Date: January 18, 1997
To: X3J3
From: R. Baker Kearfott
Subject: Interval Arithmetic - The Data Type and Low-Level Operations
References: X3J3/96-156, X3J3/96-065

Record of updates and explanations

1. January 18, 1997: Updated from X3J3/96-156 as follows:

   a) A non-ASCII character in place of "+" was corrected.

   b) The section on mixed mode operations and conversions was revised to incorporate extensive discussion within X3J3 and within the group consisting of me, Keith Bierman, George Corliss, David Hough, Andrew Pitonyak, Michael Schulte, William Walster, and Wolfgang Walter, with additional comments from Dick Hendrickson.

   c) The mixed mode conversion function INTERVAL produces compile-time errors when the arguments may not be machine representable. This is to both satisfy the condition that intervals contain true results, in a way that does not surprise the user, and to be consistent with Fortran's definition of "constant" and "value." However, it is probably not ideal from the point of view of interval computations. Bill Walster has proposed an alternate definition of mixed mode conversions that may be better from the point of view of users of interval computations. Bill's alternate proposal is appended to this document.

      (A third alternative, not developed in detail here, is to disallow mixed mode arithmetic, but have a constructor that essentially just stores endpoints.)

   The next steps:

   a) Approve the mixed-mode specifications

   b) Study the interval intrinsic functions (as briefly described in X3J3/96-065), and describe any necessary details specific to the individual functions.

==========

Name and Structure

The INTERVAL type is a numeric type. Its values are closed and bounded real intervals which are defined by an ordered pair of real values. The first real value is the lower bound (or infimum), the second real value is the upper bound (or supremum). The lower bound shall be less than or equal to the upper bound. The continuum of real numbers between and including these two bounds (or endpoints) is said to be contained in the interval.

Note: Thus intervals contain real numbers that are not otherwise machine representable.

The actual internal structure of an interval data type is implementation-dependent, and the bit structure is not directly accessible to the user. That is, the user may change the values of the endpoints of an interval only through setting an entire interval, through the IVAL constructor described below, or through an implicit conversion to interval. The values of the lower bound and upper bound of an interval can be obtained with the functions INF and SUP defined below.

The precision of each interval data type shall correspond to the precision of a real data type. The kind type parameter of each supported interval type shall be the same as the kind type parameter of the corresponding real type.

Note: Although the interval data type is opaque, a common model of intervals is that of two reals. For example, for the interval data type corresponding to the DOUBLE PRECISION IEEE binary type, the lower bound can be viewed as an IEEE DOUBLE PRECISION type and the upper bound can be viewed as an IEEE DOUBLE PRECISION type.

Note: It is recommended that the default INTERVAL kind correspond to a REAL kind with a precision roughly that of an IEEE 754 "double" . It is the consensus of experts that interval arithmetic with precision roughly that of IEEE 754 "single" is of limited use.

Interval Constants

Both where literal constants are admitted in expressions and in input or output data, INTERVAL's shall be represented by a single REAL or INTEGER or a pair of REAL's, INTEGER's, or combinations thereof, beginning with "(<", separated by "," if there are two numbers, and ending in ">)". For example

(<1, 2>), (<1E0, 3>), (<1>), and (<.1234D5>)

are all valid INTERVAL constants. An INTERVAL constant specified by a single number is the same as an INTERVAL constant specified by two numbers, both of whose endpoints are equal to the single number. When such a decimal constant is converted to its internal representation, the internal representation shall contain the decimal constant, regardless of how many digits are specified by the decimal constant. For example, upon execution of the statement:

X = (<0.3141592653589793238462643383279502 8D+01>)

the interval X shall contain the smallest-width machine interval that contains the number 3.14159265358979323846264338327950 28.

Interval constants shall admit kind type parameters, such as in the construction (<1,2>)_2. An interval constant with no kind type parameter shall be of default type.

Note: Thus, on machines in which the interval data type X appearing in the example above contained endpoints with accuracy that corresponded to less than 35 decimal digits, the interval X would contain the mathematical number PI.

