Section 14: Exceptions and IEEE arithmetic

The intrinsic modules IEEE_EXCEPTIONS, IEEE_ARITHMETIC, and IEEE_FEATURES provide support for exceptions and IEEE arithmetic. Whether the modules are provided is processor dependent. If the module IEEE_FEATURES is provided, which of the named constants defined in this standard are included is processor dependent. The module IEEE_ARITHMETIC behaves as if it contained a USE statement for IEEE_EXCEPTIONS and everything that is public in IEEE_EXCEPTIONS is public in IEEE_ARITHMETIC.

When IEEE_EXCEPTIONS or IEEE_ARITHMETIC is accessible, IEEE_OVERFLOW and IEEE_DIVIDE_BY_ZERO are supported in the scoping unit for all kinds of real and complex data. Which other exceptions are supported can be determined by the function IEEE_SUPPORT_FLAG, (14.9.24), and whether control of halting is supported can be determined by the function IEEE_SUPPORT_HALTING. The extent of support of the other exceptions may be influenced by the accessibility of the named constants IEEE_INEXACT_FLAG, IEEE_INVALID_FLAG, and IEEE_UNDERFLOW_FLAG of the module IEEE_FEATURES. If a scoping unit has access to IEEE_UNDERFLOW_FLAG of IEEE_FEATURES, the scoping unit shall support underflow and return true from IEEE_SUPPORT_FLAG(IEEE_UNDERFLOW, X) for at least one kind of real. Similarly, if IEEE_INEXACT_FLAG or IEEE_INVALID_FLAG is accessible, the scoping unit shall support the exception and return true from the corresponding inquiry for at least one kind of real. Also, if IEEE_HALTING is accessible, the scoping unit shall support control of halting and return true from IEEE_SUPPORT_HALTING(FLAG) for the flag.

If a scoping unit does not access IEEE_EXCEPTIONS or IEEE_ARITHMETIC, the level of support is processor dependent, and need not include support for any exceptions. If a flag is signaling on entry to such a scoping unit, the processor ensures that it is signaling on exit. If a flag is quiet on entry to such a scoping unit, whether it is signaling on exit is processor dependent.

For processors with IEEE arithmetic, further IEEE support is available through the module IEEE_ARITHMETIC. The extent of support may be influenced by the accessibility of the named constants of the module IEEE_FEATURES. If a scoping unit has access to IEEE_DATATYPE of IEEE_FEATURES, the scoping unit shall support IEEE arithmetic and return true from IEEE_SUPPORT_DATATYPE(X) (14.9.21) for at least one kind of real. Similarly, if IEEE_DENORMAL, IEEE_DIVIDE, IEEE_INF, IEEE_NAN, IEEE_ROUNDING, or IEEE_SQRT is accessible, the scoping unit shall support the feature and return true from the corresponding inquiry function for at least one kind of real. In the case of IEEE_ROUNDING, it shall return true for all the rounding modes IEEE_NEAREST, IEEE_TO_ZERO, IEEE_UP, and IEEE_DOWN.

Execution might be slowed on some processors by the support of some features. If IEEE_EXCEPTIONS or IEEE_ARITHMETIC is accessed but IEEE_FEATURES is not accessed, the supported subset of features is processor dependent. The processor's fullest support is provided when all of IEEE_FEATURES is accessed as in

```
USE, INTRINSIC :: IEEE_ARITHMETIC; USE, INTRINSIC :: IEEE_FEATURES
```

but execution might then be slowed by the presence of a feature that is not needed. In all cases, the extent of support can be determined by the inquiry functions.

NOTE 14.1

The types and procedures defined in these modules are not themselves intrinsic.

14.1 Derived data types defined in the modules

The modules IEEE_EXCEPTIONS, IEEE_ARITHMETIC, and IEEE_FEATURES define five derived types, whose components are all private.

The module IEEE_EXCEPTIONS defines

- IEEE_FLAG_TYPE, for identifying a particular exception flag. Its only possible values are those of named constants defined in the module: IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, and IEEE_INEXACT. The module also defines the array named constants IEEE_USUAL = (/ IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_INVALID /) and IEEE_ALL = (/ IEEE_USUAL, IEEE_UNDERFLOW, IEEE_INEXACT /).
- IEEE_STATUS_TYPE, for saving the current floating point status.

The module IEEE_ARITHMETIC defines

- IEEE_CLASS_TYPE, for identifying a class of floating–point values. Its only possible values are those of named constants defined in the module: IEEE_SIGNALING_NAN, IEEE_QUIET_NAN, IEEE_NEGATIVE_INF, IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_DENORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO, IEEE_POSITIVE_DENORMAL, IEEE_POSITIVE_NORMAL, IEEE_POSITIVE_INF.
- IEEE_ROUND_TYPE, for identifying a particular rounding mode. Its only possible values are those of named constants defined in the module: IEEE_NEAREST, IEEE_TO_ZERO, IEEE_UP, and IEEE_DOWN for the IEEE modes; and IEEE_OTHER for any other mode.
- The elemental operator == for two values of one of these types to return true if the values are the same and false otherwise.
- The elemental operator /= for two values of one of these types to return true if the values differ and false otherwise.

The module IEEE_FEATURES defines

- IEEE_FEATURES_TYPE, for expressing the need for particular IEEE features. Its only possible values are those of named constants defined in the module: IEEE_DATATYPE, IEEE_DENORMAL, IEEE_DIVIDE, IEEE_HALTING, IEEE_INEXACT_FLAG, IEEE_INF, IEEE_INVALID_FLAG, IEEE_NAN, IEEE_ROUNDING, IEEE_SQRT, and IEEE_UNDERFLOW_FLAG.

14.2 The exceptions

The exceptions are

- IEEE_OVERFLOW

  This exception occurs when the result for an intrinsic real operation or assignment has an absolute value greater than a processor–dependent limit, or the real or imaginary part of the result for an intrinsic complex operation or assignment has an absolute value greater than a processor–dependent limit.

- IEEE_DIVIDE_BY_ZERO

  This exception occurs when a real or complex division has a nonzero numerator and a zero denominator.

- IEEE_INVALID

  This exception occurs when a real or complex operation or assignment is invalid; examples are SQRT(X) when X is real and has a nonzero negative value, and conversion to an integer (by assignment or an intrinsic procedure) when the result is too large to be representable.

- IEEE_UNDERFLOW

This exception occurs when the result for an intrinsic real operation or assignment has an absolute value less than a processor–dependent limit and loss of accuracy is detected, or the real or imaginary part of the result for an intrinsic complex operation or assignment has an absolute value less than a processor–dependent limit and loss of accuracy is detected.

