

Subject: Physical or engineering units
From: Van Snyder
Reference: 03-258r1, section 1.2

1 **1 Number**

2 TBD

3 **2 Title**

4 Physical or engineering units

5 **3 Submitted By**

6 J3

7 **4 Status**

8 For consideration.

9 **5 Basic Functionality**

10 Provide for numeric variables and named constants to have physical or engineering units such as length,
11 mass, area, temperature... “Compute” and check units for expressions and assignment at compile
12 time. Verify that units for corresponding actual and dummy arguments are identical at compile time if
13 an explicit interface is available. (A processor could provide for checking units for corresponding actual
14 and dummy arguments at run time, which would be useful when procedures without explicit interface
15 are used, but this should not be required by the standard.) Input and output units. Convert numeric
16 values during formatted, namelist or list-directed input and output if the item has units, a conversion
17 expression is defined, and unit input/output is requested.

18 It may be possible to apply units to derived types, which would be useful for such user-defined types
19 as intervals or extended-precision numbers. This proposal does not include a provision for doing so. It
20 does not appear to be sensible to apply units to objects of character or logical type.

21 **6 Rationale**

22 Incorrect use of physical units is the second-most-common error in scientific or engineering software,
23 coming immediately after mismatching the types of actual and dummy arguments. Explicit interfaces
24 largely solve the second problem, but do nothing directly for the first. (One can use derived types
25 to provide a physical units system, at the expense of redefining assignment and all of the necessary
26 operations and intrinsic functions.) A particularly expensive ($\approx \$3 \times 10^8$) and embarrassing example of
27 an units mistake was the the cause of the loss of the Mars Climate Orbiter. The loss resulted because the
28 NASA contract required small forces, e.g. from attitude-control maneuvers, to be reported in Newton-
29 Seconds, but Lockheed nonetheless reported them in Pound-Seconds. (This was quite inscrutable, as
30 Lockheed had had contracts with JPL for over thirty years, and they’ve *always* specified SI units.)

31 **7 Estimated Impact**

32 In terms of changes to the standard, this is a substantial project, requiring changes in Sections 4, 5,
33 6, 8, 10, and 13. Determining the units of an expression, checking units of subexpressions, and doing
34 conversions automatically, is not tremendously difficult — I’ve already done it in an expression evaluator
35 for an input routine.

1 8 Detailed Specification

2 Define a new UNIT attribute or type parameter (call it what you will) that can be specified for any
3 numeric variable or named constant. Literal constants are unitless except in one case explained below.

4 Four varieties of units are defined: *Atomic* units are not defined in terms of any other units. *Composite*
5 units are defined in terms of atomic units or other composite units. *Value-converting* units convert values
6 according to specified linear conversions. *Abstract units* are provided to specify the relation between
7 units for dummy arguments and function results. They can be either atomic or composite, but not
8 value-converting. Non-abstract units participate in generic resolution; abstract units do not.

9 Multiplication and division operations on units are defined. Exponentiation by an unitless integer
10 constant is defined to be equivalent to repeated multiplication or division.

11 Each unit declaration creates a *unit coercion function* having the same name as the unit, that takes an
12 argument with any unit, and coerces it to have the unit specified by the name of the function. If the
13 unit of the argument and the result are related by a linear conversion expression, value conversion is
14 also implied. There is an intrinsic UNITLESS coercion function.

15 Quantities can be added, subtracted, assigned, or compared by relational operations only if they have
16 equivalent units. Atomic and value-converting units are equivalent by name. Composite units are
17 equivalent by structure. If a dummy argument has a unit that is not abstract, the associated actual
18 argument shall have an equivalent unit.

19 Abstract units allow enforcing a particular relation between the units, without requiring particular units.
20 For example, the SQRT intrinsic function result has abstract units A, and its argument has abstract
21 units A*A. If a dummy argument has an abstract unit, the associated actual argument can have any unit,
22 but if several dummy arguments have abstract units, the units of their corresponding actual arguments
23 shall be related in the same way as the units of the dummy arguments. If a function result has an
24 abstract unit, and that unit is related to units of dummy arguments, the unit of the result of invoking
25 the function is related to the actual arguments in the same way.

