Reference number of working document: **ISO/IEC JTC1/SC22/WG5 N2113**

Date: 2016-7-7

Reference number of document: **ISO/IEC TS 99999:2016(E)**

Committee identification: ISO/IEC JTC1/SC22

Secretariat: ANSI

Information Technology — Programming languages — Fortran — Units of measure for numerical quantities

Technologies de l'information — Langages de programmation — Fortran — Unités de mesure des quantités numéric

Contents

Foreword

1 ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

2 International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

3 The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75% of the member bodies casting a vote.

4 In other circumstances, particularly when there is an urgent market requirement for such documents, the joint technical committee may decide to publish an ISO/IEC Technical Specification (ISO/IEC TS), which represents an agreement between the members of the joint technical committee and is accepted for publication if it is approved by 2/3 of the members of the committee casting a vote.

5 An ISO/IEC TS is reviewed after three years in order to decide whether it will be confirmed for a further three years, revised to become an International Standard, or withdrawn. If the ISO/IEC TS is confirmed, it is reviewed again after a further three years, at which time it must either be transformed into an International Standard or withdrawn.

6 Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

7 ISO/IEC TS 99999:2016(E) was prepared by Joint Technical Committee ISO/IEC/JTC1, *Information technology*, Subcommittee SC22, *Programming languages, their environments and system software interfaces*, Working Group WG5, Fortran.

8 This technical specification specifies an extension to the computational facilities of the programming language Fortran. Fortran is specified by the International Standard ISO/IEC 1539-1:2010(E).

9 This technical specification is non-normative. Some of the functionality described by this Technical Specification may be considered for standardization in a future revision of ISO/IEC 1539, but it is not currently part of any Fortran standard. Some of the functionality in this Technical Specification may never be standardized, and other functionality may be standardized in a substantially changed form.

10 The goal of this technical specification is to build widespread existing practice for units of measure. It gives advice on extensions to those vendors who wish to provide them.

Introduction

The problem to be solved

1 The most common errors in scientific and engineering software, that a language and its processor might be expected to ameliorate, are mismatched, missing or excess actual arguments in procedure references, followed by out-of-bounds array references, and then incorrect use of physical units of measure. Explicit interfaces largely solve the first problem and help to avoid the second problem, but do nothing directly for the third. In other disciplines, data entities might have units of measure that are unrelated to what are customarily considered to be physical or engineering units of measure, or related to physical or engineering units of measure by operations other than multiplication or division. For example, in acoustics, units of measure such as decibels might be useful, while in finance, units of measure such as dollars or pound sterling might be useful.

An example of failure

1 A particularly embarrassing mistake involving units of measure caused the expensive ($\approx \$3 \times 10^8$) loss of NASA's Mars Climate Orbiter. The ultimate cause of the loss was that Lockheed reported impulses from attitude-control maneuvers in Imperial units (Pound-Seconds) even though the NASA contract required them to be reported in SI units (Newton-Seconds). If the facilities proposed in this technical specification had been available, and used correctly, Lockheed's mistake would either have been caught or corrected, entirely automatically.

Shortcomings of methods to simulate units facilities

1 Although it is possible to use type parameters of derived types to provide a units system for numerical quantities, it comes at the expense of redefining assignment and all of the necessary operations and intrinsic functions, which increases both development and maintenance cost. Schemes that have been proposed used type parameters, or integer components, to represent exponents of fundamental dimensions. The amount of work required for even a few units is enormous if kind type parameters are used, since it must be done for all combinations of intrinsic kinds of real numbers, and all combinations of dimensions. Assume two kinds of real numbers and the following ranges for exponents for the fundamental SI units or dimensions.

Fundamental SI unit	Fundamental SI dimension	Exponent Range	Number of exponents
length	meter	$-3 \cdots +3$	7
time	second	$-3 \cdots +1$	5
mass	kilogram	$-1 \cdots +1$	3
thermodynamic temperature	Kelvin	$-1 \cdots +1$	3
electric current	Ampere	$-1 \cdots +1$	3
quantity of a substance	mole	$-1 \cdots +1$	3
luminous intensity	candela	$-1 \cdots +1$	3
angle[1]	radian	$-2 \cdots +2$	5

2 Considering at first only identity, negation, addition, subtraction, and comparison, for one kind of REAL, one needs to write $10 \times (7 \times 5 \times 3^5 \times 5) = 425,250$ procedures. Add another 18 if you want square root, for arguments with all-even exponents. For multiplication and division one needs to write an additional $2 \times \prod_i f(a_i, b_i)$ procedures, where $f(a,b) = (b-a+1)^2 + a(1-a)/2 - b(1+b)/2$ and a_i, b_i are the exponent bounds in the table above, or $425,351,556$ procedures, assuming one wants not to produce any exponents outside the above ranges. Altogether $425,776,806$ procedures, for each

[1] Not an SI unit

REAL kind. This gives complete compile-time checking of the exponents of ISO fundamental units and angle, except in expressions involving exponentiation. An additional type with a kind type parameter could be used to represent exponents. This would allow exponentiation, at a cost of 45,525 additional procedures, assuming again that one does not wish to produce results outside the ranges in the above table. If additional exponent values are needed, the number increases even further. Further, this does not accommodate the possibility of non-physical dimensions such as currency or safety rates, or dimensions that are not described using exponents of fundamental dimensions, such as decibels or parts per million. Doing so would require more kind type parameters, and more procedures.

3 If components or length type parameters are used to represent exponents, error detection is deferred to run time, and has an execution cost. It might have the further effect of preventing optimization because the operation definitions are hiding inside procedures. It is possible to incorporate automatic units checking and conversion during input into this or similar schemes, but only at the cost of defining input/output procedures for each unit or unit expression.

4 More importantly, a units system for numerical quantities based upon integers can distinguish units (e.g., length from time) but not dimensions or measures (e.g., pounds from kilograms). Providing for this using derived types requires scale and offset components, imposing execution costs of both time and memory, especially for arrays unless arrays are moved into components of derived-type objects that represent quantities with units, and separate procedures are written for each rank. This prevents using Fortran's powerful array processing facilities.

5 A system based upon using a different type for each measure, say meter, kilometer, foot, acre, hectare, celsius, kelvin, fahrenheit, ampere, coulomb, ..., would require a number of types and procedures dependent upon the problem at hand, or an enormous library module. It could be equipped with conversion procedures.

6 The first and third methods described above provide some compile-checking, but cannot support abstract units, that is, units of dummy arguments and function results that specify a relationship between units of corresponding actual arguments, without requiring identical units. This is important for "library" procedures, the simplest example being square root.

7 If units of measure are integrated into the declaration of objects of real type – variables, structure components, and named constants – it is not necessary for a program to redefine operations, the use of units within expressions and procedure references can be checked without subverting optimization or imposing a runtime penalty, and units can be checked and converted during input, and reported during output.

History of units of measure in programming languages

1 A facility to accomplish what is described here was proposed in about 1978 for inclusion in what became Ada, in about 1986 for inclusion in what became Fortran 90, in 1997 for inclusion in Fortran 2003, and in 2004 for inclusion in Fortran 2008. Despite its obvious utility to aid in correct construction of programs for scientific or engineering calculations, the facility has not been provided in any major programming language. Since Fortran is the premier vehicle for such calculations, it is reasonable that Fortran should introduce this idea to the software engineering community.

What this report proposes

1 This technical specification extends the computational and input/output facilities of Fortran, to provide methods for program developers optionally to specify units of measure for data entities of real type.

Information technology – Programming languages – Fortran

Technical Specification: Units of Measure for Numerical Quantities in Fortran

1 Scope

1 This technical specification specifies an extension to the computational and input/output facilities of the programming language Fortran. The Fortran language is specified by International Standard ISO/IEC 1539-1:2010(E) : Fortran. The extension allows program authors to specify units of measure for data entities of real type.

2 The facility described might more properly be termed *dimensions*, because the fundamental SI units (length, mass, time, thermodynamic temperature, electric charge, amount of substance, and luminous intensity) are not the foundation for this technical specification. The term DIMENSION, however, already has a meaning in Fortran that applies to numerical quantities. The term UNIT already also has a meaning in Fortran, but is not an attribute of numerical quantities. Therefore the term UNIT is used. The fundamental SI dimensions (meter, kilogram, second, Kelvin, Ampere, mole, and candela) are not required by this technical report.

3 Specifications are provided for how those units of measure are declared, how they are combined in arithmetic expressions, and the rules pertaining to them during assignment, procedure invocation, and input/output.

4 Clause 5 of this technical specification contains a general and informal but precise description of the extended functionalities. Clause 6 identifies potential extensions that are not requested by this technical specification. Clause 7 contains detailed instructions for editorial changes to ISO/IEC 1539-1:2010(E).

2 Normative References

1 The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2 ISO/IEC 1539-1:2010(E) : *Information technology – Programming languages – Fortran; Part 1: Base Language*

3 Terms and definitions

1 For the purposes of this document, the terms and definitions given in ISO/IEC 1539-1:2010(E) and the following, apply.

3.1

1 **unit**

- a source or destination of data transfer during input/output
- measure of a quantity

3.2

1 **unit family**

set of units that can be related by sequences of conversions

4 Compatibility

1 The facility specified by this Technical Specification is compatible with the computational facilities of Fortran as standardized by ISO/IEC 1539-1:2010(E).

5 Requirements

5.1 General

1 The subclauses in this clause contain a general and informal, but precise, description of the extensions to the syntax and semantics of the Fortran programming language to provide units of measure for data entities of real type. The system shall

- allow to define a system of units upon an arbitrary foundation,
- check units in expressions, assignment, and procedure references at compile time,
- distinguish different measures of the same fundamental quantity, and provide for explicit conversion between them,
- not increase execution time, except where conversion is explicitly requested,
- not require additional storage,
- allow to output units, and to check and convert units during input,
- require minimal labor for its use, and
- support abstract units, which specify the relationship of units of dummy arguments and function results, and thereby the relationship of corresponding actual arguments and function results, without requiring specific units.

5.2 Summary

5.2.1 General

1 This technical specification defines a new UNIT attribute for data entities of real type.

2 The UNIT attribute specifies a unit of measure for a variable, structure component, or named constant of real type. Numerical objects of integer or complex type are unitless.

NOTE 5.1

It might be useful to allow units for objects of integer and complex type. If so, it might be necessary to allow different units of measure for the real and imaginary parts of complex objects. Pondering the value of this extension should be postponed until experience with the present proposal in real applications provides guidance.