- IEEE_INEXACT

  This exception occurs when the result of a real or complex operation or assignment is not exact.

Each exception has a flag whose value is either quiet or signaling. The value can be determined by the function IEEE_GET_FLAG. Its initial value is quiet and it signals when the associated exception occurs. Its status can also be changed by the subroutine IEEE_SET_FLAG or the subroutine IEEE_SET_STATUS. Once signaling, it remains signaling unless set quiet by an invocation of the subroutine IEEE_SET_FLAG or the subroutine IEEE_SET_STATUS.

If a flag is signaling on entry to a procedure, the processor will set it to quiet on entry and restore it to signaling on return.

Evaluation of a specification expression may cause an exception to signal.

In a scoping unit that has access to IEEE_EXCEPTIONS or IEEE_ARITHMETIC, if an intrinsic procedure executes normally, the values of the flags IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, and IEEE_INVALID shall be as on entry to the procedure, even if one or more signals during the calculation. If a real or complex result is too large for the intrinsic to handle, IEEE_OVERFLOW may signal. If a real or complex result is a NaN because of an invalid operation (for example, LOG(-1.0)), IEEE_INVALID may signal. Similar rules apply to format processing and to intrinsic operations: no signaling flag shall be set quiet and no quiet flag shall be set signaling because of an intermediate calculation that does not affect the result.

NOTE 14.2

An implementation may provide alternative versions of an intrinsic procedure; a practical example of such alternatives might be one version suitable for a call from a scoping unit with access to IEEE_EXCEPTIONS or IEEE_ARITHMETIC and one for other cases.

In a sequence of statements that has no invocations of IEEE_GET_FLAG, IEEE_SET_FLAG, IEEE_GET_STATUS, IEEE_SET_HALTING, or IEEE_SET_STATUS, if the execution of an operation would cause an exception to signal but after execution of the sequence no value of a variable depends on the operation, whether the exception is signaling is processor dependent. For example, when Y has the value zero, whether the code

```
X = 1.0/Y
X = 3.0
```

signals IEEE_DIVIDE_BY_ZERO is processor dependent. Another example is the following:

```
REAL, PARAMETER :: X=0.0, Y=6.0
IF (1.0/X == Y) PRINT *,'Hello world'
```

where the processor is permitted to discard the IF statement since the logical expression can never be true and no value of a variable depends on it.

An exception shall not signal if this could arise only during execution of an operation further to those required or permitted by the standard. For example, the statement

```
IF (F(X)>0.0) Y = 1.0/Z
```

shall not signal IEEE_DIVIDE_BY_ZERO when both F(X) and Z are zero and the statement

```
WHERE(A>0.0) A = 1.0/A
```

shall not signal IEEE_DIVIDE_BY_ZERO. On the other hand, when X has the value 1.0 and Y has the value 0.0, the expression

```
X>0.00001 .OR. X/Y>0.00001
```

is permitted to cause the signaling of IEEE_DIVIDE_BY_ZERO.

The processor need not support IEEE_INVALID, IEEE_UNDERFLOW, and IEEE_INEXACT. If an exception is not supported, its flag is always quiet. The function IEEE_SUPPORT_FLAG can be used to inquire whether a particular flag is supported. If IEEE_INVALID is supported, it signals in the case of conversion to an integer (by assignment or an intrinsic procedure) if the result is too large to be representable.

14.3 The rounding modes

The IEEE standard specifies four rounding modes:

- IEEE_NEAREST rounds the exact result to the nearest representable value.
- IEEE_TO_ZERO rounds the exact result towards zero to the next representable value.
- IEEE_UP rounds the exact result towards +infinity to the next representable value.
- IEEE_DOWN rounds the exact result towards -infinity to the next representable value.

The function IEEE_GET_ROUNDING_MODE can be used to inquire which rounding mode is in operation. Its value is one of the above four or IEEE_OTHER if the rounding mode does not conform to the IEEE standard.

If the processor supports the alteration of the rounding mode during execution, the subroutine IEEE_SET_ROUNDING_MODE can be used to alter it. The function IEEE_SUPPORT_ROUNDING can be used to inquire whether this facility is available for a particular mode. The function IEEE_SUPPORT_IO can be used to inquire whether rounding for base conversion in formatted input/output (9.4.5.12, 9.5.1.12, 10.7.7) is as specified in the IEEE standard.

In a procedure other than IEEE_SET_ROUNDING_MODE or IEEE_SET_STATUS, the processor shall not change the rounding mode on entry, and on return shall ensure that the rounding mode is the same as it was on entry.

NOTE 14.3

Within a program, all literal constants that have the same form have the same value (4.1.2). Therefore, the value of a literal constant is not affected by the rounding mode.

14.4 Halting

Some processors allow control during program execution of whether to abort or continue execution after an exception. Such control is exercised by invocation of the subroutine IEEE_SET_HALTING_MODE. Halting is not precise and may occur any time after the exception has occurred. The function IEEE_SUPPORT_HALTING can be used to inquire whether this facility is available. The initial halting mode is processor dependent. In a procedure other than IEEE_SET_HALTING_MODE or IEEE_SET_STATUS, the processor shall not change the halting mode on entry, and on return shall ensure that the halting mode is the same as it was on entry.

14.5 The floating point status

The values of all the supported flags for exceptions, rounding mode, and halting are called the floating point status. The floating point status can be saved in a scalar variable of type TYPE(IEEE_STATUS_TYPE) with the subroutine IEEE_GET_STATUS and restored with the subroutine IEEE_SET_STATUS. There are no facilities for finding the values of particular flags

held within such a variable. Portions of the floating point status can be saved with the subroutines IEEE_GET_FLAG, IEEE_GET_HALTING_MODE, and IEEE_GET_ROUNDING_MODE, and set with the subroutines IEEE_SET_FLAG, IEEE_SET_HALTING_MODE, and IEEE_SET_ROUNDING_MODE.

NOTE 14.4

Some processors hold all these flags in a floating point status register that can be saved and restored as a whole much faster than all individual flags can be saved and restored. These procedures are provided to exploit this feature.

NOTE 14.5

The processor is required to ensure that a call to a Fortran procedure does not change the floating point status other than by setting exception flags to signaling. No such requirements can be placed on procedures defined by means other than Fortran. For such procedures, it is the responsibility of the user to ensure that the floating point status is preserved.

14.6 Exceptional values

The IEEE standard specifies the following exceptional floating point values:

- Denormalized values have very small absolute values and lowered precision.
- Infinite values (+infinity and -infinity) are created by overflow or division by zero.
- Not–a–Number (NaN) values are undefined values or values created by an invalid operation.

