library. This proposal integrates [559] semantics into C++ in a manner consistent with existing language rules and implementation practice.

Motivation

C++ implementations largely target hardware which supports [559], yet the Standard does not require consistent semantics for rounding, exceptions, NaN propagation, or reproducibility. This gap leads to portability issues and limits the reliability of numerical software.

Existing facilities such as `<cfenv>` are underspecified and difficult to use correctly in modern C++ contexts, especially with optimization and concurrency.

Design Overview

This proposal introduces a new Annex specifying ISO/IEC 60559 conformance. There are nine clauses that need to be considered from ISO/IEC 60559:2020, clauses 3 to 11. [559] offers binary and decimal floating-point arithmetic features. This proposal will only introduce binary floating-point feature support.

[559] specifies attributes as something logically associated with a program block to modify its numerical and exception semantics. A user can specify a value for an attribute parameter statically or dynamically. For example, value changing optimisation attributes can be

specified on the command line, while rounding attributes can be specified in a function call at runtime.

Attribute is sadly an overloaded term, since [559] defines attributes in a different way. All attempts shall be made to avoid ambiguity.

This overview iterates through the relevant clauses of [559] describing what support is offered or needed by [C++] to implement the features.

Floating point formats [559] clause 3

[559] specifies three binary formats, with encodings of length 32, 64 and 128 bits. [C++] specifies extended floating point types in [basic.extended.fp], three of which correspond with these types. They are supported via the feature macros __STDCPP_FLOAT32_T__, __STDCPP_FLOAT64_T__ and __STDCPP_FLOAT128_T__. This section also includes implementation-defined extended and possibly wider types beyond the standard interchange formats.

{Design question - should these three types be the [559] types or should we specify that float, double and long double are these types?}

Conversion from floating-point values to integer values is covered in [expr.arith.conv].

Conversion between floating-point types and decimal character sequences is quite underspecified. [lex.fcon] §3 states “If the scaled value is not in the range of representable values for its type, the program is ill-formed. Otherwise, the value of a floating-point-literal is the scaled value if representable, else the larger or smaller representable value nearest the scaled value, chosen in an implementation-defined manner.”

Rounding [559] clause 4

This clause also defines the nature of attributes, which are not limited to rounding. There are two rounding considerations: rounding to nearest and directed rounding. Five rounding modes are specified. In [C++] at [round.style] four of these modes are specified as part of the enumeration float_round_style. They map to:

IEC 60559 identifier	C++ identifier
roundTiesToEven	round_to_nearest
roundTowardsPositive	round_toward_infinity
roundTowardsNegative	round_toward_neg_infinity
roundTowardsZero	round_toward_zero

Operations [559] clause 5

The C++ standard library also makes available the facilities of the C standard library, suitably adjusted to ensure static type safety. Support for the operations specified in clause 5 of [559] is mostly achieved by reference to the C standard library. Some functions are superseded by C++ standard library functions. Some operations are supported by infix operator notation.

The wording needs to reflect the list of operations specified in this clause and the corresponding library functions, with additional paragraphs detailing any clarifications, for example [559] requires that some of its operations be provided for mixed operand formats, which is covered by the usual arithmetic conversions and the function-call argument conversions.

Consideration needs to be given to the return statement, and how conversion should happen when returning to a different type.

Infinity, NaNs and sign bit [559] clause 6

The C standard library provides macros and functions for NaN, signaling NaN and infinity. Consideration needs to be given to how quiet and signaling NaNs are differentiated and denoted.

By default, quiet NaNs are delivered when an invalid operation is executed. Sign bits of NaNs come into play under some operations, for example negation. Specification is required for the sign of zero.

Exceptions and default exception handling [559] clause 7

When an operation raises a floating-point exception, default exception handling is assumed: the flag is set, a default result is delivered and execution continues.

Alternate exception handling attributes [559] clause 8

Implementations might provide alternate exception handling as an extension, but this annex should not specify any.