5.2.2 UNIT definition

1 This technical specification provides a new UNIT definition statement to specify the name of a unit of measure for a data entity of real type, to define whether it is abstract, and to define whether it is atomic or related to other units, and if so how it is related.

5.2.3 UNIT attribute declaration

1 This technical specification provides a new UNIT attribute specification to declare the unit of measure for a variable, structure component, or named constant of real type.

2 This technical specification provides a new UNIT declaration statement to declare the unit of measure for a variable or named constant of real type.

5.2.4 Units in expressions

1 This technical specification specifies how units are required to conform, and how units are composed, within expressions.

5.2.5 Units in assignments

1 This technical specification specifies how units are required to conform in intrinsic assignment and pointer assignment.

5.2.6 Units of associate names

1 In an ASSOCIATE or SELECT TYPE construct the units of the *associate-name* are the same as the units of the *selector*.

5.2.7 Units during formatted input/output

1 This technical specification provides a new "U[w][.s]" suffix for D, E, EN, ES, F and G format descriptors, to specify the width for names of units during input and output, and the number of blank spaces between the value and the unit name.

5.2.8 Units of dummy and actual arguments

1 This technical specification provides abstract units, which can be used to specify how units of dummy arguments of a procedure conform to each other, and thereby how units of corresponding actual arguments are required to conform to each other, without requiring specific units for the actual arguments.

2 This technical specification provides a facility to describe how units of actual arguments are required to conform to nonabstract units of dummy arguments.

5.2.9 Units of function results

1 This technical specification provides a facility either to specify units of function result variables, or to describe how units of function result variables are related to units of dummy arguments, and thereby how units of function results are related to units of function actual arguments.

5.2.10 Units and generic resolution

1 This technical specification specifies that nonabstract units are used for generic resolution.

5.2.11 Intrinsic units and intrinsic functions related to units

1 This technical specification defines intrinsic units RADIAN and UNITLESS. Objects of real type for which units are not specified are assumed to have units of measure UNITLESS.

2 This technical specification defines a generic intrinsic function UNITLESS, which has an argument of real type. The result value is the same as the argument value, and the units of the result are UNITLESS.

3 This technical specification defines a generic intrinsic function RATIONAL_POWER, used to raise a data entity of real type to a constant rational power, and to specify the units of the function result.

5.2.12 Scope and class of names of units

1 Names of units are local identifiers of class (1) as defined in subclause 16.3.1 of ISO/IEC 1539-1. They can be accessed by host and use association.

5.3 UNIT definition statement

5.3.1 Semantics of the UNIT definition statement

1 The UNIT definition statement defines the name of a unit, what category of unit it is, if and how it is related to other units, whether addition and subtraction of objects with those units are prohibited, and functions to convert, coerce, and confirm units. Each unit defined by a UNIT definition statement is a different unit, even if it has the same name as a unit defined in a different scoping unit.

5.3.2 Categories of units

5.3.2.1 General

1 A unit is an atomic unit, a composite unit, a conversion unit, or an abstract unit.

5.3.2.2 Atomic units

1 An atomic unit is one that is not defined in terms of other units.

5.3.2.3 Composite units

1 A composite unit is one that is defined by an expression using multiplication of units, division of units, or exponentiation of units by an integer or a ratio of integers.

5.3.2.4 Conversion units

1 A conversion unit is one that is defined in terms of one other unit by a linear transformation having constant numeric coefficients.

5.3.2.5 Abstract units

1 Abstract units specify the relationship between units of dummy arguments, or the dependence of function result variable units upon dummy argument units, and thereby the relationship between units of corresponding actual arguments, or between actual arguments and function results, without requiring specific units for actual arguments.

2 Abstract units are explicitly declared to be abstract. They can be atomic, or they can be composite provided they are defined only in terms of abstract units.

5.3.3 Syntax of UNIT definition statement

R501 *unit-definition-stmt* **is** UNIT [[, *unit-attr-list*] ::] *unit-definition-list*

R502 *unit-attr* **is** ABSTRACT
or EXCLUDE_ARITHMETIC
or *access-spec*

R503 *unit-definition* **is** *unit-name* [= *unit-expr*]

C501 (R501) The double colon shall appear if *unit-expr* appears in any *unit-definition* in the statement.

C502 A *unit-attr* shall not be specified more than once for a unit name, no matter how specified.

NOTE 5.2

If *unit-name* is UNITLESS or RADIAN, the intrinsic unit, unit coercion function, and unit confirmation function with that name are not accessible.

R504 *unit-expr* **is** *unit-conversion-expr*
or *unit-composition-expr*

R505 *unit-conversion-expr* **is** *mult-operand* * *unit-name* [*mult-op mult-operand*] ■
■ [*add-op add-operand*]
or [*mult-operand* *] *unit-name mult-op mult-operand* ■
■ [*add-op add-operand*]
or *unit-name add-op add-operand*
or [*mult-operand* *] (*unit-conversion-expr*) ■
■ [*mult-op mult-operand*] [*add-op add-operand*]

C503 (R505) The *unit-name* shall be defined previously within the same scoping unit, or accessible by use or host association. It shall not be the name of a conversion unit or abstract unit, or specify the intrinsic unit UNITLESS.

C504 (R505) Each *add-operand* and *mult-operand* shall be a unitless constant expression of real or integer type.

C505 (R505) The *unit-name* shall not have the EXCLUDE_ARITHMETIC attribute.

C506 (R505) After substitution of values for named constants, and algebraic simplification, *unit-conversion-expr* is always of the form $a \times$*unit-name*$+b$ where a and b are unitless numeric constants. If no *mult-operand* appears the value of a is 1. If no *add-operand* appears, the value of b is zero. The value of a shall not be zero.

NOTE 5.3

If $a = 1$ and $b = 0$, a unit synonym is effectively created. The units are related by conversion, so they can be used interchangeably in input. See 5.10.

R506 *unit-composition-expr* **is** *unit-factor* [*mult-op unit-composition-expr*]

R507 *unit-factor* **is** *unit-factor* ** *int-literal-constant*
or *unit-factor* ** (*signed-int-literal-constant*)
or RATIONAL_POWER (*unit-factor*, ■
■ *signed-int-literal-constant*, *int-literal-constant*)
or *unit-name*
or (*unit-composition-expr*)

C507 (R506) Either *mult-op* or *int-literal-constant* shall appear.

C508 (R507) The *unit-name* shall be defined previously within the same scoping unit, or accessible by use or host association.

C509 (R507) The *unit-name* shall be the name of an abstract unit if and only if the unit name being defined by the UNIT statement is an abstract unit.

C510 (R507) If *unit-name* has the EXCLUDE_ARITHMETIC attribute, the unit being defined shall have the EXCLUDE_ARITHMETIC attribute.

C511 (R507) An *int-literal-constant* shall not be zero.

1 If *unit-factor* is RATIONAL_POWER(u,n,d), it specifies a factor of the form $u^{\frac{n}{d}}$, where $\frac{n}{d}$ is a rational fraction, not Fortran integer division.

2 A unit name is abstract if and only if ABSTRACT appears in the *unit-definition-stmt* that defines it, or the ABSTRACT attribute is specified for it by an ABSTRACT statement.

5.3.4 Excluded operations

1 It does not make sense to perform arithmetic operations such as addition, or multiplication by unitless quantities, for objects with certain units, for example, decibels. Addition, subtraction, and multiplication or division by an unitless quantity, are prohibited for objects for which the EXCLUDE_ARITHMETIC *unit-attr* is specified.

5.3.5 Equivalence of units

1 The atomic form of an atomic unit is the unit name. The atomic form of a conversion unit is its unit name.

2 An atomic unit is not equivalent to any other unit. A conversion unit is equivalent only to its accessible synonyms.

NOTE 5.4

A unit might have different names in different scoping units.

3 The atomic form of a composite unit is produced by replacing each composite unit in its defining expression by that unit's atomic form. If a unit appears more than once the several appearances are combined into one by adding their exponents, in the usual manner of algebraic simplification. This results in an algebraic expression of the form $\prod_{i=1}^{n} a_i^{e_i}$, where n is the number of atomic units in the form, each a_i is an atomic unit or a conversion unit, no two units in the expression are equivalent, and each e_i is either an integer or the ratio of two integers that have no common factors.

4 This process terminates because the units that appear in a unit definition expression are required to be previously defined or accessible by use or host association.

C512 In the atomic form of a composite unit, the value of e_i shall not be zero.

C513 In the atomic form of a composite unit, the sum of the exponents of *unit-names* in any single unit family (5.3.6) shall not be zero.

NOTE 5.5

Prohibiting the sum of exponents of conversion units in the same family from being zero prevents such absurdities as CENTIMETER/INCH, which is the unitless number 2.54. This would make a composite unit a conversion unit.

NOTE 5.6

The processor cancels common factors to produce e_i, to make it easier to compare atomic forms.

5 Although each unit definition defines a different unit, composite units can be equivalent. Composite units, whether abstract or not, are equivalent if and only if their atomic forms are equivalent. If the atomic form of one is $\prod_{i=1}^{n} a_i^{e_i}$ and the other is $\prod_{j=1}^{m} b_j^{e_j}$ then the units are equivalent if and only if $m = n$, there is a one-to-one correspondence between a_i and b_j such that a_i and b_j are equivalent, and $e_i = e_j$ where a_i and b_j are equivalent. The unit names a_i and b_j are equivalent if they refer to the same unit definition, even if their names in a scoping unit are different as a result of renaming during use association.

5.3.6 Unit families

1 Among every set of conversion units that are related by *unit-definition*s in which every *unit-expr* is a *unit-conversion-expr*, there is exactly one unit that is not a conversion unit. Let that unit be called Z.

2 There is a sequence of conversions between units W ... Z if there is a set of definitions of conversion units W = f_{WX}(X), X = f_{XY}(Y), ... Y = f_{YZ}(Z), where each f is a mathematical function defined by the *unit-conversion-expr*. Denote the composition of these definitions by W = f_{WZ}(Z). The conversion relation between w and z, where w and z are values with units W and Z, respectively, is $w = f_{WX}(f_{XY}(\ldots f_{YZ}(z)\ldots)) = f_{WZ}(z)$. Since each *unit-conversion-expr* is linear and the *mult-operand* is nonzero, there are also inverse conversion relations Z = f_{ZY}(Y) = f^{-1}_{YZ}(Y), ... X = f_{XW}(W) = f^{-1}_{WX}(W), and a sequence of conversions $z = f_{YZ}(\ldots f_{YX}(f_{XW}(w))\ldots) = f_{ZW}(w) = f^{-1}_{WZ}(w)$. These compositions are always linear.

