

Reference number of working document: **ISO/IEC JTC1/SC22/WG5 N1969**

Date: 2013-04-22

Reference number of document: **ISO/IEC TS 99999:2013(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*

© ISO/IEC 2013

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or microfilm, without permission in writing from the publisher. Droits de reproduction réservés. Aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de l'éditeur.

ISO/IEC Copyright Office • Case Postale 56 • CH-1211 Genève • Switzerland

Contents

1	Scope	1
2	Normative References	3
3	Terms and definitions	5
4	Compatibility	7
5	Requirements	9
5.1	General	9
5.2	Summary	9
5.3	UNIT definition statement	11
5.4	Unit name attribute specification statements	16
5.5	Unit attribute specification	16
5.6	UNIT statement	17
5.7	Units in array constructors	17
5.8	Units in expressions	17
5.9	Units in assignments	18
5.10	Units during formatted input/output	18
5.11	Units of dummy and actual arguments	20
5.12	Units of function result variables and function references	21
5.13	Procedure pointer assignment	21
5.14	Units and generic resolution	22
5.15	Intrinsic units	22
5.16	UNITLESS (A)	22
5.17	RATIONAL_POWER (X, N, D)	22
5.18	Units relationship to other intrinsic procedures	23
6	Functionality not requested at this time	25
7	Required editorial changes to ISO/IEC 1539-1:2010(E)	27

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 99999:2013(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 It is the intention of ISO/IEC JTC1/SC22/WG5 that the semantics and syntax specified by this technical specification be included in the next revision of the Fortran International Standard, ISO/IEC 1539-1:2010(E), without change unless experience in the implementation and use of this feature identifies errors that need to be corrected, or changes are needed to achieve proper integration, in which case every reasonable effort will be made to minimize the impact of such changes on existing implementations.

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

¹Not 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).

1 2 Normative References

- 2 1 The following referenced documents are indispensable for the application of this document. For dated
3 references, only the edition cited applies. For undated references, the latest edition of the referenced
4 document (including any amendments) applies.
- 5 2 ISO/IEC 1539-1:2010(E) : *Information technology – Programming languages – Fortran; Part 1: Base*
6 *Language*

1 **3 Terms and definitions**

2 1 For the purposes of this document, the terms and definitions given in ISO/IEC 1539-1:2010(E) and the
3 following, apply.

4 **3.1**

5 1 **unit**

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

8 **3.2**

9 1 **unit family**

10 set of units that can be related by sequences of conversions

1 4 Compatibility

- 2 1 The facility specified by this Technical Specification is compatible with the computational facilities of
3 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
2 to the syntax and semantics of the Fortran programming language to provide units of measure for data
3 entities of real type. The system shall

- 4 • allow to define a system of units upon an arbitrary foundation,
- 5 • check units in expressions, assignment, and procedure references at compile time,
- 6 • distinguish different measures of the same fundamental quantity, and provide for explicit conversion
7 between them,
- 8 • not increase execution time, except where conversion is explicitly requested,
- 9 • not require additional storage,
- 10 • allow to output units, and to check and convert units during input,
- 11 • require minimal labor for its use, and
- 12 • support abstract units, which specify the relationship of units of dummy arguments and func-
13 tion results, and thereby the relationship of corresponding actual arguments and function results,
14 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
3 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
2 measure for a data entity of real type, to define whether it is abstract, and to define whether it is atomic
3 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
2 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
3 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,
2 within expressions.

1 5.2.5 Units in assignments

2 1 This technical specification specifies how units are required to conform in intrinsic assignment and pointer
3 assignment.

4 5.2.6 Units of associate names

5 1 In an ASSOCIATE or SELECT TYPE construct the units of the *associate-name* are the same as the
6 units of the *selector*.

7 5.2.7 Units during formatted input/output

8 1 This technical specification provides a new “U[*w*][.s]” suffix for D, E, EN, ES, F and G format descriptors,
9 to specify the width for names of units during input and output, and the number of blank spaces between
10 the value and the unit name.

11 5.2.8 Units of dummy and actual arguments

12 1 This technical specification provides abstract units, which can be used to specify how units of dummy ar-
13 guments of a procedure conform to each other, and thereby how units of corresponding actual arguments
14 are required to conform to each other, without requiring specific units for the actual arguments.

15 2 This technical specification provides a facility to describe how units of actual arguments are required to
16 conform to nonabstract units of dummy arguments.

17 5.2.9 Units of function results

18 1 This technical specification provides a facility either to specify units of function result variables, or to
19 describe how units of function result variables are related to units of dummy arguments, and thereby
20 how units of function results are related to units of function actual arguments.

21 5.2.10 Units and generic resolution

22 1 This technical specification specifies that nonabstract units are used for generic resolution.