In this standard, the term **normal** is used for values that are not in one of these exceptional classes.

The functions IEEE_IS_FINITE, IEEE_IS_NAN, IEEE_IS_NEGATIVE, and IEEE_IS_NORMAL are provided to test whether a value is finite, NaN, negative, or normal. The function IEEE_VALUE is provided to generate an IEEE number of any class, including an infinity or a NaN. The functions IEEE_SUPPORT_DENORMAL, IEEE_SUPPORT_DIVIDE, IEEE_SUPPORT_INF, and IEEE_SUPPORT_NAN can be used to inquire whether this facility is available for a particular kind of real.

14.7 IEEE arithmetic

The function IEEE_SUPPORT_DATATYPE can be used to inquire whether IEEE arithmetic is available for a particular kind of real. Complete conformance with the IEEE standard is not required, but the normalized numbers shall be exactly those of IEEE single or IEEE double; the arithmetic operators shall be implemented with at least one of the IEEE rounding modes; and the functions copysign, scalb, logb, nextafter, rem, and unordered shall be provided by the functions IEEE_COPY_SIGN, IEEE_SCALB, IEEE_LOGB, IEEE_NEXT_AFTER, IEEE_REM, and IEEE_UNORDERED. The inquiry function IEEE_SUPPORT_DIVIDE is provided to inquire whether the processor supports divide with the accuracy specified by the IEEE standard. For each of the other arithmetic operators and for each implemented IEEE rounding mode, the result shall be as specified in the IEEE standard whenever the operands and IEEE result are normalized.

The inquiry function IEEE_SUPPORT_NAN is provided to inquire whether the processor supports IEEE NaNs. Where these are supported, their behavior for unary and binary operations, including those defined by intrinsic functions and by functions in intrinsic modules, is as specified in the IEEE standard.

The inquiry function IEEE_SUPPORT_INF is provided to inquire whether the processor supports IEEE infinities. Where these are supported, their behavior for unary and binary operations, including those defined by intrinsic functions and by functions in intrinsic modules, is as specified in the IEEE standard.

The IEEE standard specifies a square root function that returns -0.0 for the square root of -0.0. The function IEEE_SUPPORT_SQRT can be used to inquire whether SQRT is implemented in accord with the IEEE standard for a particular kind of real.

The inquiry function IEEE_SUPPORT_STANDARD is provided to inquire whether the processor supports all the IEEE facilities defined in this standard for a particular kind of real.

14.8 Tables of the procedures

In this section, the procedures defined in the modules are tabulated with the names of their arguments and a short description.

14.8.1 Inquiry functions

The module IEEE_EXCEPTIONS contains the following inquiry functions:

- IEEE_SUPPORT_FLAG(FLAG [, X]) Inquire whether the processor supports an exception.
- IEEE_SUPPORT_HALTING(FLAG) Inquire whether the processor supports control of halting after an exception.

The module IEEE_ARITHMETIC contains the following inquiry functions:

- IEEE_SUPPORT_DATATYPE([X]) Inquire whether the processor supports IEEE arithmetic.
- IEEE_SUPPORT_DENORMAL([X]) Inquire whether the processor supports denormalized numbers.
- IEEE_SUPPORT_DIVIDE([X]) Inquire whether the processor supports divide with the accuracy specified by the IEEE standard.
- IEEE_SUPPORT_INF([X]) Inquire whether processor supports the IEEE infinity.
- IEEE_SUPPORT_IO([X]) Inquire whether the processor supports IEEE base conversion rounding during formatted input/output.
- IEEE_SUPPORT_NAN([X]) Inquire whether processor supports the IEEE Not–A–Number.
- IEEE_SUPPORT_ROUNDING(ROUND_VALUE [, X]) Inquire whether processor supports a particular rounding mode.
- IEEE_SUPPORT_SQRT([X]) Inquire whether the processor supports IEEE square root.
- IEEE_SUPPORT_STANDARD([X]) Inquire whether processor supports all IEEE facilities.

14.8.2 Elemental functions

The module IEEE_ARITHMETIC contains the following elemental functions for reals X and Y for which IEEE_SUPPORT_DATATYPE(X) and IEEE_SUPPORT_DATATYPE(Y) are true:

- IEEE_CLASS(X) IEEE class.
- IEEE_COPY_SIGN(X,Y) IEEE copysign function.
- IEEE_IS_FINITE(X) Determine if value is finite.
- IEEE_IS_NAN(X) Determine if value is IEEE Not–a–Number.
- IEEE_IS_NORMAL(X) Whether a value is normal, that is, neither an infinity, a NaN, nor denormalized.
- IEEE_IS_NEGATIVE(X) Determine if value is negative.
- IEEE_LOGB(X) Unbiased exponent in the IEEE floating point format.

- IEEE_NEXT_AFTER(X,Y) Returns the next representable neighbor of X in the direction toward Y.
- IEEE_REM(X,Y) The IEEE REM function, that is X - Y*N, where N is the integer nearest to the exact value X/Y.
- IEEE_RINT(X) Round to an integer value according to the current rounding mode.
- IEEE_SCALB(X,I) Returns 2^I X.
- IEEE_UNORDERED(X,Y) IEEE unordered function. True if X or Y is a NaN and false otherwise.
- IEEE_VALUE(X, CLASS) Generate an IEEE value.

14.8.3 Kind function

The module IEEE_ARITHMETIC contains the following transformational function:

- IEEE_SELECTED_REAL_KIND ([P,][R]) Kind type parameter value for an IEEE real with given precision and range.

14.8.4 Elemental subroutines

The module IEEE_EXCEPTIONS contains the following elemental subroutines:

- IEEE_GET_FLAG(FLAG,FLAG_VALUE) Get an exception flag.
- IEEE_GET_HALTING_MODE(FLAG, HALTING) Get halting mode for an exception.
- IEEE_SET_FLAG(FLAG,FLAG_VALUE) Set an exception flag.
- IEEE_SET_HALTING_MODE(FLAG,HALTING) Controls continuation or halting on exceptions.

14.8.5 Nonelemental subroutines

The module IEEE_EXCEPTIONS contains the following nonelemental subroutines:

- IEEE_GET_STATUS(STATUS_VALUE) Get the current state of the floating point environment .
- IEEE_SET_STATUS(STATUS_VALUE) Restore the state of the floating point environment.