Arithmetic Operations

The four basic operations +, -, *, and / are defined to contain the ranges of the corresponding operations on real numbers. Specifically, let X = [xl,xu] and Y = [yl,yu] be intervals, where xl, xu, yl, and yu represent the lower and upper bounds of X and Y, respectively. Then:

X + Y shall contain the exact value [xl + yl, xu + yu],

X - Y shall contain the exact value [xl - yu, xu - yl],

X * Y shall contain the exact value
[min{xl*yl, xl*yu, xu*yl, xu*yu}, max{xl*yl, xl*yu, xu*yl, xu*yu}],

1 / X shall contain the exact value [1/xu, 1/xl] if xu < 0 or xl > 0, and shall signal 'denominator contains zero' otherwise.

X / Y shall contain the exact value
[min{xl/yu, xl/yl, xu/yu, xu/yl}, max{xl/yu, xl/yl, xu/yu, xu/yl}]

if yu < 0 or yl > 0, and shall signal 'denominator contains zero' otherwise.

Note: Particular processors may support an extended interval data type, in which division by intervals that contain zero yields a meaningful result. For example, a meaningful interpretation of [1,2]/[-1,2] is the set-theoretic union of the two intervals (-infinity,-1] and [1/2,infinity).

Note: Using floating point arithmetic, the operations on the right-hand sides may first be computed, then the lower bound may be rounded down to a number known to be less than or equal to the exact mathematical result, and the upper bound may be rounded up to a number known to be greater than or equal to the exact mathematical result. If a processor provides directed roundings upwards (towards plus infinity) and downwards (towards minus infinity), then the operation and the rounding can be performed in one step, e.g. if the processor conforms to the IEEE 754 standard. The excess interval width caused by this outward rounding is called ROUNDOUT ERROR.

Note: There is an alternate implementation of interval multiplication that also gives the range of the real operator "*" over the intervals X and Y. This alternative involves nine cases determined by the algebraic signs of the endpoints of X and Y; see page 12 of R. E. Moore, "Methods and Applications of Interval Computations," SIAM, Philadelphia, 1979. The average number of multiplications required for this alternative is less than above, but one or more comparisons are required.
Implemented in software, the relative efficiencies of the alternative above and the nine-case alternative are architecture-dependent, although the nine-case alternative is often preferred in low-level implementations designed for efficiency.

Note: The only processor requirement is that the computed intervals contain the exact mathematical range of the corresponding point operations. In an ideal implementation (not required), the result of the operations is the smallest-width machine interval that contains the exact mathematical range.

Note: IEEE arithmetic can be used to perform ideal (minimum width) interval operations. For example, take

$$[xl,xu] + [yl,yu] = [xl+yl,xu+yu]$$

in exact interval arithmetic. The IEEE 754 standard defines a downwardly (upwardly) rounded operation as producing the same result as would be obtained by computing the exact result, then rounding it to the nearest floating point number less (greater) than or equal to the exact result. Thus, if the result xl+yl is rounded down and xu+yu is rounded up according to the IEEE specifications, the result is an ideal interval addition.

Mixed Mode Operations and Conversions

Explicit conversion to interval is performed with the conversion function INTERVAL as follows:

```
Z = INTERVAL(R[,S][,KIND=<<kind>>][,EPSILON=<<fuzz>>])
```

where Z is an INTERVAL, and R and S are INTEGER or REAL or COMPLEX. If KIND=<<kind>> is present, then <<kind>> is a valid kind parameter for the INTERVAL data type, and the converted value shall be an INTERVAL of kind <<kind>>. If KIND =<<kind>> is not present, then the result of INTERVAL shall be of default INTERVAL kind. If both R and S are present, then R shall be less than or equal to S. If S is absent, then the conversion shall be as if S were present and equal to R. If <<fuzz>> is present, then <<fuzz>> is INTEGER, REAL, or INTERVAL. If <<fuzz>> is an INTERVAL value, it shall be as if it were a REAL value of corresponding type, and value equal to SUP(<<fuzz>>). If any argument to INTERVAL is complex, it shall be as if the arguments were real and

equal to the real part of the actual value.