26 There is an intrinsic RADIAN unit, and a parallel set of generic intrinsic trigonometric functions that
27 take RADIAN arguments and produce unitless results. All of the remaining intrinsic procedures have
28 arguments with abstract units and results that are unitless (e.g. SELECTED_INT_KIND) or have
29 the same units as their argument (e.g. TINY). Because function results do not participate in generic
30 resolution, it is not possible to have a parallel set of generic intrinsic inverse trigonometric functions that
31 return RADIAN results. It may be useful to provide an intrinsic module that has some public units
32 and procedures, e.g. units TICK and SECOND and a SYSTEM_CLOCK module procedure that has
33 arguments with units TICK, TICK/SECOND and SECOND.

34 When quantities are added or subtracted, the units of the result are the same as the units of the operands.
35 When quantities are multiplied or divided, the units of the result are the units that result from applying
36 the operation to the operands' units. Multiplication or division by an unitless operand produces a result
37 having the same units as the other operand. Exponentiation by an unitless integer constant is defined
38 to be equivalent to multiplication or division. In an exponentiation operation, the exponent shall be
39 unitless. For intrinsic assignment, the units shall be the same.

40 There are two forms of the UNIT statement. One without (*unit-name*) declares and defines units.
41 Units are declared to be atomic by appearing without a definition. Units are declared to be composite
42 or value-converting by having a defining expression that uses literal constants, unitless named constants,
43 multiplication, division, exponentiation by an unitless integer, previously-declared unit names, and, in
44 the case of value-converting units, addition or subtraction. Units are declared to be abstract by having
45 the ABSTRACT attribute in their declarations. A nonatomic abstract unit shall be defined in terms of
46 abstract units.

47 Value-converting units are defined using linear expressions equivalent in form to $U = [a]U'[\pm b]$, where
48 U is thereby declared to be an unit name, U' is required to be a previously declared nonabstract unit
49 name, and $a \neq 0$ and b are unitless numeric initialization expressions. A value-converting unit is defined
50 even if a is absent or one and b is absent or zero. In these expressions, a is considered to have units
51 U/U' and b is considered to have units U , but we can't say that since U isn't defined yet. This is the

1 only case where literal constants are not unitless. This explicitly defines a conversion function from U
 2 to U' and implicitly defines its inverse (always possible because a is required to be nonzero). Although
 3 U' need not be atomic, if it is atomic it remains atomic even though an inverse conversion is defined.
 4 Value-converting functions are generic, since there may be several implicitly-defined inverse conversions.
 5 Neither U nor U' shall be abstract.

6 Value conversions are transitive. Since each unit can only be declared once, and cannot be referenced
 7 before being declared, an explicitly circular dependence of conversions is impossible. There will usually
 8 be an atomic unit involved in one of the value conversions, but a composite unit is possible. There
 9 are examples of transitive conversions, multiple implicit conversions, and conversions that depend on
 10 nonatomic units below.

11 Value conversion is implied during input or explicit units conversion using the units conversion function
 12 implied by the unit declaration or its inverse, provided the argument or input value is related to the
 13 result or input list item by a sequence of explicit or implicit conversions. Value conversion does not
 14 occur during intrinsic assignment (this could be an extension), argument association or output. When
 15 conversion is applied, all constants within a and b are converted to the kind of the argument expression
 16 or input list item. This specification could interact with the proposal to include function result types in
 17 the criteria for generic resolution.

The Mars Climate Orbiter crashed because quantities with the wrong units were written by one program, and assumed to have the correct units by another program. Automatic units checking and value conversion would have let Lockheed use whatever units they wanted to use, so long as the JPL software had the unit name, and the appropriate conversion, available. If JPL software insisted on units, and the Lockheed data were unitless, or used units that JPL software did not specify, the error would have been detected.

18
 19 In all other unit-defining expressions, the only constants allowed are 1 and 1.0, or named constants
 20 having those values.

21 Variables or named constants are declared to have units by specifying `UNIT(unit-name)` in their decla-
 22 rations, `*unit-name` after their names in declaration statements, or by an `UNIT(unit-name)` statement.
 23 `UNIT(unit-name)` is allowed in the *prefix* of a *function-stmt*.

24 There is an optional `U[w]` suffix to numeric format descriptors, that causes units to be output by
 25 write statements, or input and checked by read statements. The text of the unit that is output, or
 26 checked during input, is the same as the unit name, except that case of letters is not significant. There
 27 is a specification in OPEN, READ and WRITE statements that controls whether units for numeric
 28 quantities that have units are to be output (checked) by namelist or list-directed output (input). There
 29 is a specification for the INQUIRE statement to inquire whether this mode is set by an OPEN statement
 30 for a connection. A single specification, rather than separate ones for namelist and list-directed transfers,
 31 is adequate.