The module IEEE_ARITHMETIC contains the following nonelemental subroutines:

- IEEE_GET_ROUNDING_MODE(ROUND_VALUE) Get the current IEEE rounding mode.
- IEEE_SET_ROUNDING_MODE(ROUND_VALUE) Set the current IEEE rounding mode.

14.9 Specifications of the procedures

In this section, the procedures are described in detail. The procedure names are generic and are not specific. All the functions are pure. In the examples, it is assumed that the processor supports IEEE arithmetic for default real.

NOTE 14.6

It is intended that a processor should not check a condition given in a paragraph labeled "**Restriction**" at compile time, but rather should rely on the programmer writing code such as

```
IF (IEEE_SUPPORT_DATATYPE(X)) THEN
  C = IEEE_CLASS(X)
ELSE
  .
  .
ENDIF
```

to avoid a call ever being made on a processor for which the condition is violated.

For the elemental functions of IEEE_ARITHMETIC, as tabulated in 14.8.2, if X or Y has a value that is an infinity or a NaN, the result shall be consistent with the general rules in 6.1 and 6.2 of the IEEE standard. For example, the result for an infinity shall be constructed as the limiting case of the result with a value of arbitrarily large magnitude, if such a limit exists.

14.9.1 IEEE_CLASS (X)

Description. IEEE class function.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. TYPE(IEEE_CLASS_TYPE).

Result Value. The result value is one of IEEE_SIGNALING_NAN, IEEE_QUIET_NAN, IEEE_NEGATIVE_INF, IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_DENORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO, IEEE_POSITIVE_DENORMAL, IEEE_POSITIVE_NORMAL, or IEEE_POSITIVE_INF. Neither of the values IEEE_SIGNALING_NAN and IEEE_QUIET_NAN shall be returned unless IEEE_SUPPORT_NAN(X) has the value true. Neither of the values IEEE_NEGATIVE_INF and IEEE_POSITIVE_INF shall be returned unless IEEE_SUPPORT_INF(X) has the value true. Neither of the values IEEE_NEGATIVE_DENORMAL and IEEE_POSITIVE_DENORMAL shall be returned unless IEEE_SUPPORT_DENORMAL(X) has the value true.

Example. IEEE_CLASS(-1.0) has the value IEEE_NEGATIVE_NORMAL.

14.9.2 IEEE_COPY_SIGN (X, Y)

Description. IEEE copysign function.

Class. Elemental function.

Arguments. The arguments shall be of type real.

Restriction. A program is prohibited from invoking this procedure with X or Y such that IEEE_SUPPORT_DATATYPE(X) or IEEE_SUPPORT_DATATYPE(Y) has the value false.

Result Characteristics. Same as X.

Result Value. The result has the value of X with the sign of Y. This is true even for IEEE special values, such as a NaN or an infinity (on processors supporting such values).

Example. The value of IEEE_COPY_SIGN(X,1.0) is ABS(X) even when X is NaN.

14.9.3 IEEE_GET_FLAG (FLAG, FLAG_VALUE)

Description. Get an exception flag.

Class. Elemental subroutine.

Arguments.

FLAG shall be of type TYPE(IEEE_FLAG_TYPE). It is an INTENT(IN) argument and specifies the IEEE flag to be obtained.

FLAG_VALUE shall be of type default logical. It is an INTENT(OUT) argument. If the value of FLAG is IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, or IEEE_INEXACT, the result value is true if the corresponding exception flag is signaling and is false otherwise.

Example. Following CALL IEEE_GET_FLAG(IEEE_OVERFLOW,FLAG_VALUE), FLAG_VALUE is true if the IEEE_OVERFLOW flag is signaling and is false if it is quiet.

14.9.4 IEEE_GET_HALTING_MODE (FLAG, HALTING)

Description. Get halting mode for an exception.

Class. Elemental subroutine.

Arguments.

FLAG shall be of type TYPE(IEEE_FLAG_TYPE). It is an INTENT(IN) argument and specifies the IEEE flag. It shall have one of the values IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, and IEEE_INEXACT.

HALTING shall be of type default logical. It is of INTENT(OUT). The value is true if the exception specified by FLAG will cause halting. Otherwise, the value is false.

Example. To store the halting mode for IEEE_OVERFLOW, do a calculation without halting, and restore the halting mode later:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
LOGICAL HALTING
 ...
CALL IEEE_GET_HALTING_MODE(IEEE_OVERFLOW,HALTING) ! Store halting mode
CALL IEEE_SET_HALTING_MODE(IEEE_OVERFLOW,.FALSE.) ! No halting
 ...! calculation without halting
CALL IEEE_SET_HALTING_MODE(IEEE_OVERFLOW,HALTING) ! Restore halting mode
```

NOTE 14.7

The initial halting mode is processor dependent. Halting is not precise and may occur some time after the exception has occurred.

14.9.5 IEEE_GET_ROUNDING_MODE (ROUND_VALUE)

Description. Get the current IEEE rounding mode.

Class. Subroutine.

Argument. ROUND_VALUE shall be scalar of type TYPE(IEEE_ROUND_TYPE). It is an INTENT(OUT) argument and returns the floating point rounding mode, with value IEEE_NEAREST, IEEE_TO_ZERO, IEEE_UP, or IEEE_DOWN if one of the IEEE modes is in operation and IEEE_OTHER otherwise.

Example. To store the rounding mode, do a calculation with round to nearest, and restore the rounding mode later:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
```

```
TYPE(IEEE_ROUND_TYPE) ROUND_VALUE
 ...
CALL IEEE_GET_ROUNDING_MODE(ROUND_VALUE) ! Store the rounding mode
CALL IEEE_SET_ROUNDING_MODE(IEEE_NEAREST)
 ... ! calculation with round to nearest
CALL IEEE_SET_ROUNDING_MODE(ROUND_VALUE) ! Restore the rounding mode
```

NOTE 14.8

The result can validly be used only in an IEEE_SET_ROUNDING_MODE invocation.

14.9.6 IEEE_GET_STATUS (STATUS_VALUE)

Description. Get the current value of the floating point status (14.5).

Class. Subroutine.

Argument. STATUS_VALUE shall be scalar of type TYPE(IEEE_STATUS_TYPE). It is an INTENT(OUT) argument and returns the floating point status.

