TECHNICAL SPECIFICATION

ISO/IEC TS 18661-3

First edition
2015-10-01

Information Technology — Programming languages, their environments, and system software interfaces — Floating-point extensions for C —

Part 3: Interchange and extended types

Technologies de l'information — Langages de programmation, leurs environnements et interfaces du logiciel système — Extensions à virgule flottante pour C —

Partie 3: Types d'échange et étendus

Reference number
ISO/IEC TS 18661-3:2015(E)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments, and system software interfaces*.

ISO/IEC TS 18661 consists of the following parts, under the general title *Information technology — Programming languages, their environments, and system software interfaces — Floating-point extensions for C*:

— *Part 1: Binary floating-point arithmetic*

— *Part 2: Decimal floating-point arithmetic*

— *Part 3: Interchange and extended types*

— *Part 4: Supplementary functions*

The following part is under preparation:

— *Part 5: Supplementary attributes*

ISO/IEC TS 18661-1 updates ISO/IEC 9899:2011, *Information technology — Programming Language C*, annex F in particular, to support all required features of ISO/IEC/IEEE 60559:2011, *Information technology — Microprocessor Systems — Floating-point arithmetic*.

ISO/IEC TS 18661-2 supersedes ISO/IEC TR 24732:2009, *Information technology — Programming languages, their environments and system software interfaces — Extension for the programming language C to support decimal floating-point arithmetic*.

ISO/IEC TS 18661-3, ISO/IEC TS 18661-4, and ISO/IEC TS 18661-5 specify extensions to ISO/IEC 9899:2011 for features recommended in ISO/IEC/IEEE 60559:2011.

Introduction

Background

IEC 60559 floating-point standard

The IEEE 754-1985 standard for binary floating-point arithmetic was motivated by an expanding diversity in floating-point data representation and arithmetic, which made writing robust programs, debugging, and moving programs between systems exceedingly difficult. Now the great majority of systems provide data formats and arithmetic operations according to this standard. The IEC 60559:1989 international standard was equivalent to the IEEE 754-1985 standard. Its stated goals were the following:

1 Facilitate movement of existing programs from diverse computers to those that adhere to this standard.

2 Enhance the capabilities and safety available to programmers who, though not expert in numerical methods, may well be attempting to produce numerically sophisticated programs. However, we recognize that utility and safety are sometimes antagonists.

3 Encourage experts to develop and distribute robust and efficient numerical programs that are portable, by way of minor editing and recompilation, onto any computer that conforms to this standard and possesses adequate capacity. When restricted to a declared subset of the standard, these programs should produce identical results on all conforming systems.

4 Provide direct support for

   a. Execution-time diagnosis of anomalies

   b. Smoother handling of exceptions

   c. Interval arithmetic at a reasonable cost

5 Provide for development of

   a. Standard elementary functions such as exp and cos

   b. Very high precision (multiword) arithmetic

   c. Coupling of numerical and symbolic algebraic computation

6 Enable rather than preclude further refinements and extensions.

To these ends, the standard specified a floating-point model comprising the following:

— *formats* – for binary floating-point data, including representations for Not-a-Number (NaN) and signed infinities and zeros

— *operations* – basic arithmetic operations (addition, multiplication, etc.) on the format data to compose a well-defined, closed arithmetic system; also specified conversions between floating-point formats and decimal character sequences, and a few auxiliary operations

— *context* – status flags for detecting exceptional conditions (invalid operation, division by zero, overflow, underflow, and inexact) and controls for choosing different rounding methods

The ISO/IEC/IEEE 60559:2011 international standard is equivalent to the IEEE 754-2008 standard for floating-point arithmetic, which is a major revision to IEEE 754-1985.

The revised standard specifies more formats, including decimal as well as binary. It adds a 128-bit binary format to its basic formats. It defines extended formats for all of its basic formats. It specifies data interchange formats (which may or may not be arithmetic), including a 16-bit binary format and an unbounded tower of wider formats. To conform to the floating-point standard, an implementation must provide at least one of the basic formats, along with the required operations.

The revised standard specifies more operations. New requirements include – among others – arithmetic operations that round their result to a narrower format than the operands (with just one rounding), more conversions with integer types, more classifications and comparisons, and more operations for managing flags and modes. New recommendations include an extensive set of mathematical functions and seven reduction functions for sums and scaled products.