Recommended operations [559] clause 9

This clause specifies additional operations, such as mathematical functions, floating-point environment controls, reduction operations such as dot product, sum of squares and so on, min and max functions, and payload operations. Many of these are already defined in the C++ standard.

Expression evaluation [559] clause 10

When combining operations, the order of execution is significant. Intermediate types need to be considered, as well as possible conversion to the final result format. The same problem is addressed regarding parameters and function values.

There is also discussion of value changing optimisations. This greatly stretches the “as-if” rule since implementations often offer compiler flags which relax restrictions. This feeds into the next clause.

Reproducibility [559] clause 11

While steps to reproducibility is specified in this clause, it is not specified in the C++ standard. Efforts have been made in P3375. This proposal seeks to specify what we have before improving conformance in this way.

Floating-point environment

The floating-point environment consists of the floating-point exception status flags and the rounding-direction control modes. Implementations may provide additional information as an extension.

During translation, the floating-point environment is not specified and is implementation-defined. C has an FENV_ACCESS pragma but [cfenv.syn] explicitly remarks that it is not required. Constant expressions are evaluated as if at execution time.

At startup, floating-point exception status flags are cleared, the dynamic rounding direction mode is rounding to nearest and the dynamic rounding precision mode is set so that results are not shortened. Trapping is disabled on all floating-point exceptions.

Automatic initialisation is done at execution time, or as if at execution time, while static and thread-local object initialisation is done at translation, or as if at translation time.

The environment can be changed at execution time using the functions declared in the <cfenv> header.

Draft wording

(This is a sketch of how the wording might be arranged, rather than final normative wording)

Annex F (normative)
IEC 60559 floating-point arithmetic [iec60559]

F.1 General [iec60559.general]

1. Introduce the annex
2. Discuss binary and decimal floating-point
3. Introduce feature test macros

4. Note that bindings imply 60559 behaviour is adopted by reference
5. Observations about attributes

F.2 Types [iec60559.types]
1. Discuss float, double, long double, float16_t, float32_t, float64_t, float128_t

F.3 Operations [iec60559.operations]
1. Describe bindings between IEC 60559 operations and C++ operations.
2. Identify nooks and crannies in the bindings.
3. There are many of these, and subsequent paragraphs should detail them individually.

F.4 Infinity and NaN [iec60559.infnan]
1. Remark on range of real numbers being contained within the range of representable values.
2. Describe the entities which denote infinity and NaN.
3. (research signaling NaNs)
4. Floating-point exception => quiet Nan.
5. Supporting signaling NaNs

F.5 Maths [iec60559.math]
1. Describe the math functions
2. Iterate through the trig functions
3. Iterate through the hyperbolic functions
4. Iterate through the log/exp functions
5. Iterate through the power functions
6. Iterate through the error functions
7. Iterate through the nearest integer functions
8. Iterate through the remainder functions
9. Iterate through the manipulation functions
10. Iterate through the min, max and positive difference functions
11. fma
12. Functions that round result to narrower type
13. Iterate through the total order functions
14. Iterate through the payload functions

F.5 Conversion [iec60559.conv]
1. Reference [expr.arith.conv]
2. Consider binary floating types <-> decimal character sequences
3. Returning values to a different floating-point format

F.6 Contracted expressions [iec60559.fma]
1. Describe the workings of FMA.

F.7 Floating point environment [iec60559.env]
1. Introduce the nature of the environment
2. Static environment
3. Dynamic environment
4. Constant expressions

5. Initialisation
6. Modifying the environment

F.8 Optimisation [iec60559.opt]

1. Global transformations
2. Expression transformations
3. Relational operators
4. Constant arithmetic

F.9 Expression evaluation and reproducibility

Bibliography:

[559] ISO/IEC JTC 1/SC 25 (May 2020). ISO/IEC 60559:2020 — Information technology — Microprocessor Systems — Floating-Point arithmetic. ISO. pp. 1–74.
[C++] C++23 Working Draft
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/n4950.pdf