Example. To store all the exception flags, do a calculation involving exception handling, and restore them later:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
TYPE(IEEE_STATUS_TYPE) STATUS_VALUE
 ...
CALL IEEE_GET_STATUS(STATUS_VALUE) ! Get the flags
CALL IEEE_SET_FLAG(IEEE_ALL,.FALSE.) ! Set the flags quiet.
 ... ! calculation involving exception handling
CALL IEEE_SET_STATUS(STATUS_VALUE) ! Restore the flags
```

NOTE 14.9

The result can be used only in an IEEE_SET_STATUS invocation.

14.9.7 IEEE_IS_FINITE (X)

Description. Whether a value is finite.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical.

Result Value. The result has the value true if the value of X is finite, that is, IEEE_CLASS(X) has one of the values IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_DENORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO, IEEE_POSITIVE_DENORMAL, and IEEE_POSITIVE_NORMAL; otherwise, the result has the value false.

Example. IEEE_IS_FINITE(1.0) has the value true.

14.9.8 IEEE_IS_NAN (X)

Description. Whether a value is IEEE Not–a–Number.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_NAN(X) has the value false.

Result Characteristics. Default logical.

Result Value. The result has the value true if the value of X is an IEEE NaN; otherwise, it has the value false.

Example. IEEE_IS_NAN(SQRT(-1.0)) has the value true if IEEE_SUPPORT_SQRT(1.0) has the value true.

14.9.9 IEEE_IS_NEGATIVE (X)

Description. Whether a value is negative.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical.

Result Value. The result has the value true if IEEE_CLASS(X) has one of the values IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_DENORMAL, IEEE_NEGATIVE_ZERO and IEEE_NEGATIVE_INF; otherwise, the result has the value false.

Example. IEEE_IS_NEGATIVE(0.0)) has the value false.

14.9.10 IEEE_IS_NORMAL (X)

Description. Whether a value is normal, that is, neither an infinity, a NaN, nor denormalized.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical.

Result Value. The result has the value true if IEEE_CLASS(X) has one of the values IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO and IEEE_POSITIVE_NORMAL; otherwise, the result has the value false.

Example. IEEE_IS_NORMAL(SQRT(-1.0)) has the value false if IEEE_SUPPORT_SQRT(1.0) has the value true.

14.9.11 IEEE_LOGB (X)

Description. Unbiased exponent in the IEEE floating point format.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Same as X.

Result Value. Result Value.

Case (i): If the value of X is neither zero, infinity, nor NaN, the result has the value of the unbiased exponent of X. Note: this value is equal to EXPONENT(X)-1.

Case (ii): If X==0, the result is -infinity if IEEE_SUPPORT_INF(X) is true and -HUGE(X) otherwise; IEEE_DIVIDE_BY_ZERO signals.

Example. IEEE_LOGB(-1.1) has the value 0.0.

14.9.12 IEEE_NEXT_AFTER (X, Y)

Description. Returns the next representable neighbor of X in the direction toward Y.

Class. Elemental function.

Arguments. The arguments shall be of type real.

Restriction. A program is prohibited from invoking this procedure with X or Y such that IEEE_SUPPORT_DATATYPE(X) or IEEE_SUPPORT_DATATYPE(Y) has the value false.

Result Characteristics. Same as X.

Result Value.

Case (i): If X == Y, the result is X without any exception ever being signaled.

Case (ii): If X /= Y, the result has the value of the next representable neighbor of X in the direction of Y. The neighbors of zero (of either sign) are both nonzero. IEEE_OVERFLOW is signaled when X is finite but IEEE_NEXT_AFTER(X,Y) is infinite; IEEE_UNDERFLOW is signaled when IEEE_NEXT_AFTER(X,Y) is denormalized; in both cases, IEEE_INEXACT signals.

Example. The value of IEEE_NEXT_AFTER(1.0,2.0) is 1.0+EPSILON(X).

14.9.13 IEEE_REM (X, Y)

Description. IEEE REM function.

Class. Elemental function.

Arguments. The arguments shall be of type real.

Restriction. A program is prohibited from invoking this procedure with X or Y such that IEEE_SUPPORT_DATATYPE(X) or IEEE_SUPPORT_DATATYPE(Y) has the value false.

Result Characteristics. Real with the kind type parameter of whichever argument has the greater precision.

Result Value. The result value, regardless of the rounding mode, shall be exactly X - Y*N, where N is the integer nearest to the exact value X/Y; whenever |N - X/Y| = 1/2, N shall be even. If the result value is zero, the sign shall be that of X.

Examples. The value of IEEE_REM(4.0,3.0) is 1.0, the value of IEEE_REM(3.0,2.0) is -1.0, and the value of IEEE_REM(5.0,2.0) is 1.0.

14.9.14 IEEE_RINT (X)

Description. Round to an integer value according to the current rounding mode.

Class. Elemental function.

Argument. X shall be of type real.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Same as X.

Result Value. The value of the result is the value of X rounded to an integer according to the current rounding mode. If the result has the value zero, the sign is that of X.

Examples. If the current rounding mode is round–to–nearest, the value of IEEE_RINT(1.1) is 1.0. If the current rounding mode is round–up, the value of IEEE_RINT(1.1) is 2.0.

14.9.15 IEEE_SCALB (X, I)

Description. Returns 2^I X.

Class. Elemental function.

Arguments.

X	shall be of type real.
I	shall be of type integer.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Same as X.

Result Value.

Case (i): If $2^I X$ is representable as a normal number, the result has this value.

Case (ii): If X is finite and $2^I X$ is too large, the IEEE_OVERFLOW exception shall occur. If IEEE_SUPPORT_INF(X) is true, the result value is infinity with the sign of X; otherwise, the result value is SIGN(HUGE(X),X).

Case (iii): If $2^I X$ is too small and there is loss of accuracy, the IEEE_UNDERFLOW exception shall occur. The result is the nearest representable number with the sign of X.

Case (iv): If X is infinite, the result is the same as X; no exception signals.

Example. The value of IEEE_SCALB(1.0,2) is 4.0.

14.9.16 IEEE_SELECTED_REAL_KIND ([P, R])

Description. Returns a value of the kind type parameter of an IEEE real data type with decimal precision of at least P digits and a decimal exponent range of at least R. For data objects of such a type, IEEE_SUPPORT_DATATYPE(X) has the value true.

Class. Transformational function.

Arguments. At least one argument shall be present.

P (optional)	shall be scalar and of type integer.
R (optional)	shall be scalar and of type integer.

Result Characteristics. Default integer scalar.

Result Value. The result has a value equal to a value of the kind type parameter of an IEEE real data type with decimal precision, as returned by the function PRECISION, of at least P digits and a decimal exponent range, as returned by the function RANGE, of at least R, or if no such kind type parameter is available on the processor, the result is -1 if the precision is not available, -2 if the exponent range is not available, and -3 if neither is available. If more than one kind type parameter value meets the criteria, the value returned is the one with the smallest decimal precision, unless there are several such values, in which case the smallest of these kind values is returned.

Example. IEEE_SELECTED_REAL_KIND(6,70) has the value KIND(0.0) on a machine that supports IEEE single precision arithmetic for its default real approximation method.

14.9.17 IEEE_SET_FLAG (FLAG, FLAG_VALUE)

Description. Assign a value to an exception flag.

Class. Elemental subroutine.

Arguments.

FLAG	shall be of type TYPE(IEEE_FLAG_TYPE). It is an INTENT(IN) argument. If the value of FLAG is IEEE_INVALID, IEEE_OVERFLOW,

IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, or IEEE_INEXACT, the corresponding exception flag is assigned a value.

FLAG_VALUE shall be of type default logical. It is an INTENT(IN) argument. If it has the value true, the flag is set to be signaling; otherwise, the flag is set to be quiet.

Example. CALL IEEE_SET_FLAG(IEEE_OVERFLOW,.TRUE.) sets the IEEE_OVERFLOW flag to be signaling.

14.9.18 IEEE_SET_HALTING_MODE (FLAG, HALTING)

Description. Controls continuation or halting after an exception.

Class. Elemental subroutine.

Arguments.

FLAG shall be scalar and of type TYPE(IEEE_FLAG_TYPE). It is of INTENT(IN) and shall have one of the values IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, or IEEE_INEXACT.

HALTING shall be scalar and of type default logical. It is of INTENT(IN). If the value is true, the exception specified by FLAG will cause halting. Otherwise, execution will continue after this exception.

Restriction. A program is prohibited from invoking this procedure with a FLAG such that IEEE_SUPPORT_HALTING(FLAG) has the value false.

Example. CALL IEEE_SET_HALTING_MODE(IEEE_DIVIDE_BY_ZERO,.TRUE.) causes halting after a divide_by_zero exception.

NOTE 14.10

The initial halting mode is processor dependent. Halting is not precise and may occur some time after the exception has occurred.

14.9.19 IEEE_SET_ROUNDING_MODE (ROUND_VALUE)

Description. Set the current IEEE rounding mode.

Class. Subroutine.

Argument. ROUND_VALUE shall be scalar and of type TYPE(IEEE_ROUND_TYPE). It is an INTENT(IN) argument and specifies the mode to be set.

Restriction. A program is prohibited from invoking this procedure unless IEEE_SUPPORT_ROUNDING(ROUND_VALUE,X) is true for some X such that IEEE_SUPPORT_DATATYPE(X) is true.

Example. To store the rounding mode, do a calculation with round to nearest, and restore the rounding mode later:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
TYPE(IEEE_ROUND_TYPE) ROUND_VALUE
 ...
CALL IEEE_GET_ROUNDING_MODE(ROUND_VALUE) ! Store the rounding mode
CALL IEEE_SET_ROUNDING_MODE(IEEE_NEAREST)
: ! calculation with round to nearest
CALL IEEE_SET_ROUNDING_MODE(ROUND_VALUE) ! Restore the rounding mode
```