In all cases, where containment cannot be guaranteed, a compile-time error shall be issued. Variables and variable expressions, regardless of their history, are assumed to be exactly machine representable.

The interval value stored in Z shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

Examples: Z = INTERVAL(1) results in an interval of default kind that contains the value 1.

Z = INTERVAL(0.5) results in an interval of default kind that contains the value 0.5.

If the statement Z = INTERVAL(0.1) is encountered, a compile-time error is issued on machines that don't use fractional decimal representations. Similarly, if the statement Z = INTERVAL(RPARAM) is encountered, where RPARAM is a named REAL constant that may not be an exact representation of its corresponding exact decimal expression, then a compile-time error is issued.

Z = INTERVAL(R) results in an interval of default kind that contains the real value R.

Note: When compile-time errors are issued is processor-dependent in certain cases. The expression Z = INTERVAL(0.1) shall always result in an error on processors without fractional decimal arithmetic, since 0.1 is not exactly representable. However, a particular processor may ascertain through analysis that a named REAL constant is exactly represented, whereas another will issue an error.

If EPSILON=<<fuzz>> is absent, then the lower bound of Z shall be less than or equal to R, and the upper bound of Z shall be greater than or equal to S.

If EPSILON=<<fuzz>> is present and <<fuzz>> is equal to zero, then Z shall be as if <<fuzz>> were absent.

If EPSILON=<<fuzz>> is present and <<fuzz>> is not equal to zero, then the lower bound of Z shall be less than or equal to R-eps, and the upper bound of Z shall be greater than or equal to S+eps, where eps = SUP(ABS(INTERVAL(<<fuzz>>)).

Note: Non machine-representable values of <<fuzz>> produce compile-time errors.

INTERVAL may also be used to convert between different kinds of INTERVAL: INTERVAL(Z,KIND=<<kind2>>) shall be an INTERVAL with kind number <<kind2>> that contains the value Z, where Z is an INTERVAL value of any available kind. If EPSILON=<<fuzz>> is present, then INTERVAL(Z[,KIND=<<kind2>>],EPSILON=<<fuzz>>) shall be an INTERVAL (of kind <<kind2>> if KIND=<<kind2>> is present) whose lower bound is less than or equal to INF(Z)-<<fuzz>> and whose upper bound is greater than or equal to SUP(Z)+<<fuzz>>.

Examples. INTERVAL((<3.141>),EPSILON=(<1e-3>)) will be an INTERVAL of default type that contains an interval centered at the exact value 3.141 and of semi-width exactly 1e-3.

If X is a default real variable, then INTERVAL(X,EPSILON=(<1e-3>)) will be an INTERVAL of default type that contains [X-.001,X+.001], where here .001 means the exact

decimal constant.

INTERVAL(3,EPSILON=0) will, in general, be an interval of default type that contains the number 3. On most processors, INTERVAL(3,EPSILON=0) can equal [3,3].

Mixed mode operations between INTERVAL's of one kind and another are permitted, as well as mixed mode operations between INTERVAL's and INTEGER and REAL. The result of a mixed mode operation between an INTERVAL and an INTEGER or REAL shall be an INTERVAL.

The operations shall be consistent with 7.1.4.2 of X3J3/96-007 (March 4, 1996 version). In a mixed-mode operation between INTERVAL's of different types, the kind type parameter of the result shall be that of the operand with greater decimal precision or is processor dependent if the operands have the same decimal precision. In a mixed-mode operation between an INTEGER and an INTERVAL, the kind type parameter of the result is that of the INTERVAL. In a mixed mode operation between an INTERVAL and a REAL, the kind type parameter of the result shall be that of the operand with greater decimal precision or is processor dependent if the operands have the same decimal precision, provided an INTERVAL of such a type is supported on the processor; if no such INTERVAL type is supported on the processor, then the result shall be an INTERVAL of type corresponding to the highest decimal precision supported on the processor.