23 5.2.11 Intrinsic units and intrinsic functions related to units

24 1 This technical specification defines intrinsic units RADIANT and UNITLESS. Objects of real type for
25 which units are not specified are assumed to have units of measure UNITLESS.

26 2 This technical specification defines a generic intrinsic function UNITLESS, which has an argument of
27 real type. The result value is the same as the argument value, and the units of the result are UNITLESS.

28 3 This technical specification defines a generic intrinsic function RATIONAL_POWER, used to raise a
29 data entity of real type to a constant rational power, and to specify the units of the function result.

30 5.2.12 Scope and class of names of units

31 1 Names of units are local identifiers of class (1) as defined in subclause 16.3.1 of ISO/IEC 1539-1. They
32 can be accessed by host and use association.

1 5.3 UNIT definition statement

2 5.3.1 Semantics of the UNIT definition statement

3 1 The UNIT definition statement defines the name of a unit, what category of unit it is, if and how it is
4 related to other units, whether addition and subtraction of objects with those units are prohibited, and
5 functions to convert, coerce, and confirm units. Each unit defined by a UNIT definition statement is a
6 different unit, even if it has the same name as a unit defined in a different scoping unit.

7 5.3.2 Categories of units

8 5.3.2.1 General

9 1 A unit is an atomic unit, a composite unit, a conversion unit, or an abstract unit.

10 5.3.2.2 Atomic units

11 1 An atomic unit is one that is not defined in terms of other units.

12 5.3.2.3 Composite units

13 1 A composite unit is one that is defined by an expression using multiplication of units, division of units,
14 or exponentiation of units by an integer or a ratio of integers.

15 5.3.2.4 Conversion units

16 1 A conversion unit is one that is defined in terms of one other unit by a linear transformation having
17 constant numeric coefficients.

18 5.3.2.5 Abstract units

19 1 Abstract units specify the relationship between units of dummy arguments, or the dependence of func-
20 tion result variable units upon dummy argument units, and thereby the relationship between units of
21 corresponding actual arguments, or between actual arguments and function results, without requiring
22 specific units for actual arguments.

23 2 Abstract units are explicitly declared to be abstract. They can be atomic, or they can be composite
24 provided they are defined only in terms of abstract units.

25 5.3.3 Syntax of UNIT definition statement

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

27 R502 *unit-attr* is ABSTRACT
28 or EXCLUDE_ARITHMETIC
29 or *access-spec*

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

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

32 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.

- 1 R504 *unit-expr* is *unit-conversion-expr*
 2 or *unit-composition-expr*
- 3 R505 *unit-conversion-expr* is *mult-operand* * *unit-name* [*mult-op mult-operand*] ■
 4 ■ [*add-op add-operand*]
 5 or [*mult-operand* *] *unit-name mult-op mult-operand* ■
 6 ■ [*add-op add-operand*]
 7 or *unit-name add-op add-operand*
 8 or [*mult-operand* *] (*unit-conversion-expr*) ■
 9 ■ [*mult-op mult-operand*] [*add-op add-operand*]
- 10 C503 (R505) The *unit-name* shall be defined previously within the same scoping unit, or accessible
 11 by use or host association. It shall not be the name of a conversion unit or abstract unit, or
 12 specify the intrinsic unit UNITLESS.
- 13 C504 (R505) Each *add-operand* and *mult-operand* shall be a unitless constant expression of real or
 14 integer type.
- 15 C505 (R505) The *unit-name* shall not have the EXCLUDE_ARITHMETIC attribute.
- 16 C506 (R505) After substitution of values for named constants, and algebraic simplification, *unit-*
 17 *conversion-expr* is always of the form $a \times \textit{unit-name} + b$ where *a* and *b* are unitless numeric con-
 18 stants. If no *mult-operand* appears the value of *a* is 1. If no *add-operand* appears, the value of
 19 *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.

- 20 R506 *unit-composition-expr* is *unit-factor* [*mult-op unit-composition-expr*]
- 21 R507 *unit-factor* is *unit-factor* ** *int-literal-constant*
 22 or *unit-factor* ** (*signed-int-literal-constant*)
 23 or RATIONAL_POWER (*unit-factor*, ■
 24 ■ *signed-int-literal-constant*, *int-literal-constant*)
 25 or *unit-name*
 26 or (*unit-composition-expr*)
- 27 C507 (R506) Either *mult-op* or *int-literal-constant* shall appear.
- 28 C508 (R507) The *unit-name* shall be defined previously within the same scoping unit, or accessible
 29 by use or host association.
- 30 C509 (R507) The *unit-name* shall be the name of an abstract unit if and only if the unit name being
 31 defined by the UNIT statement is an abstract unit.
- 32 C510 (R507) If *unit-name* has the EXCLUDE_ARITHMETIC attribute, the unit being defined shall
 33 have the EXCLUDE_ARITHMETIC attribute.
- 34 C511 (R507) An *int-literal-constant* shall be zero.