14.9.20 IEEE_SET_STATUS (STATUS_VALUE)

Description. Restore the value of the floating point status (14.5).

Class. Subroutine.

Argument. STATUS_VALUE shall be scalar and of type TYPE(IEEE_STATUS_TYPE). It is an INTENT(IN) argument. Its value shall have been set in a previous invocation of IEEE_GET_STATUS.

Example. To store all the exceptions flags, do a calculation involving exception handling, and restore them later:

```
USE, INTRINSIC :: IEEE_ARITHMETIC
TYPE(IEEE_STATUS_TYPE) STATUS_VALUE
 ...
CALL IEEE_GET_STATUS(STATUS_VALUE) ! Store the flags
CALL IEEE_SET_FLAGS(IEEE_ALL,.FALSE.) ! Set them quiet
 ... ! calculation involving exception handling
CALL IEEE_SET_STATUS(STATUS_VALUE) ! Restore the flags
```

NOTE 14.11

Getting and setting might be expensive operations. It is the programmer's responsibility to do it when necessary to assure correct results.

14.9.21 IEEE_SUPPORT_DATATYPE ([X])

Description. Inquire whether the processor supports IEEE arithmetic.

Class. Inquiry function.

Argument. X (optional) shall be scalar and of type real.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports IEEE arithmetic for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false. Here, support means using an IEEE data format and performing the binary operations of +, -, and * as in the IEEE standard whenever the operands and result all have normal values.

Example. IEEE_SUPPORT_DATATYPE(1.0) has the value true if default reals are implemented as in the IEEE standard except that underflowed values flush to zero instead of being denormal.

14.9.22 IEEE_SUPPORT_DENORMAL ([X])

Description. Inquire whether the processor supports IEEE denormalized numbers.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports arithmetic operations and assignments with denormalized numbers (biased exponent $e = 0$ and fraction $f \neq 0$, see section 3.2 of the IEEE standard) for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_DENORMAL(X) has the value true if the processor supports denormalized numbers for X.

NOTE 14.12

The denormalized numbers are not included in the 13.5 model for real numbers and all satisfy the inequality ABS(X) < TINY(X). They usually occur as a result of an arithmetic operation whose exact result is less than TINY(X). Such an operation causes IEEE_UNDERFLOW to signal unless the result is exact. IEEE_SUPPORT_DENORMAL(X) is false if the processor never returns a denormalized number as the result of an arithmetic operation.

14.9.23 IEEE_SUPPORT_DIVIDE ([X])

Description. Inquire whether the processor supports divide with the accuracy specified by the IEEE standard.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports divide with the accuracy specified by the IEEE standard for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_DIVIDE(X) has the value true if the processor supports IEEE divide for X.

14.9.24 IEEE_SUPPORT_FLAG (FLAG [, X])

Description. Inquire whether the processor supports an exception.

Class. Inquiry function.

Arguments.

FLAG	shall be scalar and of type TYPE(IEEE_FLAG_TYPE). Its value shall be one of IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, or IEEE_INEXACT.
X (optional)	shall of type real. It may be scalar or array valued.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports detection of the specified exception for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. ALL(IEEE_SUPPORT_FLAG(IEEE_ALL)) has the value true if the processor supports all the exceptions.

14.9.25 IEEE_SUPPORT_HALTING (FLAG)

Description. Inquire whether the processor supports the ability to control during program execution whether to abort or continue execution after an exception.

Class. Inquiry function.