The revised standard places more emphasis on reproducible results, which is reflected in its standardization of more operations. For the most part, behaviors are completely specified. The standard requires conversions between floating-point formats and decimal character sequences to be correctly rounded for at least three more decimal digits than is required to distinguish all numbers in the widest supported binary format; it fully specifies conversions involving any number of decimal digits. It recommends that transcendental functions be correctly rounded.

The revised standard requires a way to specify a constant rounding direction for a static portion of code, with details left to programming language standards. This feature potentially allows rounding control without incurring the overhead of runtime access to a global (or thread) rounding mode.

Other features recommended by the revised standard include alternate methods for exception handling, controls for expression evaluation (allowing or disallowing various optimizations), support for fully reproducible results, and support for program debugging.

The revised standard, like its predecessor, defines its model of floating-point arithmetic in the abstract. It neither defines the way in which operations are expressed (which might vary depending on the computer language or other interface being used), nor does it define the concrete representation (specific layout in storage, or in a processor's register, for example) of data or context, except that it does define specific encodings that are to be used for the exchange of floating-point data between different implementations that conform to the specification.

IEC 60559 does not include bindings of its floating-point model for particular programming languages. However, the revised standard does include guidance for programming language standards, in recognition of the fact that features of the floating-point standard, even if well supported in the hardware, are not available to users unless the programming language provides a commensurate level of support. The implementation's combination of both hardware and software determines conformance to the floating-point standard.

C support for IEC 60559

The C standard specifies floating-point arithmetic using an abstract model. The representation of a floating-point number is specified in an abstract form where the constituent components (sign, exponent, significand) of the representation are defined but not the internals of these components. In particular, the exponent range, significand size, and the base (or radix) are implementation-defined. This allows flexibility for an implementation to take advantage of its underlying hardware architecture. Furthermore, certain behaviors of operations are also implementation-defined, for example in the area of handling of special numbers and in exceptions.

The reason for this approach is historical. At the time when C was first standardized, before the floating-point standard was established, there were various hardware implementations of floating-point arithmetic in common use. Specifying the exact details of a representation would have made most of the existing implementations at the time not conforming.

Beginning with ISO/IEC 9899:1999 (C99), C has included an optional second level of specification for implementations supporting the floating-point standard. C99, in conditionally normative annex F, introduced nearly complete support for the IEC 60559:1989 standard for binary floating-point arithmetic. Also, C99's informative annex G offered a specification of complex arithmetic that is compatible with IEC 60559:1989.

ISO/IEC 9899:2011 (C11) includes refinements to the C99 floating-point specification, though it is still based on IEC 60559:1989. C11 upgraded annex G from "informative" to "conditionally normative".

ISO/IEC TR 24732:2009 introduced partial C support for the decimal floating-point arithmetic in ISO/IEC/IEEE 60559:2011. ISO/IEC TR 24732, for which technical content was completed while IEEE 754-2008 was still in the later stages of development, specifies decimal types based on ISO/IEC/IEEE 60559:2011 decimal formats, though it does not include all of the operations required by ISO/IEC/IEEE 60559:2011.

Purpose

The purpose of ISO/IEC TS 18661 is to provide a C language binding for ISO/IEC/IEEE 60559:2011, based on the C11 standard, that delivers the goals of ISO/IEC/IEEE 60559 to users and is feasible to implement. It is organized into five parts.

ISO/IEC TS 18661-1 provides changes to C11 that cover all the requirements, plus some basic recommendations, of ISO/IEC/IEEE 60559:2011 for binary floating-point arithmetic. C implementations intending to support ISO/IEC/IEEE 60559:2011 are expected to conform to conditionally normative annex F as enhanced by the changes in ISO/IEC TS 18661-1.

ISO/IEC TS 18661-2 enhances ISO/IEC TR 24732 to cover all the requirements, plus some basic recommendations, of ISO/IEC/IEEE 60559:2011 for decimal floating-point arithmetic. C implementations intending to provide an extension for decimal floating-point arithmetic supporting ISO/IEC/IEEE 60559:2011 are expected to conform to ISO/IEC TS 18661-2.

ISO/IEC TS 18661-3 (Interchange and extended types), ISO/IEC TS 18661-4 (Supplementary functions), and ISO/IEC TS 18661-5 (Supplementary attributes) cover recommended features of ISO/IEC/IEEE 60559:2011. C implementations intending to provide extensions for these features are expected to conform to the corresponding parts.