When an implicit conversion occurs according to the above rules, it shall be as if a call to INTERVAL were used, with appropriate KIND=<<kind>> argument and without an EPSILON=<<fuzz>> argument.

Note: Mixed mode operations in general do not give appropriate enclosures, since REAL values may represent results of computations and conversions in which error has accumulated. Users are encouraged to explicitly use INTERVAL with an appropriate value of <<fuzz>> for conversion of variables, to use interval constants rather than converting real constants, and to use interval constants and interval arithmetic from the beginning where rigorous enclosures are desired.

Example. If X is an INTERVAL of default type, and R is a REAL variable, then X**2 + 3*X + R shall be equal to X**2 + INTERVAL(3)*X + INTERVAL(R)

Implicit conversion from INTERVAL to REAL and INTERVAL to INTEGER are defined. If the conversion is to REAL, then the converted value shall be equal to MID(X), where X the interval to be converted and where MID is defined below.
If the converted value is to be INTEGER, then the value shall be equal to INT(MID(X)). Such an implicit conversion from INTERVAL occurs only in an assignment statement. In particular, an INTERVAL value may be assigned to an INTEGER or a REAL.

Example: If X is an INTERVAL and R is a REAL, then
R = X
is identical to R = MID(X), which, in turn,
is identical to R = REAL(X).

Implicit conversion to interval in assignment statements is also allowed. The conversion from INTEGER or REAL to INTERVAL shall be as if a binary operation between the left member and the right member occurred.

Example: If X is an INTERVAL of default type, and
the default INTERVAL type corresponded
to DOUBLE PRECISION real, then
X = 0.3D0
is the same as
X = INTERVAL(0.3D0)

Note: The functions INF and SUP defined below may also be used to convert an INTERVAL to another data type.

New Infix Operators

The following infix operators shall be a part of standard interval support.

Syntax	function
Z = X.IS.Y	Z <-- intersection of X and Y, that is, [max{xl,yl},min{xu,yu}] if max{xl,yl} < = min{xu,yu} and signals an "intersection of disjoint intervals" otherwise.
Z = X.CH.Y	Z <-- [min{xl,yl}, max{xu,yu}] ("interval hull" of X and Y. The mnemonic is "convex hull")
X.SB.Y	.TRUE. if X is a subset of Y (i.e. if xl >= yl .AND. xu <= yu)
X.PRSB.y	.TRUE. if X is a proper subset of Y (i.e. if X.SB.Y .AND. (xl > yl .OR. xu < yu)
X.SP.Y	.TRUE. if and only if Y.SB.X is true (i.e. if xl <= yl .AND. xu >= yu)
X.PRSP.Y	.TRUE. if and only if Y.PSB.X is true (i.e. if Y.SB.X .AND. (yl > xl .OR. yu < xu)
X.DJ.Y	.TRUE. if X and Y are disjoint sets (i.e. if xl > yu or xu < yl)
R.IN.X	.TRUE. if the REAL value R is contained in the interval X (i.e. if xl <= R <= xu)

Note: Intervals are closed. So, if R.IN.X, then R may be equal to one of the endpoints of X.

Interval Versions of Relational Operators

The following relational operators shall be extended to interval operations, in the "certainly true" sense. That is, the result is .TRUE. if and only if it is true for each pair of real values taken from the corresponding interval values.

Syntax	function
X.LT.Y	.TRUE. if xu < yl
X.GT.Y	.TRUE. if xl > yu
X.LE.Y	.TRUE. if xu <= yl
X.GE.Y	.TRUE. if xl >= yu

Another set of relational operators, the POSSIBLY TRUE relationals, shall be defined as follows.