3 If there is a unit A such that there is a sequence of conversions f_{AZ} between Z and A but no direct sequence of conversions f_{AW} between W and A, there is nonetheless a sequence of conversions between w and a where a has units A defined by $a = f_{AZ}(f_{ZW}(w))$. If A and W both depend upon some intermediate unit I $\neq$ Z, this sequence can be simplified since within it there is the identity conversion consisting of the sequence $f_{IZ}(f_{ZI}(z)) = f_{IZ}(f^{-1}_{IZ}(z)) = z$

4 Units that can be related by sequences of conversions constitute a unit family (3).

5 The atomic form of a composite unit can be represented by a tree in which every internal vertex represents an operator and every terminal vertex represents a unit or a constant, and the relationship of internal vertices is consistent with the hierarchy of operator precedences established by the *unit-composition-expr*s that define the unit and the composite units in its *unit-composition-expr*. A tree that is isomorphic can be constructed by exchanging the subtrees of multiplication operators because multiplication of units is commutative.

6 If all conversion units in the tree that represents the atomic form of one composite unit can be put into one-to-one correspondence with all the conversion units in a tree that is isomorphic to the one that represents the atomic form of another composite unit, such that the units in every such pair of corresponding units are in the same unit family, then there is a conversion between the composite units, which is constructed by applying the sequence of conversions between members of each corresponding pair, and the composite units are in the same unit family.

NOTE 5.7

Consider the following definitions:

```
UNIT :: FOOT, SECOND
UNIT :: MILE = FOOT/5280.0, HOUR = SECOND/3600.0
! 5280.0 has units FOOT/MILE, 3600.0 has units SECOND/HOUR
UNIT :: FPS = FOOT/SECOND
UNIT :: MPH = MILE/HOUR
```

The units FPS and MPH are in the same family because `FOOT` is in the same family as `MILE` and `SECOND` is in the same family as `HOUR`.

5.3.7 Unit conversion functions

1 A unit family defines a set of pure elemental generic unit conversion functions. For each unit, there is a generic function having the same name as the local name of the unit, with specific functions having a real argument with every kind supported by the processor, and every unit in the family. The result of each function has real type of the same kind as its dummy argument, and units specified by its generic name. Each specific function implements a conversion or sequence of conversions between a value having

units of its argument and a value having units of its result variable.

NOTE 5.8

Consider the following unit definitions:

```
UNIT :: KELVIN
UNIT :: CELSIUS = KELVIN - 273.15
UNIT :: FAHRENHEIT = 1.8 * CELSIUS + 32.0
UNIT :: RANKINE = 1.8 * KELVIN
```

CELSIUS, FAHRENHEIT and RANKINE are conversion units. Those units, together with KELVIN, constitute a unit family. The definition of CELSIUS defines a generic function named CELSIUS that converts values with units KELVIN to values with units CELSIUS. It also defines specific functions for the generic KELVIN that convert values having CELSIUS units to values having KELVIN units. The definition of FAHRENHEIT defines a generic function that converts values with units KELVIN or CELSIUS to values with units FAHRENHEIT, and specific functions for KELVIN and CELSIUS that convert values with units FAHRENHEIT to values with units KELVIN and CELSIUS, respectively. It is always possible to construct the inverse conversions because the *mult-operand* in a *unit-conversion-expr* cannot be zero. The Celsius to Fahrenheit conversion function is defined directly by the conversion expression in the definition of FAHRENHEIT; the Fahrenheit to Celsius conversion is defined indirectly by the inverse of that relation. The Kelvin to Fahrenheit conversion function is defined by applying the Kelvin to Celsius conversion and then the Celsius to Fahrenheit conversion. It is recommended that the processor algebraically simplify the function compositions. The usual round-off and truncation considerations apply, and the results might not be identical to an analytic composition.

Although there is no direct conversion relation between RANKINE and FAHRENHEIT, this nonetheless defines the same conversion as would be defined by

```
UNIT :: RANKINE = FAHRENHEIT + 459.67
```

by the sequences of conversions FAHRENHEIT↔CELSIUS↔KELVIN↔RANKINE.

5.3.8 Unit coercion and confirmation functions

1 Definition of a unit defines two pure elemental specific functions in the generic interface having the same name as the unit, for each kind of real supported by the processor. Each has a dummy argument of real type. The result value of each is of real type, has the same kind and value as the dummy argument, and units that are the same as the function name.

2 The first is a units coercion function. Its dummy argument shall have UNITLESS units.

3 The second is a units confirmation function. Its dummy argument shall have units equivalent to the units of its result.

NOTE 5.9

It is possible to coerce a value from one unit to another even if the units are not in the same family by first removing the units from one of them by using the UNITLESS function (5.16) and then applying the units coercion function for the other. The appearance of the UNITLESS function is a signal that this might be a dangerous coercion.

5.4 Unit name attribute specification statements

5.4.1 ABSTRACT statement

1 The ABSTRACT statement specifies the ABSTRACT attribute for a unit name.

R508 *abstract-stmt* **is** ABSTRACT [::] *unit-name-list*

C514 (R508) Each *unit-name* shall be the name of a unit that is defined in the same scoping unit.

5.4.2 EXCLUDE ARITHMETIC statement

1 The EXCLUDE ARITHMETIC statement specifies the EXCLUDE_ARITHMETIC attribute for the specified units.

R509 *exclude-arithmetic-stmt* **is** EXCLUDE ARITHMETIC [::] *unit-name-list*

C515 (R509) Each *unit-name* shall be the name of a unit that is defined in the same scoping unit.

5.4.3 Unit names in accessibility statements

1 An accessibility statement may specify the accessibility attribute for a unit name.

R525 *access-id* **is** *use-name*
or *generic-spec*
or *unit-name*

C516 (R525) The *unit-name* shall be the name of a unit.

5.5 Unit attribute specification

1 A *unit-attr-spec* may appear in a *component-attr-spec-list* in a *data-component-def-stmt* or an *attr-spec-list* in a *type-declaration-stmt*. It specifies the units for a variable, structure component, or named constant of real type. If a unit attribute is not specified for a variable, structure component, or named constant of real type, the units of that entity are the intrinsic unit UNITLESS.

R510 *unit-attr-spec* **is** UNIT (*unit-name*)

C517 (R510) The *unit-name* shall be the name of a unit that is defined previously within the same scoping unit, or accessible by use or host association.

C518 (R510) The *unit-attr-spec* shall not be specified for an entity that is not a variable, structure component, named constant, or external function, or is not of real type.

C519 (R510) If *unit-name* is abstract and composite, all abstract units in its atomic form shall be abstract units of nonoptional dummy arguments of the same procedure as for the entity being declared.

Unresolved Technical Issue Nonoptional

> Alternatively, if an actual argument corresponds to a dummy argument that has an abstract composite unit, then every atomic unit in the atomic form of that composite unit shall be the unit of a dummy argument to which an actual argument corresponds.

C520 (R510) If *unit-name* is abstract, atomic, and is the unit of a function result, that unit shall appear in the atomic form of the unit of a dummy argument of that function.

NOTE 5.10

It's not obvious that atomic abstract function result variable units can be guaranteed to be useful and consistent unless some dummy argument has the same atomic abstract unit. If they can be useful and consistent, it might still be the case that the explanation is too difficult.

5.6 UNIT statement

1 A unit specification statement specifies the units for a list of variables, named constants, or external functions of real type.

R511 *unit-stmt* **is** UNIT (*unit-name*) [::] *entity-name-list*

C521 (R511) The *entity-name* shall be the name of a variable, named constant, or external function, and shall be of real type.

5.7 Units in array constructors

1 Within an *array-constructor* of real type, every *expr* shall have equivalent units. See 5.3.5.

5.8 Units in expressions

1 The form of expressions is defined in Subclause 7.1.2 of ISO/IEC 1539-1:2010(E).

2 If a *mult-operand* includes a *power-op* the units of the second *mult-operand* of the *power-op* shall be UNITLESS. If the units of the *level-1-expr* of the *power-op* are UNITLESS the units of the *mult-operand* are UNITLESS. Otherwise the second *mult-operand* shall be a constant of type integer, and the units expression of the *mult-operand* is formed by multiplying the exponents of every factor of the units expression of the *level-1-expr* by the value of the second *mult-operand* of the *power-op*.

NOTE 5.11

```
unit :: A, B, SQRTA=RATIONAL_POWER(A,1,2) ! A**(1/2)
real, unit(sqrta) :: X
real, unit(B) :: Y
real, unit(A) :: Z
X**Y    ! prohibited
X**2    ! has units SQRTA**2, which is equivalent to A
X**3    ! has units SQRTA**3, which is equivalent to A**(3/2)
```

3 If an *add-operand* includes a *mult-op* that is * the units expression of the *add-operand* is formed by multiplying the units expression of the first *add-operand* of the *mult-op* by the units expression of the second *mult-operand*. If one of the operands has UNITLESS units the units of the result are the units of the other operand.

4 If an *add-operand* includes a *mult-op* that is / the units expression of the *add-operand* is formed by dividing the units expression of the first *add-operand* of the *mult-op* by the units expression of the second *mult-operand*. If the second operand has UNITLESS units the units of the result are the units of the first operand. If the first operand has UNITLESS units the units of the result are the inverse of the units of the second operand.

NOTE 5.12

```
X*Y     ! has units SQRTA*B
7*X     ! has units SQRTA
```

5 If a *level-2-expr* consists of *add-operand* or *add-op* *add-operand*, the units of the *level-2-expr* are the units of the *add-operand*.

6 If a *level-2-expr* includes an *add-op* with two operands and if both operands of the *add-op* are of real type, the operands shall have equivalent units (5.3.5) and neither unit shall have the EXCLUDE_ARITHMETIC attribute. If only one operand is of real type, it shall have UNITLESS units. The units of the result of the operation are the units of the operand of real type.

7 If a *level-2-expr* includes a *mult-op* and the unit of one of the operands has the EXCLUDE_ARITHMETIC attribute, the other operand shall not be unitless.

NOTE 5.13

```
X + Y              ! prohibited -- units of X and Y are not equivalent
Z + A(UNITLESS(Y)) ! has units A because units of Y are coerced to A
```

8 If both operands of *level-4-expr* are of type real they shall have equivalent units. If only one operand is of type real its units shall be the intrinsic unit UNITLESS.