32 Some thought and debate will be necessary to decide what to do about input and output of arrays that
 33 have units. Should the value in the input for every element be required to specify its units, or is it
 34 enough that at least one does? If one is enough should it be specified to be first (or last)? Should units
 35 be provided for every element of output, or is one enough? There is not a problem for array constructors,
 36 since they can be wrapped with an units coercion or conversion function.

37 8.1 Examples

38 8.1.1 Atomic and composite units

```
39  UNIT :: INCH, SECOND
40  UNIT :: CM, INCH_TO_CM = CM / INCH
41  REAL, PARAMETER, UNIT(INCH_TO_CM) :: CONVERT = INCH_TO_CM(2.54)
42  UNIT :: SQINCH = INCH * INCH ! or INCH ** 2
43  UNIT :: IPS = INCH / SECOND, Hz = 1 / SECOND ! or SECOND ** (-1)
44  REAL, UNIT(SQINCH) :: A
45  REAL, UNIT(Hz) :: F
```

```

1  REAL, UNIT(INCH) :: L, L2, C*CM
2  REAL, UNIT(SECOND) :: T
3  REAL :: V
4  UNIT(IPS) :: V
5
6  V = A + L                ! INVALID -- SQINCH cannot be added to INCH,
7                          ! and neither one can be assigned to IPS
8  V = IPS(A + SQINCH(L))  ! VALID -- I'm screwing this up intentionally
9  V = (A / L + L2) / T    ! VALID -- IPS is compatible with INCH / SECOND
10 A = L * L2              ! VALID -- SQINCH is compatible with INCH * INCH
11 F = V / L               ! VALID -- units of LHS and RHS are both 1/SECOND
12 C = CONVERT * L        ! VALID -- CM / INCH * INCH = CM
13 L = SQRT(A) * 5.0e-3    ! VALID -- exercise for reader

```

14 8.1.2 Abstract units

```

15  INTERFACE
16    REAL UNIT(UR) FUNCTION CBRT ( A )
17      UNIT, ABSTRACT :: UR, UA = UR**3
18      REAL, UNIT(UA) :: A
19    END FUNCTION CBRT
20  END INTERFACE

```

21 8.1.3 Value-converting units

```

22  UNIT :: MHz = 1.0e6 * Hz, GHz = 1000 * MHz, KHz = 0.001 * MHz
23  UNIT :: F, C = ( F - 32 ) * 5.0 / 9.0 ! Also defines (atomic) F = 1.8 * C + 32.0
24  UNIT :: DEGREES = 45.0 * RADIANT / ATAN(1.0)
25  REAL, UNIT(Radian) :: Angle = Degrees(45) ! = Radian(0.785398163)
26  REAL, UNIT(MHz) :: Frq = MHz(3310.0)
27  REAL, UNIT(C) :: Temp = F(212.0)          ! = C(100.0)
28  ...
29  CHARACTER(32) :: LINE
30  WRITE(LINE,*,UNITS="yes") GHz(Frq), F(Temp), Degrees(Angle)
31  READ(LINE,*,UNITS="yes") Frq, Temp, Angle

```

32 Execution of the WRITE statement causes LINE to have the value "3.31 GHz 212.0 F 45.0 Degrees"
 33 (approximately). Execution of the READ statement causes the variables Frq, Temp and Angle to get their
 34 original values (approximately).

35 Notice that MHz(3.31), C(212) and Radian(45) have values 3.31, 212 and 45 respectively, with units
 36 MHz, C and Radian respectively, not 3310.0, 100.0 and 0.785398163 respectively, since generic resolution
 37 selects the simple unit-coercion functions, not the conversion ones, for unitless arguments.

38 Input of the form "3.3021148036e-10 Second 212 F 45 Degrees" is not permitted, since the decla-
 39 ration of Hz does not define a value-converting unit.

40 If we have

```
41  REAL(kind(0.0d0)), UNIT(Radian) :: AngleD
```

42 then in Degrees(AngleD) the 1.0 that is the argument of ATAN in the definition of the Degrees conversion
 43 function is converted to 1.0d0.

44 9 History