- 1 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
 2 fraction, not Fortran integer division.
- 3 2 A unit name is abstract if and only if ABSTRACT appears in the *unit-definition-stmt* that defines it,
 4 or the ABSTRACT attribute is specified for it by an ABSTRACT statement.

5 5.3.4 Excluded operations

- 6 1 It does not make sense to perform arithmetic operations such as addition, or multiplication by unitless
 7 quantities, for objects with certain units, for example, decibels. Addition, subtraction, and multiplication
 8 or division by an unitless quantity, are prohibited for objects for which the EXCLUDE_ARITHMETIC
 9 *unit-attr* is specified.

10 5.3.5 Equivalence of units

- 11 1 The atomic form of an atomic unit is the unit name. The atomic form of a conversion unit is its unit
 12 name.
- 13 2 An atomic unit is not equivalent to any other unit. A conversion unit is equivalent only to its accessible
 14 synonyms.

NOTE 5.4

A unit might have different names in different scoping units.

- 15 3 The atomic form of a composite unit is produced by replacing each composite unit in its defining
 16 expression by that unit's atomic form. If a unit appears more than once the several appearances are
 17 combined into one by adding their exponents, in the usual manner of algebraic simplification. This
 18 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
 19 form, each a_i is an atomic unit or a conversion unit, no two units in the expression are equivalent, and
 20 each e_i is either an integer or the ratio of two integers that have no common factors.
- 21 4 This process terminates because the units that appear in a unit definition expression are required to be
 22 previously defined or accessible by use or host association.
- 23 C512 In the atomic form of a composite unit, the value of e_i shall not be zero.
- 24 C513 In the atomic form of a composite unit, the sum of the exponents of *unit-names* in any single
 25 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.

- 26 5 Although each unit definition defines a different unit, composite units can be equivalent. Composite
 27 units, whether abstract or not, are equivalent if and only if their atomic forms are equivalent. If the
 28 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
 29 $m = n$, there is a one-to-one correspondence between a_i and b_j such that a_i and b_j are equivalent, and
 30 $e_i = e_j$ where a_i and b_j are equivalent. The unit names a_i and b_j are equivalent if they refer to the
 31 same unit definition, even if their names in a scoping unit are different as a result of renaming during
 32 use association.

1 5.3.6 Unit families

- 2 1 Among every set of conversion units that are related by *unit-definitions* in which every *unit-expr* is a
3 *unit-conversion-expr*, there is exactly one unit that is not a conversion unit. Let that unit be called Z.
- 4 2 There is a sequence of conversions between units $W \dots Z$ if there is a set of definitions of conver-
5 sion units $W = f_{WX}(X)$, $X = f_{XY}(Y)$, \dots $Y = f_{YZ}(Z)$, where each f is a mathematical function
6 defined by the *unit-conversion-expr*. Denote the composition of these definitions by $W = f_{WZ}(Z)$.
7 The conversion relation between w and z , where w and z are values with units W and Z , respectively,
8 is $w = f_{WX}(f_{XY}(\dots f_{YZ}(z)\dots)) = f_{WZ}(z)$. Since each *unit-conversion-expr* is linear and the *mult-*
9 *operand* is nonzero, there are also inverse conversion relations $Z = f_{ZY}(Y) = f_{YZ}^{-1}(Y)$, \dots $X = f_{XW}(W)$
10 $= f_{WX}^{-1}(W)$, and a sequence of conversions $z = f_{YZ}(\dots f_{YX}(f_{XW}(w))\dots) = f_{ZW}(w) = f_{WZ}^{-1}(w)$. These
11 compositions are always linear.
- 12 3 If there is a unit A such that there is a sequence of conversions f_{AZ} between Z and A but no direct
13 sequence of conversions f_{AW} between W and A, there is nonetheless a sequence of conversions between
14 w and a where a has units A defined by $a = f_{AZ}(f_{ZW}(w))$. If A and W both depend upon some
15 intermediate unit $I \neq Z$, this sequence can be simplified since within it there is the identity conversion
16 consisting of the sequence $f_{IZ}(f_{ZI}(z)) = f_{IZ}(f_{IZ}^{-1}(z)) = z$
- 17 4 Units that can be related by sequences of conversions constitute a **unit family**.
- 18 5 The atomic form of a composite unit can be represented by a tree in which every internal vertex represents
19 an operator and every terminal vertex represents a unit or a constant, and the relationship of internal
20 vertices is consistent with the hierarchy of operator precedences established by the *unit-composition-exprs*
21 that define the unit and the composite units in its *unit-composition-expr*. A tree that is isomorphic can
22 be constructed by exchanging the subtrees of multiplication operators because multiplication of units is
23 commutative.
- 24 6 If all conversion units in the tree that represents the atomic form of one composite unit can be put
25 into one-to-one correspondence with all the conversion units in a tree that is isomorphic to the one
26 that represents the atomic form of another composite unit, such that the units in every such pair of
27 corresponding units are in the same unit family, then there is a conversion between the composite units,
28 which is constructed by applying the sequence of conversions between members of each corresponding
29 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.