5.9 Units in assignments

1 If the *variable* and *expr* in an intrinsic assignment are of real type they shall have equivalent units (5.3.5).

NOTE 5.14

Alternatives if units of *variable* are a conversion unit:

- a workable structural definition of equivalence of conversion units might be devised, or
- *variable* and *expr* can be related conversion units, with automatic conversion taking place during assignment.

2 If only one of the *variable* or *expr* in an intrinsic assignment is of real type it shall have UNITLESS units.

3 If the *data-pointer-object* and *data-target* in a pointer assignment statement are of real type, they shall have equivalent units (5.3.5).

5.10 Units during formatted input/output

5.10.1 Unit name format descriptors

1 A unit name edit descriptor suffix U[*w*][.*s*] is provided for D, E, EN, ES, F, or G edit descriptors.

2 If a D, E, EN, ES, F, or G edit descriptor has a U format descriptor suffix, and it corresponds to an input or output list item, the list item shall be of real type.

5.10.2 Formatted output of quantities with units and their unit names

1 If an effective list item is of real type, its units expression shall consist of *unit-name*.

NOTE 5.15

```
unit :: LENGTH, AREA = LENGTH**2
real, unit(length) :: X
real, unit(area) :: Y
write ( *, * ) X*X       ! prohibited
```


NOTE 5.15 (cont.)

```
write ( *, * ) AREA(X*X) ! allowed -- AREA(X*X) is a units confirmation
```

2 If an output list item appears in an output statement in which a *format* other than asterisk appears, and it corresponds to an edit descriptor that has a U[w][.s] suffix, the value of w to be used to output the unit name is specified by that suffix. The width of the field is w if w appears, and the number of characters in the unit name otherwise. The value of w shall not be zero. If .s does not appear, the unit name shall be separated from the value by one blank. If .s appears, the unit name shall be separated from the value by s blanks. The value of s shall not be negative. The unit name shall be output if and only if the list item corresponds to an edit descriptor that has a U[w][.s] suffix. A U[w][.s] edit descriptor suffix shall not apply to an output item that is not of real type.

NOTE 5.16

The UNITLESS unit name is intrinsic. It is not an intrinsic function so its name cannot appear in an intrinsic statement (although the UNITLESS intrinsic coercion function name can). Therefore, the unit name UNITLESS cannot be renamed during use association (but the UNITLESS intrinsic coercion function name can be).

3 Unit name editing is the same as for character editing using an A[w] edit descriptor. The case of unit names in output is processor dependent.

4 For list-directed or namelist output, if the units of a real output item are the intrinsic unit UNITLESS, the unit name shall not be output. Otherwise the unit name is output after the value, separated from the value by one space; the width is the number of characters in the unit name.

5.10.3 Formatted input of quantities with units and their unit names

1 In a formatted READ statement in which a *format* other than asterisk appears, if a real input list item has units other than the intrinsic unit UNITLESS, the edit descriptor shall have a U[w][.s] suffix, and a unit name shall appear in the field of width w. If .s does not appear, the unit name shall be separated from the value by one blank. If .s appears the unit name shall be separated from the value by s blanks. The value of s shall not be negative. Unit name editing is the same as for character editing using an A[w] edit descriptor. If w appears it shall not be zero.

2 If a scalar of real type is input using list-directed or namelist input, and its units are not the intrinsic unit UNITLESS, a unit name shall follow the value on the same line and be separated from it by one or more spaces. If the units of the input item are UNITLESS, a unit name shall not appear.

3 If an array of real type is input using list-directed or namelist input, and its units are not the intrinsic unit UNITLESS, a unit name shall follow the first value and be separated from it by one or more blank spaces, and if more than one element is input, a unit name shall follow the last value and be separated from it by one or more blank spaces. Whether a unit name follows other elements is optional. If a unit name is not specified for the value of an array element, its units are the same as for the previous element. If the units of the array are the intrinsic unit UNITLESS, a unit name shall not appear.

NOTE 5.17

It is necessary for a unit name to follow a scalar item or the last element of an array item to avoid an ambiguity if the next effective item is of character type. It is necessary for a unit name to follow the first item in an array so as not to require the processor to do retroactive conversion.

4 The input unit name shall be the same as the declared units of the input list item, or shall be in the same unit family, without regard to case. If the input unit name is not the same as the units of the input list item, the sequence of conversions between the input unit and declared unit is applied to convert the

input value to the units of the input list item. If the input unit name is not the same as the units of the input list item and not in the same unit family as the units of the input list item, an error condition exists.

5 A unit other than the intrinsic unit UNITLESS can be renamed during USE association. The unit name that appears in the input shall be the name used in the declaration of the input list item.

NOTE 5.18

If the program that read the impulses of attitude-control maneuvers of the Mars Climate Orbiter had included the following definitions

```
UNIT :: KG              ! Kilogram mass
UNIT :: M               ! Meter
UNIT :: S               ! Second
UNIT :: A = M / S**2    ! Acceleration
UNIT :: Newton = KG * A ! F = ma!
UNIT :: NS = Newton * S
UNIT :: LbS = NS * 4.4482216152605
```

and the input variables had been given the units NS, and the file Lockheed had sent to report the impulses had included the unit LBS instead of just a number, the reported values would automatically have been converted to the correct units. If the file had not included units an error would have been announced. Either way, a loss of $\$3 \times 10^8$ would have been averted – unless Lockheed reported the impulses in pound-second units and claimed they were Newton-second units, but that would have required committing two errors.

5.11 Units of dummy and actual arguments

5.11.1 Characteristics, explicit interface

1 The units of dummy arguments, and if they are abstract the relationship of those units to the units of other dummy arguments or a function result variable, are characteristics of the procedure.

2 If a procedure has a dummy argument that has a unit other than UNITLESS, its interface shall be explicit in a scoping unit where it is referenced, appears as an actual argument, or appears as a procedure pointer target.

5.11.2 Abstract units

1 Abstract units may be specified for dummy arguments. Abstract units may be atomic or may be composite units composed from other abstract units.

2 If the unit of a dummy argument or function result is abstract and composite, the atomic form of that unit shall consist only of abstract units of dummy arguments of the same procedure, or a function result variable of the same procedure.

5.11.3 Actual arguments corresponding to dummy arguments with nonabstract units

1 An actual argument that corresponds to a dummy argument that has nonabstract units shall have units that are equivalent to the units of the corresponding dummy argument.

NOTE 5.19

Alternatives if units of the actual argument are a conversion unit:

- a workable structural definition of equivalence of conversion units might be devised, or

NOTE 5.19 (cont.)

- the actual and dummy arguments can be conversion units in the same family, with automatic conversion taking place during association, implying copy-in/copy-out.

5.11.4 Actual arguments corresponding to dummy arguments with abstract units

1 An actual argument that corresponds to a dummy argument that has an atomic abstract unit need not have the same units as the dummy argument.

2 If an actual argument corresponds to a dummy argument that has abstract units, its units shall be related to the units of other actual arguments in the same way that the units of their corresponding dummy arguments are related.

NOTE 5.20

It's not obvious that this would work if abstract units were allowed to be composed of nonabstract units.

NOTE 5.21

Because of the rules for construction of units of results of expressions involving multiplication and exponentiation operations, if an actual argument with UNITLESS units corresponds to a dummy argument with an atomic abstract unit, that abstract unit behaves as an identity in compositions that define other abstract units. That is, for any unit U, the units of UNITLESS*U and U*UNITLESS are U, and the units of UNITLESS***int-literal-constant* or RATIONAL_POWER(UNITLESS,*int-literal-constant*,*int-literal-constant*) are UNITLESS.

5.12 Units of function result variables and function references

1 The units of function results, and if they are abstract their relationship to units of dummy arguments, are a characteristic of the procedure.

2 Function result variables can have abstract or nonabstract units.

3 If the units of the function result variable are nonabstract, the units of the function reference are the same as the units of the function result variable.

4 If the unit of the function result variable is an abstract unit, the atomic form of that unit shall consist only of abstract units of dummy arguments of the same procedure.

5 If the unit of the function result variable is abstract the units of the result of a function reference are related to the units of the actual arguments in the same way that the units of the result variable are related to the units of the dummy arguments.

6 If the units of the function result variable are not the intrinsic unit UNITLESS, the interface of the function shall be explicit in a scoping unit where the function name appears.

5.13 Procedure pointer assignment

1 A procedure pointer that has implicit interface shall not have a target that has a dummy argument that has a unit other than UNITLESS. A function procedure pointer shall not have a target that has result units different from the result units of the pointer.

5.14 Units and generic resolution

1 If all of the dummy arguments of some pair of specific interfaces in a generic interface have the same type, kind and rank, at least one of the dummy arguments of one of the specific procedures shall have units that are not equivalent to the units of the corresponding dummy argument in the other specific procedure, and those two dummy arguments shall not both have abstract units.

2 If the characteristics of two specific procedures in a generic interface are identical except that the units of some dummy arguments of the first are not abstract while the units of corresponding units of the second are abstract, the first procedure shall not have any arguments with abstract units.

3 If the characteristics of two specific procedures in a generic interface are identical except for the units of some dummy arguments, and those dummy arguments of the first procedure have abstract units, and the corresponding actual arguments of the second have the same units as the dummy argument of the second procedure, the second procedure is invoked.

5.15 Intrinsic units

1 This technical specification defines an intrinsic atomic unit UNITLESS, an intrinsic units coercion function UNITLESS that coerces an actual argument with any units to a result with UNITLESS units, and an intrinsic units confirmation function UNITLESS.

2 This technical specification defines an intrinsic atomic unit RADIAN. The intrinsic unit RADIAN defines unit coercion and confirmation functions as described in subclause 5.3.8 of this technical specification.

5.16 UNITLESS (A)

1 **Description.** Value of the argument with units UNITLESS.

2 **Class.** Elemental function.

3 **Argument.** A shall be of real type with any units.

4 **Result Characteristics.** Same type and kind as A with units UNITLESS.

5 **Result Value.** Same as A.

5.17 RATIONAL_POWER (X, N, D)

1 **Description.** Raise a number to a rational exponent.

2 **Class.** Elemental function.

3 **Arguments.**

X	shall be of numeric type. If X is of type real the units of X are abstract; otherwise X has no units.
N	shall be of integer type. If X is real and the units of the actual argument associated with X are not the intrinsic unit UNITLESS then the actual argument associated with N shall be a constant expression.
D	shall be of integer type. If X is real and the units of the actual argument associated with X are not the intrinsic unit UNITLESS then the actual argument associated with D shall be a constant expression.

4 **Result Characteristics.** Default real if X is of integer type; otherwise the same type and kind as X.