Argument. FLAG shall be of type TYPE(IEEE_FLAG_TYPE). It is an INTENT(IN) argument. Its value shall be one of IEEE_INVALID, IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_UNDERFLOW, or IEEE_INEXACT.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports the ability to control during program execution whether to abort or continue execution after the exception specified by FLAG; otherwise, it has the value false. Here, support shall include the ability to change the mode by CALL IEEE_SET_HALTING(FLAG).

Example. IEEE_SUPPORT_HALTING(IEEE_OVERFLOW) has the value true if the processor supports control of halting after an overflow.

14.9.26 IEEE_SUPPORT_INF ([X])

Description. Inquire whether the processor supports the IEEE infinity facility.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports IEEE infinities (positive and negative) for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_INF(X) has the value true if the processor supports IEEE infinities for X.

14.9.27 IEEE_SUPPORT_IO ([X])

Description. Inquire whether the processor supports IEEE base conversion rounding during formatted input/output (9.4.5.12, 9.5.1.12, 10.7.7).

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports IEEE base conversion during formatted input/output (9.4.5.12, 9.5.1.12, 10.7.7) as described in the IEEE standard for the modes UP, DOWN, ZERO, and NEAREST for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_IO(X) has the value true if the processor supports IEEE base conversion for X.

14.9.28 IEEE_SUPPORT_NAN ([X])

Description. Inquire whether the processor supports the IEEE Not–A–Number facility.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result. The result has the value true if the processor supports IEEE NaNs for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_NAN(X) has the value true if the processor supports IEEE NaNs for X.

14.9.29 IEEE_SUPPORT_ROUNDING (ROUND_VALUE [,X])

Description. Inquire whether the processor supports a particular rounding mode for IEEE kinds of reals.

Class. Inquiry function.

Arguments.

ROUND_VALUE shall be of type TYPE(IEEE_ROUND_TYPE).

X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor supports the rounding mode defined by ROUND_VALUE for all reals (X absent) or for real variables of the same kind type parameter as X; otherwise, it has the value false. Here, support shall include the ability to change the mode by CALL IEEE_SET_ROUNDING_MODE(ROUND_VALUE).

Example. IEEE_SUPPORT_ROUNDING(IEEE_TO_ZERO) has the value true if the processor supports rounding to zero for all reals.

14.9.30 IEEE_SUPPORT_SQRT ([X])

Description. Inquire whether the processor implements SQRT in accord with the IEEE standard.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the processor implements SQRT in accord with the IEEE standard for all reals (X absent) of for real variables of the same kind type parameter as X; otherwise, it has the value false.

Example. IEEE_SUPPORT_SQRT(X) has the value true if the processor implements SQRT(X) in accord with the IEEE standard. In this case, SQRT(-0.0) has the value -0.0.

14.9.31 IEEE_SUPPORT_STANDARD ([X])

Description. Inquire whether the processor supports all the IEEE facilities defined in this standard.

Class. Inquiry function.

Argument. X (optional) shall of type real. It may be scalar or array valued.

Result Characteristics. Default logical scalar.

Result Value.

Case (i): If X is absent, the result has the value true if the results of all the functions IEEE_SUPPORT_DATATYPE(), IEEE_SUPPORT_DENORMAL(), IEEE_SUPPORT_DIVIDE(), IEEE_SUPPORT_FLAG(FLAG) for valid FLAG, IEEE_SUPPORT_HALTING(FLAG) for valid FLAG, IEEE_SUPPORT_INF(),

IEEE_SUPPORT_NAN(), IEEE_SUPPORT_ROUNDING(ROUND_VALUE) for valid ROUND_VALUE, and IEEE_SUPPORT_SQRT() are all true; otherwise, the result has the value false.

Case (ii): If X is present, the result has the value true if the results of all the functions IEEE_SUPPORT_DATATYPE(X), IEEE_SUPPORT_DENORMAL(X), IEEE_SUPPORT_DIVIDE(X), IEEE_SUPPORT_FLAG(FLAG,X) for valid FLAG, IEEE_SUPPORT_HALTING(FLAG) for valid FLAG, IEEE_SUPPORT_INF(X), IEEE_SUPPORT_NAN(X), IEEE_SUPPORT_ROUNDING(ROUND_VALUE,X) for valid ROUND_VALUE, and IEEE_SUPPORT_SQRT(X) are all true; otherwise, the result has the value false.

Example. IEEE_SUPPORT_STANDARD() has the value false if the processor supports both IEEE and non-IEEE kinds of reals.

14.9.32 IEEE_UNORDERED (X, Y)

Description. IEEE unordered function. True if X or Y is a NaN, and false otherwise.

Class. Elemental function.

Arguments. The arguments shall be of type real.

Restriction. A program is prohibited from invoking this procedure with X or Y such that IEEE_SUPPORT_DATATYPE(X) or IEEE_SUPPORT_DATATYPE(Y) has the value false.

Result Characteristics. Default logical.

Result Value. The result has the value true if X or Y is a NaN or both are NaNs; otherwise, it has the value false.

Example. IEEE_UNORDERED(0.0,SQRT(-1.0)) has the value true if IEEE_SUPPORT_SQRT(1.0) has the value true.

14.9.33 IEEE_VALUE (X, CLASS)

Description. Generate an IEEE value.

Class. Elemental function.

Arguments.

X	shall be of type real.
CLASS	shall be of type TYPE(IEEE_CLASS_TYPE). The value of is permitted to be: IEEE_SIGNALING_NAN or IEEE_QUIET_NAN if IEEE_SUPPORT_NAN(X) has the value true, IEEE_NEGATIVE_INF or IEEE_POSITIVE_INF if IEEE_SUPPORT_INF(X) has the value true, IEEE_NEGATIVE_DENORMAL or IEEE_POSITIVE_DENORMAL if IEEE_SUPPORT_DENORMAL(X) has the value true, IEEE_NEGATIVE_NORMAL, IEEE_NEGATIVE_ZERO, IEEE_POSITIVE_ZERO or IEEE_POSITIVE_NORMAL.

Restriction. A program is prohibited from invoking this procedure with an X such that IEEE_SUPPORT_DATATYPE(X) has the value false.

Result Characteristics. Same as X.

Result Value. The result value is an IEEE value as specified by CLASS. Although in most cases the value is processor dependent, the value shall not vary between invocations for any particular X kind type parameter and CLASS value.

Example. IEEE_VALUE(1.0,IEEE_NEGATIVE_INF) has the value -infinity.