Additional background on formats

The revised floating-point arithmetic standard, ISO/IEC/IEEE 60559:2011, introduces a variety of new formats, both fixed and extendable. The new fixed formats include

— a 128-bit basic binary format (the 32 and 64 bit basic binary formats are carried over from ISO/IEC 60559:1989)

— 64 and 128 bit basic decimal formats

— interchange formats, whose precision and range are determined by the width *k*, where

for binary, *k* = 16, 32, 64, and *k* ≥ 128 and a multiple of 32, and

for decimal, *k* ≥ 32 and a multiple of 32

— extended formats, for each basic format, with minimum range and precision specified

Thus IEC 60559 defines five basic formats — binary32, binary64, binary128, decimal64, and decimal128 — and five corresponding extended formats, each with somewhat more precision and range than the basic format it extends. IEC 60559 defines an unlimited number of interchange formats, which include the basic formats.

Interchange formats may or may not be supported as arithmetic formats. If not, they may be used for the interchange of floating-point data but not for arithmetic computation. IEC 60559 provides conversions between non-arithmetic interchange formats and arithmetic formats which can be used for computation.

Extended formats are intended for intermediate computation, not input or output data. The extra precision often allows the computation of extended results which when converted to a narrower output format differ from the ideal results by little more than a unit in the last place. Also, the extra range often avoids any intermediate overflow or underflow that might occur if the computation were done in the format of the data. The essential property of extended formats is their sufficient extra widths, not their specific widths. Extended formats for any given basic format may vary among implementations.

Extendable formats, which provide user control over range and precision, are not covered in ISO/IEC TS 18661.

The 32 and 64 bit binary formats are supported in C by types **`float`** and **`double`**. If a C implementation defines the macro **`__STDC_IEC_60559_BFP__`** (see ISO/IEC TS 18661-1) signifying that it supports C Annex F for binary floating-point arithmetic, then its **`float`** and **`double`** formats must be IEC 60559 binary32 and binary64.

ISO/IEC TS 18661-2 defines types **`_Decimal32`**, **`_Decimal64`**, and **`_Decimal128`** with IEC 60559 formats decimal32, decimal64, and decimal128. Although IEC 60559 does not require arithmetic support (other than conversions) for its decimal32 interchange format, ISO/IEC TS 18661-2 has full arithmetic and library support for **`_Decimal32`**, just like for **`_Decimal64`** and **`_Decimal128`**.

The C Standard provides just three standard floating types (**`float`**, **`double`**, and **`long double`**) that are required of all implementations. C Annex F for binary floating-point arithmetic requires the standard floating types to be binary. The **`long double`** type must be at least as wide as **`double`**, but C does not further specify details of its format, even in Annex F.

ISO/IEC TS 18661-3, this document, provides nomenclatures for types with IEC 60559 arithmetic interchange formats and extended formats. The nomenclatures allow portable use of the formats as envisioned in IEC 60559. This document covers these aspects of the types:

— names

— characteristics

— conversions

— constants

— function suffixes

— character sequence conversion interfaces

This specification includes interchange and extended nomenclatures for formats that, in some cases, already have C nomenclatures. For example, types with the IEC 60559 double format may include **`double`**, **`_Float64`** (the type for the binary64 interchange format), and maybe **`_Float32x`** (the type for the binary32-extended format). This redundancy is intended to support the different programming models appropriate for the types with arithmetic interchange formats and extended formats and C standard floating types.

This document also supports the IEC 60559 non-arithmetic interchange formats with functions that convert among encodings and between encodings and character sequences, for all interchange formats.

Information technology — Programming languages, their environments, and system software interfaces — Floating-point extensions for C —

Part 3: **Interchange and extended types**

1 Scope

This part of ISO/IEC TS 18661 extends programming language C to include types with the arithmetic interchange and extended floating-point formats specified in ISO/IEC/IEEE 60559:2011, and to include functions that support the non-arithmetic interchange formats in that standard.

2 Conformance

An implementation conforms to this part of ISO/IEC TS 18661 if

a) it meets the requirements for a conforming implementation of C11 with all the changes to C11 specified in parts 1-3 of ISO/IEC TS 18661;

b) it conforms to ISO/IEC TS 18661-1 or ISO/IEC TS 18661-2 (or both); and

c) it defines **`__STDC_IEC_60559_TYPES__`** to **`201506L`**.