If X is of type real, the units of the result are the units of X raised to the rational power N/D.

NOTE 13.20a

If the units of X are the intrinsic unit UNITLESS the units of the result are UNITLESS.

5 **Result Value.** The result has a value equal to a processor-dependent approximation to X raised to the rational power N/D.

6 **Example.** Assume there is an atomic unit LENGTH, a unit VOLUME=LENGTH**3, and a unit AREA=LENGTH**2. The value of the result of RATIONAL_POWER (VOLUME(8.0), 2, 3) is $8.0^{2/3} = 4.0$ (approximately) and the units of the result are $\text{VOLUME}^{2/3}$ = LENGTH**2, which is equivalent to AREA.

5.18 Units relationship to other intrinsic procedures

1 If an intrinsic function has any arguments of real type but the result is not of real type, the units of the arguments are the intrinsic unit UNITLESS. If an intrinsic function has a real result but no arguments of real type, the units of the result are the intrinsic unit UNITLESS.

2 With the following exceptions, the units of real arguments of an intrinsic function with real result are the same abstract unit, and the units of the result are that abstract unit.

3 The units of the arguments and results of the following functions are UNITLESS.

Hyperbolic functions	Inverse hyperbolic functions	Bessel functions
Error functions	EXP	FRACTION
GAMMA	LOG	LOG_GAMMA
LOG10	PRODUCT	RRSPACING
TRANSFER		

Inverse trigonometric functions other than ATAN2 or ATAN with two arguments

4 The arguments of ATAN2, or ATAN with two arguments, have the same abstract unit. The result of ATAN2 or ATAN with two arguments has the intrinsic unit UNITLESS.

5 Trigonometric functions have arguments with either UNITLESS or RADIAN units, and UNITLESS results.

NOTE 5.22

Although it might be desirable that inverse trigonometric functions return results with either UNITLESS or RADIAN units, this is not possible because characteristics of function results do not participate in generic resolution.

6 Where the following intrinsic functions have real arguments, their arguments have atomic abstract units A and B, and the units of the result are the composite abstract unit C = A*B.

DOT_PRODUCT DPROD MATMUL

NOTE 5.23

The unit expression UNITLESS*UNITLESS is also UNITLESS. Therefore the above functions can be applied to UNITLESS arguments.

7 Where the arguments of MOD or MODULO are real they have different abstract units, and the units

of the result are those of the first argument.

8 The argument of RANDOM_NUMBER has an abstract unit.

9 The argument of the SQRT function has an abstract unit A and the unit of the result is A**(1/2), where 1/2 denotes a rational fraction, not an integer division operation.

10 Where the first argument of RATIONAL_POWER is real, that argument has an abstract unit A, and the unit of the result is A**(N/D), where N and D are the values of the second and third arguments, and N/D denotes a rational fraction, not an integer division operation.

NOTE 5.24

> The unit expression UNITLESS**(N/D) is also UNITLESS. Therefore the SQRT or RATIONAL_POWER function can be applied to a UNITLESS actual argument and produce a UNITLESS result.

11 The unit of the result of the UNITLESS intrinsic function is the intrinsic unit UNITLESS.

6 Functionality not requested at this time

1 If the type parameter system were extended to units, it would be possible for components to have abstract units. Where objects are declared, their unit parameter values would have to be related to each other in the same way that abstract units of corresponding type parameters are related, similar to the requirement in this technical specification concerning the relationship between actual and dummy arguments. Where unit parameters in a type definition are not abstract, objects' corresponding unit parameter values would have to match exactly.

2 This would allow a procedure to have a dummy argument of a derived type that has abstract unit parameters, with the unit parameter values of the dummy argument being abstract. For example, in a procedure for solution of differential equations, a dummy argument might have parameters that specify abstract units, say state and independent variable. Users of the library procedure would have their own units, say meters and seconds.

3 Components could have composite abstract units, such as derivative = state / independent variable. It would be desirable to allow to define abstract units, especially composite abstract units, within a type definition.

4 It is possible in principle to allow entities of complex type to have units. The real and imaginary parts could have independent units. The descriptions would be complicated.

5 It might be possible in principle to allow entities of integer type to have units. The definition of the units of results of expressions involving integer division would be controversial.

6 It is unlikely to be useful to allow entities of derived type to have units. It is sufficient that their components have units, especially if those units can be type parameters.

7 It does not seem to be possible, even in principle, to allow entities of logical or character type to have units.

8 Unit conversions could be performed during argument association, implying copy-in/copy-out argument passing. Additional changes to the rules for distinguishing generic procedures would be necessary .

7 Required editorial changes to ISO/IEC 1539-1:2010(E)

The following editorial changes to ISO/IEC 1539-1:2010(E), if implemented, would provide the facilities described in foregoing clauses of this report. Descriptions of how and where to place the new material are enclosed between square brackets.

[Introduction, probably page xv] Editor: Insert a bullet item

- "Units of numerical quantities:
  Units for real data objects may be defined. Data entities may have units. Units are combined and checked in expressions, argument association, assignment, and input/output, and converted if necessary during input."

[20:26-29 1.3.151] Editor: Replace the subclause:

1.3.151
unit
- means, specified by an *io-unit*, for referring to a file (9.5.1)
- measure for a numerical quantity, such as meters or seconds

1.3.151.1
unit family
set of units that can be related by sequences of conversions (4.4a.7)

[24:7+ 1.6.1p1+] Editor: Insert a subclause:

1.6.1a Fortran 2008 compatibility

This part of ISO/IEC 1539 is an upward compatible extension to the preceding Fortran International Standard, ISO/IEC 1539-1:2010(E) (Fortran 2008). This part of ISO/IEC 1539 defines intrinsic UNITLESS and RATIONAL_POWER functions. Therefore, a Fortran program conforming to an older standard might have a different interpretation or no interpretation under this part of ISO/IEC 1539 if it has functions of those names that are not declared explicitly to have the EXTERNAL attribiute.

[28:41 R212] Editor: Insert an alternative for R212 *other-specification-stmt*

R212	*other-specification-stmt*	**is**	*abstract-stmt*
		or	*access-stmt*

[29:2+ R212] Editor: Insert an alternative for R212 *other-specification-stmt*

		or	*exclude-arithmetic-stmt*

[29:11+ R212] Editor: Insert alternatives for R212 *other-specification-stmt*

		or	*unit-definition-stmt*
		or	*unit-stmt*

[60:20+ 4.4p63+] Editor: Insert a subclause

4.4a Units of data objects of real type

4.4a.1 General

In addition to a type, data objects of real type may have a UNIT attribute. The UNIT definition statement defines the name of a unit, what category of unit it is, if and how it is related to other units, and units conversion, coercion, and confirmation functions. Each unit defined by a UNIT definition statement is a different unit, even if it has the same name as a unit defined in a different scoping unit.

4.4a.2 Categories of units

4.4a.2.1 General

A unit is an atomic unit, a composite unit, a conversion unit, or an abstract unit.

4.4a.2.2 Atomic units

An atomic unit is one that is not defined in terms of other units.

4.4a.2.3 Composite units

A composite unit is one that is defined by an expression using multiplication of units, division of units, or exponentiation of a unit by an integer or a ratio of integers.

4.4a.2.4 Conversion units

A conversion unit is one that is defined in terms of one other unit by a linear transformation having constant numeric coefficients.

4.4a.2.5 Abstract units

Abstract units specify the relationship between units of dummy arguments, or the dependence of function result units on dummy argument units, and thereby the relationship between units of corresponding actual arguments, or between actual arguments and the results of function references, without requiring specific units for actual arguments.

Abstract units are explicitly declared to be abstract. They can be atomic, or they can be composite provided they are defined only in terms of abstract units.

4.4a.3 Syntax of UNIT definition statement

R424a *unit-definition-stmt* **is** UNIT [[, *unit-attr-list*] ::] *unit-definition-list*

R424b *unit-attr* **is** ABSTRACT
or EXCLUDE_ARITHMETIC
or *access-spec*

R424c *unit-definition* **is** *unit-name* [= *unit-expr*]

C425a (R424a) The double colon shall appear if *unit-expr* appears in any *unit-definition* in the statement.

Unresolved Technical Issue C425b

C514 might be construed to imply the following constraint. Even though it is in Clause 5, it constrains entities.

C425b (R424b) A *unit-attr* shall not be specified more than once for a unit name, no matter how specified.

NOTE 4.14a

If *unit-name* is UNITLESS or RADIAN, the intrinsic unit, unit coercion function, and unit con-

NOTE 4.14a (cont.)

firmation function with that name are not accessible.

R424d *unit-expr* **is** *unit-conversion-expr*
or *unit-composition-expr*

R424e *unit-conversion-expr* **is** *mult-operand* * *unit-name* [*mult-op mult-operand*] ■
■ [*add-op add-operand*]
or [*mult-operand* *] *unit-name mult-op mult-operand* ■
■ [*add-op add-operand*]
or *unit-name add-op add-operand*
or [*mult-operand* *] (*unit-conversion-expr*) ■
■ [*mult-op mult-operand*] [*add-op add-operand*]

C425c (R424e) The *unit-name* shall be defined previously within the same scoping unit, or accessible by use or host association. It shall not be the name of a conversion unit or abstract unit, or specify the intrinsic unit UNITLESS.

C425d (R424e) Each *add-operand* and *mult-operand* shall be a unitless constant expression of real or integer type.

C425e (R424e) The *unit-name* shall not have the EXCLUDE_ARITHMETIC attribute.

C425f (R424e) After substitution of values for named constants, and algebraic simplification, *unit-conversion-expr* is always of the form $a \times$*unit-name*$+b$ where a and b are unitless numeric constants. If no *mult-operand* appears the value of a is 1. If no *add-operand* appears, the value of b is zero. The value of a shall not be zero.

NOTE 4.14b

If $a = 1$ and $b = 0$, a unit synonym is effectively created. The units are related by conversion, so they can be used interchangeably in input. See 10.7.5a.

R424f *unit-composition-expr* **is** *unit-factor* [*mult-op unit-composition-expr*]

R424g *unit-factor* **is** *unit-factor* ** *int-literal-constant*
or *unit-factor* ** (*signed-int-literal-constant*)
or RATIONAL_POWER (*unit-factor*, ■
■ *signed-int-literal-constant*, *int-literal-constant*)
or *unit-name*
or (*unit-composition-expr*)

C425h (R424f) Either *mult-op* or *int-literal-constant* shall appear.

C425i (R424g) The *unit-name* shall be defined previously within the same scoping unit, or accessible by use or host association.