Syntax	function
X.PLT.Y	.TRUE. if xl < yu (i.e. if .NOT.(X.GE.Y))
X.PGT.Y	.TRUE. if xu > yl (i.e. if .NOT.(X.LE.Y))
X.PLE.Y	.TRUE. if xl <= yu (i.e. if .NOT.(X.GT.Y))
X.PGE.Y	.TRUE. if xu >= yl (i.e. if .NOT.(X.LT.Y))

Finally, equality and inequality of intervals are defined by viewing the intervals as sets.

Syntax	function
X.EQ.Y	.TRUE. if xl=yl and xu=yu
X.NE.Y	.TRUE. if .NOT. (X.EQ.Y)

Operator Precedence

The precedence of these operators is as follows:

Category of operation	Operators	Precedence
Extension	<defined-unary-op>	Highest
Numeric	**	.
Numeric	*, /, .IS.	.
Numeric	unary + or -	.
Numeric	binary + or -, .CH.	.
Character	//	.
Relational	.EQ., .NE., .LT., .PLT., .LE., .PLE., .GT., .PGT., .GE., .PGE., .SB., .PRSB., .SP., .PRSP., .DJ.,.IN.	.
Logical	.NOT.	.
Logical	.AND.	.
Logical	.OR.	.
Logical	.EQV. or .NEQV.	.
Extension	<defined-binary-op>	Lowest

Special Interval Functions

The following utility functions shall be provided for conversion from INTERVAL to REAL, etc.

Syntax	function	attainable accuracy
R = INF(X)	Lower bound of X	(the value in the lower storage unit of the interval datum X)
R = SUP(X)	Upper bound of X	(the value in the upper storage unit of the interval datum X)
R = MID(X)	Midpoint of X	(a floating point approximation, always greater than or equal to the value returned by INF and less than or equal to the value returned by SUP)
R = WID(X)	R <-- xu - xl "Width"	(the value shall be rounded up, to be greater than or equal to the actual value)
R = MAG(X)	R <-- max { |xl|, |xu| } "Magnitude"	
R = MIG(X)	R <-- | min { |xl|, |xu| } if .NOT.(0.IN.X), | 0 otherwise. "Mignitude"	

```
                        |
Z = ABS(X)    Z <-- |  [min{|x|}, max{|x|}]
                        |   x.IN.X    x.IN.X
```

Range of absolute value

Z = MAX(X,Y) Z <-- [max {xl,yl}, max {xu,yu}]

Range of maximum
MAX shall be extended analogously
for more than two
arguments.

Z = MIN(X,Y) Z <-- [min {xl,yl}, min {xu,yu}]

Range of minimum
MIN shall be extended analogously
for more than two
arguments.

N = NDIGITS(X) Number of leading decimal digits that are the same in xl and xu. n digits shall be counted as the same if rounding xl to the nearest decimal number with n significant digits gives the same result as rounding xu to the nearest decimal number with n significant digits. If xl=xu, , then NDIGITS(X) shall return PRECISION(R)+1, where R is any real of kind the same as the kind of the INTERVAL X (that is, whose precision is the same as the precision of X).

Note: On many machines, INF, SUP, MAG, MIG, ABS, MAX, and MIN can be exact, if the target is of a type that corresponds to the input. This is because these functions merely involve storing one of the endpoints of the interval into the target variable. Similarly, the conversion function IVAL can be exact on such machines if it specifies conversion from REAL data of corresponding type.

Except for NDIGITS, all of these special interval functions shall be elemental.

Note regarding NDIGITS: For example, if X = [0.1996,0.2004], then three leading decimal digits of this function are the same, and NDIGITS(X) is equal to 3. This is because, if .1996 and .2004 are each rounded to the nearest decimal number with three significant digits, they both round to .200, yet they round to different four-digit decimal numbers. This value can be computed as INT(-LOG10(2.004-1.996). In many cases, the value can be computed as INT(-LOG10(xu-xl) + e LOG10(b)), where e = MAX(EXPONENT(xl),EXPONENT(xu)) and b=RADIX(X).