3 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9899:2011, *Information technology — Programming languages — C*

ISO/IEC/IEEE 60559:2011, *Information technology — Microprocessor Systems — Floating-point arithmetic*

ISO/IEC TS 18661-1:2014, *Information technology — Programming languages, their environments and system software interfaces — Floating-point extensions for C — Part 1: Binary floating-point arithmetic*

ISO/IEC TS 18661-2:2015, *Information technology — Programming languages, their environments and system software interfaces — Floating-point extensions for C — Part 2: Decimal floating-point arithmetic*

4 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 9899:2011, ISO/IEC/IEEE 60559:2011, ISO/IEC TS 18661-1:2014, ISO/IEC TS 18661-2:2015, and the following apply.

4.1

C11
standard ISO/IEC 9899:2011, *Information technology — Programming languages C*, including *Technical Corrigendum 1* (ISO/IEC 9899:2011/Cor. 1:2012)

5 C standard conformance

5.1 Freestanding implementations

The specification in C11 + TS18661-1 + TS18661-2 allows freestanding implementations to conform to this part of ISO/IEC TS 18661.

5.2 Predefined macros

Change to C11 + TS18661-1 + TS18661-2:

In 6.10.8.3#1, add:

> **__STDC_IEC_60559_TYPES__** The integer constant **201506L**, intended to indicate support of interchange and extended floating types according to IEC 60559.

5.3 Standard headers

The new identifiers added to C11 library headers by this part of ISO/IEC TS 18661 are defined or declared by their respective headers only if **__STDC_WANT_IEC_60559_TYPES_EXT__** is defined as a macro at the point in the source file where the appropriate header is first included. The following changes to C11 + TS18661-1 + TS18661-2 list these identifiers in each applicable library subclause.

Changes to C11 + TS18661-1 + TS18661-2:

After 5.2.4.2.2#6b, insert the paragraph:

> [6c] The following identifiers are defined only if **__STDC_WANT_IEC_60559_TYPES_EXT__** is defined as a macro at the point in the source file where **<float.h>** is first included:
>
> for supported types **_Float*N***:

FLT*N*_MANT_DIG	**FLT*N*_MIN_10_EXP**	**FLT*N*_EPSILON**
FLT*N*_DECIMAL_DIG	**FLT*N*_MAX_EXP**	**FLT*N*_MIN**
FLT*N*_DIG	**FLT*N*_MAX_10_EXP**	**FLT*N*_TRUE_MIN**
FLT*N*_MIN_EXP	**FLT*N*_MAX**	

> for supported types **_Float*N*x**:

FLT*N*X_MANT_DIG	**FLT*N*X_MIN_10_EXP**	**FLT*N*X_EPSILON**
FLT*N*X_DECIMAL_DIG	**FLT*N*X_MAX_EXP**	**FLT*N*X_MIN**
FLT*N*X_DIG	**FLT*N*X_MAX_10_EXP**	**FLT*N*X_TRUE_MIN**
FLT*N*X_MIN_EXP	**FLT*N*X_MAX**	

for supported types **_Decimal***N*, where *N* ≠ 32, 64, and 128:

DEC*N*_MANT_DIG	**DEC*N*_MAX**	**DEC*N*_TRUE_MIN**
DEC*N*_MIN_EXP	**DEC*N*_EPSILON**	
DEC*N*_MAX_EXP	**DEC*N*_MIN**	

for supported types **_Decimal*N*x**:

DEC*N*X_MANT_DIG	**DEC*N*X_MAX**	**DEC*N*X_TRUE_MIN**
DEC*N*X_MIN_EXP	**DEC*N*X_EPSILON**	
DEC*N*X_MAX_EXP	**DEC*N*X_MIN**	

After 7.3#2, insert the paragraph:

[2a] The following identifiers are declared or defined only if **__STDC_WANT_IEC_60559_TYPES_EXT__** is defined as a macro at the point in the source file where **<complex.h>** is first included:

for supported types **_Float***N*:

cacosf*N*	**catanhf***N*	**csqrtf***N*
casinf*N*	**ccoshf***N*	**cargf***N*
catanf*N*	**csinhf***N*	**cimagf***N*
ccosf*N*	**ctanhf***N*	**CMPLXF***N*
csinf*N*	**cexpf***N*	**conjf***N*
ctanf*N*	**clogf***N*	**cprojf***N*
cacoshf*N*	**cabsf***N*	**crealf***N*
casinhf*N*	**cpowf***N*	

for supported types **_Float*N*x**:

cacosf*N*x	**catanhf*N*x**	**csqrtf*N*x**
casinf*N*x	**ccoshf*N*x**	**cargf*N*x**
catanf*N*x	**csinhf*N*x**	**cimagf*N*x**
ccosf*N*x	**ctanhf*N*x**	**CMPLXF*N*X**
csinf*N*x	**cexpf*N*x**	**conjf*N*x**
ctanf*N*x	**clogf*N*x**	**cprojf*N*x**
cacoshf*N*x	**cabsf*N*x**	**crealf*N*x**
casinhf*N*x	**cpowf*N*x**	

After 7.12#1c, insert the paragraph:

[1d] The following identifiers are defined or declared only if **__STDC_WANT_IEC_60559_TYPES_EXT__** is defined as a macro at the point in the source file where **<math.h>** is first included:

long_double_t

for supported types **_Float***N*:

_Float*N*_t	**log1pf***N*	**fromfpf***N*
HUGE_VAL_F*N*	**log2f***N*	**ufromfpf***N*

SNANF*N*
FP_FAST_FMAF*N*
acosf*N*
asinf*N*
atanf*N*
atan2f*N*
cosf*N*
sinf*N*
tanf*N*
acoshf*N*
asinhf*N*
atanhf*N*
coshf*N*
sinhf*N*
tanhf*N*
expf*N*
exp2f*N*
expm1f*N*
frexpf*N*
ilogbf*N*
ldexpf*N*
llogbf*N*
logf*N*
log10f*N*
logbf*N*
modff*N*
scalbnf*N*
scalblnf*N*
cbrtf*N*
fabsf*N*
hypotf*N*
powf*N*
sqrtf*N*
erff*N*
erfcf*N*
lgammaf*N*
tgammaf*N*
ceilf*N*
floorf*N*
nearbyintf*N*
rintf*N*
lrintf*N*
llrintf*N*
roundf*N*
lroundf*N*
llroundf*N*
truncf*N*
roundevenf*N*
fromfpxf*N*
ufromfpxf*N*
fmodf*N*
remainderf*N*
remquof*N*
copysignf*N*
nanf*N*
nextafterf*N*
nextupf*N*
nextdownf*N*
canonicalizef*N*
encodef*N*
decodef*N*
fdimf*N*
fmaxf*N*
fminf*N*
fmaxmagf*N*
fminmagf*N*
fmaf*N*
totalorderf*N*
totalordermagf*N*
getpayloadf*N*
setpayloadf*N*
setpayloadsigf*N*

for supported types _Float*N*x:

HUGE_VAL_F*N*X
SNANF*N*X
FP_FAST_FMAF*N*X
acosf*N*x
asinf*N*x
atanf*N*x
atan2f*N*x
cosf*N*x
sinf*N*x
tanf*N*x
acoshf*N*x
asinhf*N*x
atanhf*N*x
expf*N*x
exp2f*N*x
expm1f*N*x
frexpf*N*x
ilogbf*N*x
llogbf*N*x
ldexpf*N*x
logf*N*x
logbf*N*x
modff*N*x
scalbnf*N*x
scalblnf*N*x
cbrtf*N*x
fabsf*N*x
hypotf*N*x
powf*N*x
sqrtf*N*x
erff*N*x
erfcf*N*x
lgammaf*N*x
tgammaf*N*x
ceilf*N*x
floorf*N*x
nearbyintf*N*x
rintf*N*x
lrintf*N*x
llrintf*N*x
roundf*N*x
lroundf*N*x
fromfpf*N*x
ufromfpf*N*x
fromfpxf*N*x
ufromfpxf*N*x
fmodf*N*x
remainderf*N*x
remquof*N*x
copysignf*N*x
nanf*N*x
nextafterf*N*x
nextupf*N*x
nextdownf*N*x
canonicalizef*N*x
fdimf*N*x
fmaxf*N*x
fminf*N*x
fmaxmagf*N*x
fminmagf*N*x
fmaf*N*x
totalorderf*N*x
totalordermagf*N*x