C425j (R424g) The *unit-name* shall be the name of an abstract unit if and only if the *unit-name* being defined by the *unit-definition-stmt* is an abstract unit.

C425k (R424g) If *unit-name* has the EXCLUDE_ARITHMETIC attribute, the unit being defined shall have the EXCLUDE_ARITHMETIC attribute.

C425l (424h) Neither *int-literal-constant* shall be zero.

If *unit-factor* is RATIONAL_POWER(u,n,d), it specifies a factor of the form $u^{\frac{n}{d}}$, where $\frac{n}{d}$ is a rational fraction, not Fortran integer division.

A unit name is abstract if and only if ABSTRACT appears in the *unit-definition-stmt* that defines it, or the ABSTRACT attribute is specified for it by an ABSTRACT statement.

4.4a.4 Excluded operations

Addition and subtraction are prohibited between objects that have units for which the EXCLUDE_-ARITHMETIC *unit-attr* is specified. If an operand of a *mult-op* in a *level-2-expr* has units for which the EXCLUDE_ARITHMETIC *unit-attr* is specified, the units of the other operand shall not be the intrinsic unit UNITLESS.

4.4a.5 Equivalence of units

The atomic form of an atomic unit is the unit name. The atomic form of a conversion unit is its unit name.

An atomic unit is not equivalent to any other unit. A conversion unit is equivalent only to its accessible synonyms.

NOTE 4.14d

A unit might have different names in different scoping units.

The atomic form of a composite unit is produced by replacing each composite unit in its defining expression by that unit's atomic form. If a unit appears more than once the several appearances are combined into one by adding their exponents, in the usual manner of algebraic simplification. This results in an algebraic expression of the form $\prod_{i=1}^{n} a_i^{e_i}$, where n is the number of units in the form, each a_i is an atomic unit or a conversion unit, no two units in the expression are equivalent, and each e_i is either an integer or the ratio of two integers that have no common factors.

This process terminates because the units that appear in a unit definition expression are required to be previously defined or accessible by use or host association.

C425m In the atomic form of a composite unit, the value of e_i shall not be zero.

C425n In the atomic form of a composite unit, if several a_i are conversion units in the same family (4.4a.6), the sum of their exponents e_i shall not be zero.

NOTE 4.14d′

Prohibiting the sum of exponents of conversion units in the same family from being zero prevents such absurdities as CENTIMETER/INCH, which is the number 2.54. This would make a composite unit a conversion unit.

NOTE 4.14e

The processor cancels common factors to produce e_i, to make it easier to compare atomic forms.

Although each unit definition defines a different unit, composite units can be equivalent. Composite units, whether abstract or not, are equivalent if and only if their atomic forms are equivalent. If the atomic form of one is $\prod_{i=1}^{n} a_i^{e_i}$ and the other is $\prod_{j=1}^{m} b_j^{e_j}$ then the units are equivalent if and only if $m = n$, there is a one-to-one correspondence between a_i and b_j such that a_i and b_j are equivalent, and $e_i = e_j$ where a_i and b_j are equivalent. The unit names a_i and b_j are equivalent if they refer to the same unit definition.

4.4a.6 Unit families

Among every set of conversion units that are related by *unit-definition*s in which every *unit-expr* is a *unit-conversion-expr*, there is exactly one unit that is not a conversion unit. Let that unit be called Z.

There is a sequence of conversions between units W ... Z if there is a set of definitions of conversion units W = f_{WX}(X), X = f_{XY}(Y), ... Y = f_{YZ}(Z), where each f is a mathematical function defined by the *unit-conversion-expr*. Denote the composition of these definitions by W = f_{WZ}(Z). The conversion relation between w and z, where w and z are values with units W and Z, respectively, is $w = f_{WX}(f_{XY}(\ldots f_{YZ}(z)\ldots)) = f_{WZ}(z)$. Since each *unit-conversion-expr* is linear and the *mult-operand* is nonzero, there are also inverse conversion relations Z = f_{ZY}(Y) = f^{-1}_{YZ}(Y), ... X = f_{XW}(W) = f^{-1}_{WX}(W), and a sequence of conversions $z = f_{YZ}(\ldots f_{YX}(f_{XW}(w))) = f_{ZW}(w) = f^{-1}_{WZ}(w)$.

If there is a unit A such that there is a sequence of conversions f_{AZ} between Z and A but no direct sequence of conversions f_{AW} between W and A, there is nonetheless a sequence of conversions between w and a where a has units A defined by $a = f_{AZ}(f_{ZW}(w))$. If A and W both depend upon some intermediate unit I $\neq$ Z, this sequence can be simplified since within it there is the identity conversion consisting of the sequence $f_{IZ}(f_{ZI}(z)) = f_{IZ}(f^{-1}_{IZ}(z)) = z$

Units that can be related by sequences of conversions constitute a unit family.

The atomic form of a composite unit can be represented by a tree in which every internal vertex represents an operator and every terminal vertex represents a unit name or a constant, and the relationship of internal vertices is consistent with the hierarchy of operator precedences established by the *unit-composition-expr*s that define the unit and the composite units in its *unit-composition-expr*. A tree that is isomorphic can be constructed by exchanging the subtrees of multiplication operators because multiplication of units is commutative.

If all conversion units in the tree that represents the atomic form of one composite unit can be put into one-to-one correspondence with all the conversion units in a tree that is isomorphic to the one that represents the atomic form of another composite unit, such that the units in every such pair of corresponding units are in the same unit family, then there is a conversion between the composite units that is constructed by applying the sequence of conversions between members of each corresponding pair, and the composite units are in the same unit family.

NOTE 4.14f

Consider the following definitions:

```
UNIT :: FOOT, SECOND
UNIT :: MILE = FOOT/5280.0, HOUR = SECOND/3600.0
! 5280.0 has units FOOT/MILE, 3600.0 has units SECOND/HOUR
UNIT :: FPS = FOOT/SECOND
UNIT :: MPH = MILE/HOUR
```

The units FPS and MPH are in the same family because `FOOT` is in the same family as `MILE` and `SECOND` is in the same family as `HOUR`.

4.4a.7 Unit conversion functions

A unit family defines a set of pure elemental generic unit conversion functions. For each unit, there is a generic function having the same name as the local name of the unit, with specific functions having a real argument with every kind, and every unit in the family. The result of each function has real type of the same kind as its dummy argument, and units specified by its generic name. Each specific function implements a conversion or sequence of conversions between a value having units of its argument and a value having units of its result variable.

NOTE 4.14g

Consider the following unit definitions:

```
UNIT :: KELVIN
UNIT :: CELSIUS = KELVIN - 273.15
UNIT :: FAHRENHEIT = 1.8 * CELSIUS + 32.0
UNIT :: RANKINE = 1.8 * KELVIN
```

CELSIUS, FAHRENHEIT and RANKINE are conversion units. Those units, together with KELVIN, constitute a unit family. The definition of CELSIUS defines a generic function named CELSIUS that converts values with units KELVIN to values with units CELSIUS. It also defines specific functions for the generic KELVIN that convert values having CELSIUS units to values having KELVIN units. The definition of FAHRENHEIT defines a generic function that converts values with units KELVIN or CELSIUS to values with units FAHRENHEIT, and specific functions for KELVIN and CELSIUS that convert values with units FAHRENHEIT to values with units KELVIN and CELSIUS, respectively. It is always possible to construct the inverse conversions because the *mult-operand* in a *unit-conversion-expr* cannot be zero. The Celsius to Fahrenheit conversion function is defined directly by the conversion expression in the definition of FAHRENHEIT; the Fahrenheit to Celsius conversion is defined indirectly by the inverse of that relation. The Kelvin to Fahrenheit conversion function is defined by applying the Kelvin to Celsius conversion and then the Celsius to Fahrenheit conversion. It is expected that the processor will algebraically simplify the function compositions. The usual round-off and truncation considerations apply, and the results might not be identical to an analytic composition.

Although there is no direct conversion relation between RANKINE and FAHRENHEIT, this nonetheless defines the same conversion as would be defined by

```
UNIT :: RANKINE = FAHRENHEIT + 459.67
```

by the sequences of conversions FAHRENHEIT↔CELSIUS↔KELVIN↔RANKINE.

4.4a.8 Unit coercion and confirmation functions

Definition of a unit defines two pure elemental specific functions in the generic interface having the same name as the unit, for each kind of real supported by the processor. Each has a dummy argument of real type. The result value of each is of real type, has the same kind and value as the dummy argument, and units that are the same as the function name.

The first is a **units coercion function**. Its dummy argument shall have UNITLESS units.

The second is a **units confirmation function**. Its dummy argument shall have units equivalent to the units of its result.

NOTE 4.14h

It is possible to coerce a value from one unit to another even if the units are neither equivalent nor in the same family by first removing the units using the UNITLESS function (13.7.173a) and then applying the units coercion function. The appearance of the UNITLESS function is a signal that this might be a dangerous coercion.

4.4a.9 ABSTRACT statement

The ABSTRACT statement specifies the ABSTRACT attribute for a unit name.

R424j *abstract-stmt* **is** ABSTRACT [::] *unit-name-list*

C425o (424j) Each *unit-name* shall be the name of a unit that is defined in the same scoping unit.

4.4a.10 EXCLUDE ARITHMETIC statement

The EXCLUDE ARITHMETIC statement specifies the EXCLUDE_ARITHMETIC attribute for the specified units.

R424k *exclude-arithmetic-stmt* **is** EXCLUDE ARITHMETIC [::] *unit-name-list*

C715 (R509) Each *unit-name* shall be the name of a unit that is defined in the same scoping unit.

4.4a.11 Intrinsic units

This part of ISO/IEC 1539 defines an intrinsic atomic unit UNITLESS, an intrinsic units coercion function UNITLESS that coerces an actual argument with any units to a result with UNITLESS units, and an intrinsic units confirmation function UNITLESS. This part of ISO/IEC 1539 defines an intrinsic atomic unit RADIAN. The intrinsic unit RADIAN defines unit coercion and unit confirmation functions (4.4a.8).

[66:13+ R437] Editor: Add an alternative for R439 *component-attr-spec*:

or *unit-attr-spec*

[69:20 4.5.4.6p2] Editor: After first "target" insert ", if the pointer variable or component is of real type its units are equivalent to the units of *initial-data-target* (4.4a.5)"

[88:10+ C505+] Editor: Insert a constraint:

C505a (R503) If *initialization* appears and *object-name* is of real type, the units of *object-name* and *initialization* shall be equivalent (4.4a.5)

[101:19- Note 5.24+] Editor: Insert a subclause:

5.3.17a UNIT attribute

The UNIT attribute specifies the units for a variable, structure component, or named constant of real type. If a unit attribute is not specified for a variable, structure component, or named constant of numeric type, the units of that entity are the intrinsic unit UNITLESS.