14.10 Examples

NOTE 14.13

```
MODULE DOT
! Module for dot product of two real arrays of rank 1.
! The caller needs to ensure that exceptions do not cause halting.
   USE, INTRINSIC :: IEEE_EXCEPTIONS
   LOGICAL MATRIX_ERROR = .FALSE.
   INTERFACE OPERATOR(.dot.)
      MODULE PROCEDURE MULT
   END INTERFACE
CONTAINS
   REAL FUNCTION MULT(A,B)
      REAL, INTENT(IN) :: A(:),B(:)
      INTEGER I
      LOGICAL OVERFLOW
      IF (SIZE(A)/=SIZE(B)) THEN
         MATRIX_ERROR = .TRUE.
         RETURN
      END IF
! The processor ensures that IEEE_OVERFLOW is quiet
      MULT = 0.0
      DO I = 1, SIZE(A)
         MULT = MULT + A(I)*B(I)
      END DO
      CALL IEEE_GET_FLAG(IEEE_OVERFLOW,OVERFLOW)
      IF (OVERFLOW) MATRIX_ERROR = .TRUE.
   END FUNCTION MULT
END MODULE DOT
```

This module provides the dot product of two real arrays of rank 1. If the sizes of the arrays are different, an immediate return occurs with MATRIX_ERROR true. If overflow occurs during the actual calculation, the IEEE_OVERFLOW flag will signal and MATRIX_ERROR will be true.

NOTE 14.14

```
USE, INTRINSIC :: IEEE_EXCEPTIONS
USE, INTRINSIC :: IEEE_FEATURES, ONLY: IEEE_INVALID_FLAG
! The other exceptions of IEEE_USUAL (IEEE_OVERFLOW and
! IEEE_DIVIDE_BY_ZERO) are always available with IEEE_EXCEPTIONS
TYPE(IEEE_STATUS_TYPE) STATUS_VALUE
LOGICAL, DIMENSION(3) :: FLAG_VALUE
 ...
CALL IEEE_GET_STATUS(STATUS_VALUE)
CALL IEEE_SET_HALTING_MODE(IEEE_USUAL,.FALSE.) ! Needed in case the
!                  default on the processor is to halt on exceptions
CALL IEEE_SET_FLAG(IEEE_USUAL,.FALSE.)
! First try the "fast" algorithm for inverting a matrix:
MATRIX1 = FAST_INV(MATRIX) ! This shall not alter MATRIX.
CALL IEEE_GET_FLAG(IEEE_USUAL,FLAG_VALUE)
IF (ANY(FLAG_VALUE)) THEN
! "Fast" algorithm failed; try "slow" one:
   CALL IEEE_SET_FLAG(IEEE_USUAL,.FALSE.)
   MATRIX1 = SLOW_INV(MATRIX)
   CALL IEEE_GET_FLAG(IEEE_USUAL,FLAG_VALUE)
```

NOTE 14.14 *(Continued)*

```
   IF (ANY(FLAG_VALUE)) THEN
      WRITE (*, *) 'Cannot invert matrix'
      STOP
   END IF
END IF
CALL IEEE_SET_STATUS(STATUS_VALUE)
```

In this example, the function FAST_INV may cause a condition to signal. If it does, another try is made with SLOW_INV. If this still fails, a message is printed and the program stops. Note, also, that it is important to set the flags quiet before the second try. The state of all the flags is stored and restored.

NOTE 14.15

```
USE, INTRINSIC :: IEEE_EXCEPTIONS
LOGICAL FLAG_VALUE
 ...
CALL IEEE_SET_HALTING_MODE(IEEE_OVERFLOW,.FALSE.)
! First try a fast algorithm for inverting a matrix.
CALL IEEE_SET_FLAG(IEEE_OVERFLOW,.FALSE.)
DO K = 1, N
 ...
   CALL IEEE_GET_FLAG(IEEE_OVERFLOW,FLAG_VALUE)
   IF (FLAG_VALUE) EXIT
END DO
IF (FLAG_VALUE) THEN
! Alternative code which knows that K-1 steps have executed normally.
 ...
END IF
```

Here the code for matrix inversion is in line and the transfer is made more precise by adding extra tests of the flag.

NOTE 14.16

```
REAL FUNCTION HYPOT(X, Y)
! In rare circumstances this may lead to the signaling of IEEE_OVERFLOW
! The caller needs to ensure that exceptions do not cause halting.
   USE, INTRINSIC :: IEEE_ARITHMETIC
   USE, INTRINSIC :: IEEE_FEATURES, ONLY: IEEE_UNDERFLOW_FLAG
! IEEE_OVERFLOW is always available with IEEE_ARITHMETIC
   REAL X, Y
   REAL SCALED_X, SCALED_Y, SCALED_RESULT
   LOGICAL, DIMENSION(2) :: FLAGS
   TYPE(IEEE_FLAG_TYPE), PARAMETER, DIMENSION(2) :: &
         OUT_OF_RANGE = (/ IEEE_OVERFLOW, IEEE_UNDERFLOW /)
   INTRINSIC SQRT, ABS, EXPONENT, MAX, DIGITS, SCALE
! The processor clears the flags on entry
! Try a fast algorithm first
   HYPOT = SQRT( X**2 + Y**2 )
   CALL IEEE_GET_FLAG(OUT_OF_RANGE,FLAGS)
   IF ( ANY(FLAGS) ) THEN
      CALL IEEE_SET_FLAG(OUT_OF_RANGE,.FALSE.)
      IF ( X==0.0 .OR. Y==0.0 ) THEN
         HYPOT = ABS(X) + ABS(Y)
      ELSE IF ( 2*ABS(EXPONENT(X)-EXPONENT(Y)) > DIGITS(X)+1 ) THEN
         HYPOT = MAX( ABS(X), ABS(Y) )! one of X and Y can be ignored
```

NOTE 14.16 *(Continued)*

```
      ELSE ! scale so that ABS(X) is near 1
         SCALED_X = SCALE( X, –EXPONENT(X) )
         SCALED_Y = SCALE( Y, –EXPONENT(X) )
         SCALED_RESULT = SQRT( SCALED_X**2 + SCALED_Y**2 )
         HYPOT = SCALE( SCALED_RESULT, EXPONENT(X) ) ! may cause overflow
      END IF
   END IF
! The processor resets any flag that was signaling on entry
END FUNCTION HYPOT
```

An attempt is made to evaluate this function directly in the fastest possible way. This will work almost every time, but if an exception occurs during this fast computation, a safe but slower way evaluates the function. This slower evaluation might involve scaling and unscaling, and in (very rare) extreme cases this unscaling can cause overflow (after all, the true result might overflow if X and Y are both near the overflow limit). If the IEEE_OVERFLOW or IEEE_UNDERFLOW flag is signaling on entry, it is reset on return by the processor, so that earlier exceptions are not lost.