Note: three interval functions, MAG, MIG, and ABS, correspond to the point intrinsic ABS. The specification of ABS is as the range of the absolute value function, consistent with the general principle that the results of interval functions shall contain the ranges of corresponding point intrinsics. Although "MAG(X)" is written |X| in much of the interval literature, it is more natural in various applications to have ABS(X) denote the range of the absolute value function.

If WID(X) is not exact, then its value shall be upwardly rounded.

Note: WID(X) often appears in convergence criteria of the form WID(X) < EPS. The criterion is certain to be satisfied if the computed value WID(X), used in the comparison, is greater than or equal to the exact value.

Optimization of Interval Expressions

In an interval expression, any transformations by the optimizer

that are permissible for REAL's shall also be permissible for INTERVAL's. For example 1/x/y may be replaced by 1/(x*y).

Note: Different code optimizations may yield significantly different interval values, but each will be an enclosure of the range of the expression over the input intervals.

Note: If the program requires the precise form of an interval expression, then either parentheses may be used or optimization may be turned off through compiler options.

==========
==========

Addendum: An alternate proposal of G. William Walster for mixed-mode arithmetic specification

---

Here is what I would prefer.

Mixed Mode Operations and Conversions

Explicit conversion to interval is performed with the conversion function INTERVAL as follows:

```
Z = INTERVAL(R[,S][,KIND=<<kind>>][,EPSILON=<<fuzz>>])
```

where Z is an INTERVAL, and R and S are INTEGER or REAL or COMPLEX. If KIND=<<kind>> is present, then <<kind>> is a valid kind parameter for the INTERVAL data type, and the converted value shall be an INTERVAL of kind <<kind>>. If KIND =<<kind>> is not present, then the result of INTERVAL shall be of default INTERVAL kind. If both R and S are present, then R shall be less than or equal to S. If S is absent, then the conversion shall be as if S were present and equal to R. If <<fuzz>> is present, then <<fuzz>> is INTEGER, REAL, or INTERVAL. If <<fuzz>> is an INTERVAL value, it shall be as if it were a REAL value of corresponding type, and value equal to SUP(<<fuzz>>). If any argument, to INTERVAL is complex, it shall be as if the arguments were real and equal to the real part of the actual value.

In all cases where containment cannot be guaranteed, an error shall be issued. For example, if R or S or both are constants that are not machine representable, the processor shall have the option of returning a containing interval or issuing a compile-time error.

Examples: Z = INTERVAL(1) results in an interval of default kind that contains the value 1.

Z = INTERVAL(0.5) results in an interval of default kind that contains the value 0.5.

If the statement Z = INTERVAL(0.1) is encountered, then either an error is issued or Z must be set to an interval that contains the decimal number 0.1. Similarly, if the statement Z = INTERVAL(RPARAM) is encountered, where RPARAM is a named REAL constant that may not be an exact representation of its corresponding exact decimal expression, then an error must be issued if a containing interval may not be produced.

Z = INTERVAL(R) results in an interval of default kind that contains the real value of the variable R.

If EPSILON=<<fuzz>> is absent, then the lower bound of Z shall be less than or equal to R, and the upper bound of Z shall be greater than or equal to S.

If EPSILON=<<fuzz>> is present and <<fuzz>> is equal to zero, then Z shall be as if <<fuzz>> were absent.

If EPSILON=<<fuzz>> is present and <<fuzz>> is not equal to

zero, then the lower bound of Z shall be less than or equal to R-SUP(ABS(INTERVAL(<<fuzz>>)), and the upper bound of Z shall be greater than or equal to S+SUP(ABS(INTERVAL(<<fuzz>>)), or an error condition must be set.