R523a *unit-attr-spec* **is** UNIT (*unit-name*)

C556a (R523a) The *unit-name* shall be the name of a unit that is defined previously within the same scoping unit, or accessible by use or host association.

C556b (R523a) The *unit-attr-spec* shall not be specified for an entity that is not a variable, structure component, named constant, or external function, or is not of real type.

C556c (R523a) If *unit-name* is abstract and composite, all abstract units in its atomic form shall be abstract units of nonoptional dummy arguments of the same procedure as for the entity being declared.

C556d (R523a) If *unit-name* is abstract, atomic, and is the unit of a function result, that unit shall appear in the atomic form of the unit of a dummy argument of that function.

[102:28+ R525] Editor: Add an alternative for R525 *access-stmt*:

or *unit-name*

[102:32+ C563+] Editor: Add a constraint:

C563a (R525) Each *unit-name* shall be the name of a unit (4.4a) that is defined previously within the same scoping unit, or accessible by use or host association.

[108:18- Note 5.36+] Editor: Insert a subclause:

5.4.15a UNIT statement

A UNIT statement specifies the units for a list of variables or named constants of real type.

R557a *unit-stmt* **is** UNIT (*unit-name*) [::] *entity-name-list*

C581a (R557a) The *unit-name* shall be the name of a unit that is defined previously within the same scoping unit, or accessible by use or host association.

C581b (R557a) The *entity-name* shall be the name of a variable, named constant, or external function, and shall be of real type.

[134:18+ R709+] Editor: Insert constraints and paragraphs:

C703a (R704) If a *mult-operand* includes a *power-op* the units of the second *mult-operand* of the *power-op* shall be UNITLESS. If the units of the *level-1-expr* are not UNITLESS, the second *mult-operand* of the *power-op* shall be a constant of type integer.

C703b (R705) If the units of *add-operand* or *mult-operand* have the EXCLUDE_ARITHMETIC unit attribute, the units of the other operand shall not be the intrinsic unit UNITLESS.

C703c (R706) The units of *level-2-expr* or *add-operand* shall not have the EXCLUDE_ARITHMETIC unit attribute.

C703d (R706) If a *level-2-expr* includes an *add-op* with two operands and if both operands of the *add-op* are of real type, the operands shall have equivalent units (4.4a.5) that do not have the EXCLUDE_ARITHMETIC attribute. If only one operand is of real type, its units shall be the intrinsic unit UNITLESS.

If the result type of *mult-operand*, *add-operand*, or *level-2-expr* is not real the result has no units.

If the units of the *level-1-expr* of the *power-op* are the intrinsic unit UNITLESS the units of the *mult-operand* are the intrinsic unit UNITLESS. Otherwise the units expression of the *mult-operand* is formed by multiplying the exponents of every factor of the units expression of the *level-1-expr* by the value of the *mult-operand* of the *power-op*.

If an *add-operand* includes a *mult-op* that is * the units expression of the *add-operand* is formed by multiplying the units expression of the first *add-operand* of the *mult-op* by the units expression of the second *mult-operand*. If the units of one of the operands is the intrinsic unit UNITLESS the units of the result are the units of the other operand.

If an *add-operand* includes a *mult-op* that is / the units expression of the *add-operand* is formed by dividing the units expression of the first *add-operand* of the *mult-op* by the units expression of the second *mult-operand*. If the units of the second operand are the intrinsic unit UNITLESS the units of the result are the units of the first operand. If the units of the first operand are the intrinsic unit

UNITLESS the units of the result are the inverse of the units of the second operand.

If a *level-2-expr* consists of *add-operand* or *add-op add-operand*, the units of the *level-2-expr* are the units of the *add-operand*.

If a *level-2-expr* consists of *level-2-expr add-op add-operand*, the units of the result are the units of the operand of real type.

C703c (R704) The atomic form of the units expression of a *mult-operand* shall satisfy the requirements specified in 4.4a.5.

[135:1- Note 7.3+] Editor: Insert notes and a paragraph:

NOTE 7.3a

```
unit :: A, B, SQRTA=A**(1/2)
real, unit(sqrta) :: X
real, unit(B) :: Y
real, unit(A) :: Z
X**Y    ! prohibited because Y is not unitless
X**2    ! has units SQRTA**2, which is equivalent to A
X**3    ! has units SQRTA**3, which is equivalent to A**(3/2)
```

NOTE 7.3b

```
X*Y     ! has units SQRTA*B
7*X     ! has units SQRTA
```

NOTE 7.3c

```
X + Y              ! prohibited because SQRTA and B are not equivalent
Z + A(UNITLESS(Y)) ! has units A because units of Y are coerced to A
```

[135:19+ R713+] Editor: Insert a constraint:

C703d (R712) If both operands of *level-4-expr* are of real type they shall have equivalent units. If only one operand is of real type its units shall be the intrinsic unit UNITLESS.

[153:21,24,24+ 7.2.12p1(7-8)] Editor: Delete "and" at the end of item (7); insert list items:

(9) if the *variable* and *expr* in an intrinsic assignment are of real type they shall have equivalent units (4.4a.5), and

(10) if only one of the *variable* or *expr* in an intrinsic assignment is of real type its units shall be the intrinsic unit UNITLESS.

[157:36+ C714+] Editor: Insert a constraint:

C714a (R733) If *data-target* is of real type, the units of *data-pointer-object* and *data-target* shall be equivalent (4.4a.5)

[160:13+ 7.2.2.4p5+] Editor: Insert a paragraph:

If the pointer object has an implicit interface, *proc-target* shall not be a procedure that has a dummy argument or result with units other than the intrinsic unit UNITLESS.

[160:15 7.2.2.4p6] Editor: Append a sentence to the paragraph:

"The units of the result of the pointer object shall not be different from the units of the result of the pointer target."

[170:33 8.1.3.2p1] Editor: Before "If" insert a sentence: "If the selector is of real type the units of the associating entity are assumed from the units of the selector."

[217:16+ C935+] Editor: Insert a constraint:

C935a (R917) If an *output-item* is of real type, its unit expression (4.4a.3) shall consist of *unit-name*.

NOTE 9.31a

```
UNIT :: METER, STERE=METER**3
REAL, UNIT(METER) :: W = 2.0
WRITE ( *, '(G15.8,1x,U5)' ) W**3 ! Illegal because W**3 has no unit name
WRITE ( *, '(G15.8,1x,U5)' ) STERE(W**3) ! Allowed; STERE is a confirmation
```

[247:3-7,10 R1007] Editor: Replace the descriptions for D, E, EN, ES, F, and G edit descriptors in *data-edit-desc*:

or F*w*.*d*[U[*w*][.*s*]]
or E*w*.*d*[E*e*][U[*w*][.*s*]]
or EN*w*.*d*[E*e*][U[*w*][.*s*]]
or ES*w*.*d*[E*e*][U[*w*][.*s*]]
or G*w*.*d*[E*e*][U[*w*][.*s*]]
or D*w*.*d*[U[*w*][.*s*]]

[247:15+ R1011] Editor: Insert a syntax rule:

R1011a *s* **is** *int-literal-constant*

[259:8+ 10.7.5.4+] Editor: Insert a subclause:

10.7.5a Unit name editing

TheU[*w*][.*s*] edit descriptor suffix may specify the width to use for output and input of unit names. Unit names are edited as described for character editing (10.7.4) using an A[*w*] edit descriptor with the same value of *w* (if any). If .*s* does not appear, the unit name shall be separated from the value by one blank. If .*s* appears, the unit name shall be separated from the value by *s* blanks. If a D, E, EN, ES, F, or G edit descriptor has a U suffix, and it corresponds to an input or output list item, the list item shall be of real type.

During output, if a U[*w*][.*s*] suffix does not appear, the unit name of the effective item is not output.

If an effective item corresponds to an edit descriptor that has a U[*w*][.*s*] suffix, the local name of the unit of the effective list item is output. The case of unit names in output is processor dependent.

During input, if a U[*w*][.*s*] suffix does not appear, a unit name is not read or checked during input. Otherwise, a unit name shall appear in the input, in the field of width *w* specified by the U[*w*][.*s*] suffix. If .*s* does not appear, the unit name shall be separated from the value by one blank. If .*s* appears, the unit name shall be separated from the value by *s* blanks. The unit name shall be the local name of a unit that is equivalent (4.4a.5) to the unit of the effective list item or in the same unit family (4.4a.6), without regard to case. If the input unit and the unit of the effective list item are in the same unit family, conversion takes place as if the conversion function defined by the unit of the effective list item were applied to the input value.

[263:14 10.10.3p4] Editor: Append sentences: “If the item is a real scalar and has units other than the intrinsic unit UNITLESS, a unit name shall follow the value, separated from the value by one or more spaces. If the item is a real array and has units other than the intrinsic unit UNITLESS, a unit name shall follow the first value, separated from the value by one or more spaces, and may follow other values provided the unit specified for each value after the first is equivalent to the unit for the previous value or related to it by conversion. If a unit is not specified for the value of an array element the unit of the previous value is used. If the units of the item are the intrinsic unit UNITLESS, a unit name shall not appear. The unit specified in the input shall be related to the unit of the item as specified in 10.7.5a. If the relationship is a conversion, conversion shall be applied as specified in 10.7.5a.”

[265:19 10.10.4p5] Editor: Append sentences: “If the item is scalar and has units other than the intrinsic unit UNITLESS, its unit name shall follow the value, separated from the value by one or more spaces and be edited as if by a U edit descriptor without w. If the item is an array and has units other than the intrinsic unit UNITLESS, its unit name shall follow the value of the first element of the array, separated from that value by one or more spaces, and edited as if by a U edit descriptor without w.”