INTERVAL may also be used to convert between different kinds of INTERVAL: INTERVAL(Z,KIND=<<kind2>>) shall be an INTERVAL with kind number <<kind2>> that contains the value Z, where Z is an INTERVAL value of any available kind. If EPSILON=<<fuzz>> is present, then INTERVAL(Z[,KIND=<<kind2>>],EPSILON=<<fuzz>>) shall be an INTERVAL (of kind <<kind2>> (if KIND=<<kind2>> is present) whose lower bound is less than or equal to INF(Z)-SUP(ABS(INTERVAL(<<fuzz>>)) and whose upper bound is greater than or equal to SUP(Z)+SUP(ABS(INTERVAL(<<fuzz>>)), or a compile-time error message must be generated.

Examples. INTERVAL((<3.141>),EPSILON=(<1e-3>)) will be an INTERVAL of default type whose lower bound is less than or equal to 3.140, and whose upper bound is greater than or equal to 3.142. Otherwise, an error condition must be set.

If X is a default real variable, then INTERVAL(X,EPSILON=1e-3) will be an INTERVAL of default type that contains [X-1e-3,X+1e-3], or an error condition will be set.

INTERVAL(3,EPSILON=0) will, in general, be an interval of default type that contains the number 3. On most processors, INTERVAL(3,EPSILON=0) can equal [3,3].

Mixed mode operations between INTERVAL's of one kind and another are permitted, as well as mixed mode operations between INTERVAL's and INTEGER or REAL type variables and constants. The result of a mixed mode operation between an INTERVAL and an INTEGER or REAL shall be an INTERVAL.

The operations shall be consistent with 7.1.4.2 of X3J3/96-007 (March 4, 1996 version). In a mixed-mode operation between INTERVAL's of different types, the kind type parameter of the result shall be that of the operand with greater decimal precision or is processor dependent if the operands have the same decimal precision. In a mixed-mode operation between an INTEGER and an INTERVAL, the kind type parameter of the result is that of the INTERVAL, but may be increased if needed to increase the sharpness of the resulting interval. In all cases, containment must be guaranteed, or an error condition set.

In a mixed mode operation between an INTERVAL and a REAL, the kind type parameter of the result shall be that of the operand with greater decimal precision or is processor dependent if the operands have the same decimal precision, provided an INTERVAL of such a type is supported on the processor.
If no such INTERVAL type is supported on the processor, then the result shall be an INTERVAL of type corresponding to the highest decimal precision supported on the processor. In all cases, containment must be guaranteed, or an error condition set.

When an implicit conversion occurs according to the above rules, it shall be as if a call to INTERVAL were used, with appropriate KIND=<<kind>> argument and without an EPSILON=<<fuzz>> argument.

Note: Whenever a mode operation cannot be guaranteed to provide a proper enclosure, an error condition must be set. Since REAL variables may represent results of computations and conversions in which error has accumulated, users are encouraged to avoid mixed mode expressions involving REAL variables whose values may be suspect.

Example. If X is an INTERVAL of default type, and R is a REAL variable, then X**2 + 3*X + R shall be equal to X**2 + INTERVAL(3)*X + INTERVAL(R)

X**2 + 0.1*X will either result in a containing interval or an error condition, depending on whether the processor is capable of guaranteeing containment.

Implicit conversion from INTERVAL to REAL and INTERVAL to INTEGER are defined. If the conversion is to REAL, then the converted value shall be equal to MID(X), where X the interval to be converted and where MID is defined below.
If the converted value is to be INTEGER, then the value shall be equal to NINT(MID(X)). Such an implicit conversion from INTERVAL occurs only in an assignment statement. In particular, an INTERVAL value may be assigned to an INTEGER or a REAL.

Example: If X is an INTERVAL and R is a REAL, then
R = X
is identical to R = MID(X), which, in turn,
is identical to R = REAL(X).

Implicit conversion to interval in assignment statements is also allowed. The conversion from INTEGER or REAL to INTERVAL shall be as if a binary operation between the left member and the right member occurred.

Example: If X is an INTERVAL of default type, and
the default INTERVAL type corresponded
to DOUBLE PRECISION real, then
X = 0.3D0
is the same as
X = INTERVAL(0.3D0)

Note: The functions INF and SUP defined below may also be used to convert an INTERVAL to another data type.

Bill Walster