[278:17 12.3.2.2p1] Editor: After “(if any)” insert “its nonabstract units (if any), the relationship of its units to the units of other dummy arguments if its units are abstract”

[278:30 12.3.3p1] Editor: After “(if any)” insert “its nonabstract units (if any), the relationship of its units to the units of dummy arguments if its units are abstract”

[279:21- 12.4.2.2p1(1)(a-)] Editor: Insert a list item:

“(a′) with an actual argument that is a procedure that has an argument or result variable that has units other than the intrinsic unit UNITLESS,”

[279:24- 12.4.2.2p1(2)(a-)] Editor: Insert a list item:

“(a′) has units other than the intrinsic unit UNITLESS,”

[279:31- 12.4.2.2p1(3)(a-)] Editor: Insert a list item:

“(a′) has units other than the intrinsic unit UNITLESS,”

[279:35+ 12.4.2.2p1+] Editor: Insert a paragraph:

“A procedure shall have explicit interface if it is an actual argument in a reference to a procedure that has explicit interface and the corresponding dummy procedure has an argument or result variable that has units other than the intrinsic unit UNITLESS.”

[285:15-16 12.4.3.4.5p2] Editor: After first “kind” insert “, units”; replace “TKR” by “TKUR”; after “second” insert “, if they are of type real the units of the first are equivalent to the units of the second or one has abstract units”; change the index item to “TKUR compatible”.

[286:3 12.4.3.4.5p3 second item] Editor: Replace “TKR” by “TKUR”.

[286:18 C1215(1)(a)] Editor: Replace “TKR” by “TKUR”.

[286:33+ C1215+] Editor: Insert two constraints:

C1215a Within the scope of a generic name that is the same as a unit name, a specific procedure shall not have a *dummy-arg-name-list* that is consistent with a reference to a units conversion, confirmation, or coercion function (4.4a.7, 4.4a.8).

C1215b If the only difference between the characteristics of the dummy arguments of two procedures within the scope of a generic identifier is that one has an argument of real type with nonabstract units while the other has a corresponding argument with abstract units, the first procedure shall have no arguments with abstract units.

[288:24+ 12.4.3.6p9+] Editor: Insert a paragraph:

If *procedure-entity-name* has an implicit interface, *initial-proc-target* shall not be a procedure that has a dummy argument or result with units other than the intrinsic unit UNITLESS.

[293:14+ 12.5.2.4p4+] Editor: Insert paragraphs and notes:

An actual argument that corresponds to a dummy argument that has nonabstract units shall have units that are equivalent to the units of the corresponding dummy argument.

NOTE 12.21a

Conversion units are not equivalent to the units to which they are related by conversions. For example, a unit CENTIMETER is not equivalent to a unit INCH defined as 2.54*CENTIMETER.

An actual argument that corresponds to a dummy argument that has an atomic abstract unit need not have the same units as the dummy argument.

If an actual argument corresponds to a dummy argument that has abstract units, its units shall be related to the units of other actual arguments in the same way that the units of their corresponding dummy arguments are related.

NOTE 12.21b

For example, if one dummy argument has abstract units A and another has abstract units A**2, the corresponding actual arguments could have units LENGTH and AREA, where AREA is defined as LENGTH**2.

NOTE 12.21c

Because of the rules for construction of units of results of expressions involving multiplication and exponentiation operations, if an actual argument with UNITLESS units corresponds to a dummy argument with an atomic abstract unit, that abstract unit behaves as an identity in compositions that define other abstract units. That is, for any unit U, the units of UNITLESS*U and U*UNITLESS are U, and the units of UNITLESS***int-literal-constant* or RATIONAL_POWER(UNITLESS,*int-literal-constant*,*int-literal-constant*) are UNITLESS.

The units of the dummy argument of a units conversion function shall be in the same units family (4.4a.6) as the units of the function result. The units of the dummy argument of a units coercion function (4.4a.8) shall be the intrinsic unit UNITLESS. The units of the dummy argument of a units confirmation function (4.4a.8) shall be equivalent (4.4a.5) to the units of the function result.

[302:6 12.5.3p1] Editor: Before "If" insert a sentence: "If the function result variable has abstract units, the units of the result value of the function reference are related to the units of the actual arguments in the same way that the units of the function result variable are related to the units of the dummy argument.

[303:27,32 12.5.5.2p1,p2] Editor: Replace "one such specific procedure" by "two such specific procedures, and only one of those shall have arguments with abstract units. If the reference is consistent with the procedure with arguments with nonabstract units, the reference is to that procedure; otherwise the reference is to the procedure with arguments with abstract units" twice.

[304:1- Note 12.39+] Editor: Insert a note:

NOTE 12.39a

These rules allow a particular specific procedure with nonabstract units to be used if it is consistent with a reference, and one with abstract units to be used otherwise. Assume the following

```
UNIT :: A, A_I=A**(-1), A2_I=A**(-2), B, B_I=B**(-1), B2_I=B**(-2)
REAL, UNIT(A_I) :: X_I
REAL, UNIT(A2_I) :: X2_I
REAL, UNIT(B_I) :: Y_I
REAL, UNIT(B2_I) :: Y2_I
```

Assume there are two procedures with generic name G, with specific names and arguments P(P1,P2) and Q(Q1,Q2), where P1 has units A_I, P2 has units A2_I, Q1 has an abstract unit C, and Q2 has an abstract unit C**2. Then a reference G(X_I, X2_I) would be consistent with P, while a reference G(Y_I,Y2_I) would be consistent with Q. A reference G(X_I,Y2_I) would not be consistent with either P or Q.

[305:21 12.6.2.1p1] Editor: Append a sentence: "A unit definition defines conversion (4.4a.7), coercion and confirmation (4.4a.8) functions of the same name."

[306:35+ C1256+] Editor: Insert a constraint:

C1256a (R1227) If the function result variable has abstract units, each abstract unit in the atomic form of its units shall be related to the units of a dummy argument.

[307:8 12.6.2.2p3] Editor: After "result" insert "has units other than the intrinsic unit UNITLESS, or"

[316:29+ 13.2.4p2+] Editor: Insert a subclause

13.3a Units of intrinsic function arguments and results

If an intrinsic function has any arguments of real type but the result is not of real type, the units of the arguments are the intrinsic unit UNITLESS. If an intrinsic function has a real result but no arguments of real type, the units of the result are the intrinsic unit UNITLESS.

With the following exceptions, the units of real arguments of an intrinsic function with real result are the same abstract unit, and the units of the result are that abstract unit.

The units of the arguments and results of the following functions are the intrinsic unit UNITLESS.

Hyperbolic functions	Inverse hyperbolic functions	Bessel functions
Error functions	CMPLX	EXP
FRACTION	GAMMA	LOG
LOG_GAMMA	LOG10	PRODUCT
RRSPACING	TRANSFER	

Inverse trigonometric functions other than ATAN2 or ATAN with two arguments

The arguments of ATAN2 or ATAN with two arguments have the same abstract unit. The result of ATAN2 or ATAN with two arguments has the intrinsic unit UNITLESS.

NOTE 13.3a

The results of inverse trigonometric functions mathematically have RADIAN units. For compatibility with previous standards, their results in this part of ISO/IEC 1539 are the intrinsic unit UNITLESS, because function result characteristics are not used for generic resolution. Programs can coerce the result units to RADIAN.

Trigonometric functions have arguments with units either the intrinsic unit UNITLESS or the intrinsic unit RADIAN, and results with the intrinsic unit UNITLESS.

Where the following intrinsic functions have real arguments, their arguments have atomic abstract units A and B, and the units of the result are the composite abstract unit C = A*B.

DOT_PRODUCT DPROD MATMUL

NOTE 13.3b

The unit expression UNITLESS*UNITLESS is also UNITLESS. Therefore the above functions can be applied to UNITLESS arguments.

Where the arguments of MOD or MODULO are real they have different abstract units, and the units of the result are those of the first argument.

The argument of RANDOM_NUMBER has an abstract unit.

The argument of the SQRT function has an abstract unit A and the unit of the result is A**(1/2).

Where the first argument of RATIONAL_POWER is real, that argument has an abstract unit A, and the unit of the result is A**(N/D), where N and D are the values of the second and third arguments.

NOTE 13.3c

The unit expression UNITLESS**(N/D) is also UNITLESS. Therefore the SQRT or RATIONAL_POWER function can be applied to a UNITLESS actual argument and produce a UNITLESS result.

The unit of the result of the UNITLESS intrinsic function is the intrinsic unit UNITLESS.

[322-323] Editor: Insert items into Table 13.1:

RATIONAL_POWER	(X, N, D)	E	X ** (N / D) where N/D is a rational number
UNITLESS	(A)	E	The argument with UNITLESS units

[381:16+ 13.7.137p6+] Editor: Insert a subclause

13.7.137a RATIONAL_POWER (X, N, D)

Description. Raise a number to a rational exponent.

Class. Elemental function.

Arguments.

X shall be of numeric type. If X is of type real the units of X are abstract; otherwise X has no units.

N — shall be of integer type. If X is real and the units of the actual argument associated with X are not the intrinsic unit UNITLESS, then the actual argument associated with N shall be a constant expression.

D — shall be of integer type. If X is real and the units of the actual argument associated with X are not the intrinsic unit UNITLESS, then the actual argument associated with D shall be a constant expression. The value of D shall not be zero.

Result Characteristics. Default real if X is of integer type; otherwise the same type and kind as X. If X is of type real, the units of the result are the units of X raised to the rational power N/D.

NOTE 13.20a

If the units of X are the intrinsic unit UNITLESS, the units of the result are UNITLESS.

Result Value. The result has a value equal to a processor-dependent approximation to X raised to the rational power N/D.

Example. Assume there is an atomic unit LENGTH, a unit VOLUME=LENGTH**3, and a unit AREA=LENGTH**2. The value of the result of RATIONAL_POWER (VOLUME(8.0), 2, 3) is $8.0^{2/3} = 4.0$ (approximately) and the units of the result are $\text{VOLUME}^{2/3}$ = LENGTH**2, which is equivalent to AREA.

[395:26+ 13.7.173p6+] Editor: Insert a subclause

13.7.173a UNITLESS (A)

Description. Value of the argument with units the intrinsic unit UNITLESS.

Class. Elemental function.

Argument. A shall be of real type with any units.

Result Characteristics. Real type and same kind as A. The units of the result are the intrinsic unit UNITLESS.

Result Value. Same as A.

[440:7 16.3.1p1(1)] Editor: After "derived types" insert ", units".

[440:20-21 16.3.1p3] Editor: Replace "or the same as" with ", the same as". After "(4.5.10)" insert ", or the same as the name of a unit (4.4a, 12.4.3.4.5)".

[444:4+ 16.5.1.4p2(1)+] Editor: Insert a list item:

"(1a) a *unit-name* in a *unit-definition-stmt*,"

[461:44+ A.2] Editor: Insert a list item

- " the case of unit names in output (10.7.5a);"

[513:29 C.9.6p24] Editor: Replace "TKR" by "TKUR".