30 5.3.7 Unit conversion functions

- 31 1 A unit family defines a set of pure elemental generic unit conversion functions. For each unit, there is
32 a generic function having the same name as the local name of the unit, with specific functions having a
33 real argument with every kind supported by the processor, and every unit in the family. The result of
34 each function has real type of the same kind as its dummy argument, and units specified by its generic
35 name. Each specific function implements a conversion or sequence of conversions between a value having

1 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.

2 5.3.8 Unit coercion and confirmation functions

3 1 Definition of a unit defines two pure elemental specific functions in the generic interface having the same
4 name as the unit, for each kind of real supported by the processor. Each has a dummy argument of real
5 type. The result value of each is of real type, has the same kind and value as the dummy argument, and
6 units that are the same as the function name.

7 2 The first is a units coercion function. Its dummy argument shall have UNITLESS units.

8 3 The second is a units confirmation function. Its dummy argument shall have units equivalent to the
9 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.

1 5.4 Unit name attribute specification statements

2 5.4.1 ABSTRACT statement

3 1 The ABSTRACT statement specifies the ABSTRACT attribute for a unit name.

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

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

6 5.4.2 EXCLUDE ARITHMETIC statement

7 1 The EXCLUDE ARITHMETIC statement specifies the EXCLUDE_ARITHMETIC attribute for the
8 specified units.

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

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

11 5.4.3 Unit names in accessibility statements

12 1 An accessibility statement may specify the accessibility attribute for a unit name.

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

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

17 5.5 Unit attribute specification

18 1 A *unit-attr-spec* may appear in a *component-attr-spec-list* in a *data-component-def-stmt* or an *attr-spec-*
19 *list* in a *type-declaration-stmt*. It specifies the units for a variable, structure component, or named
20 constant of real type. If a unit attribute is not specified for a variable, structure component, or named
21 constant of real type, the units of that entity are the intrinsic unit UNITLESS.

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

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

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

27 C519 (R510) If *unit-name* is abstract and composite, all abstract units in its atomic form shall be
28 abstract units of nonoptional dummy arguments of the same procedure as for the entity being
29 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.

30 C520 (R510) If *unit-name* is abstract, atomic, and is the unit of a function result, that unit shall
31 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.

1 **5.6 UNIT statement**

2 1 A unit specification statement specifies the units for a list of variables or named constants of real type.

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

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

6 **5.7 Units in array constructors**

7 1 Within an *array-constructor* of real type, every *expr* shall have equivalent units. See 5.3.5.

8 **5.8 Units in expressions**

9 1 The form of expressions is defined in Subclause 7.1.2 of ISO/IEC 1539-1:2010(E).

10 2 If a *mult-operand* includes a *power-op* the units of the second *mult-operand* of the *power-op* shall be
11 UNITLESS. If the units of the *level-1-expr* of the *power-op* are UNITLESS the units of the *mult-operand*
12 are UNITLESS. Otherwise the second *mult-operand* shall be a constant of type integer, and the units
13 expression of the *mult-operand* is formed by multiplying the exponents of every factor of the units
14 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=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)

```

15 3 If an *add-operand* includes a *mult-op* that is * the units expression of the *add-operand* is formed by
16 multiplying the units expression of the first *add-operand* of the *mult-op* by the units expression of the
17 second *mult-operand*. If one of the operands has UNITLESS units the units of the result are the units
18 of the other operand.

19 4 If an *add-operand* includes a *mult-op* that is / the units expression of the *add-operand* is formed by
20 dividing the units expression of the first *add-operand* of the *mult-op* by the units expression of the
21 second *mult-operand*. If the second operand has UNITLESS units the units of the result are the units
22 of the first operand. If the first operand has UNITLESS units the units of the result are the inverse of
23 the units of the second operand.

NOTE 5.12

```

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

```

- 1 5 If a *level-2-expr* consists of *add-operand* or *add-op add-operand*, the units of the *level-2-expr* are the
 2 units of the *add-operand*.
- 3 6 If a *level-2-expr* includes an *add-op* with two operands and if both operands of the *add-op* are of real
 4 type, the operands shall have equivalent units (5.3.5) and neither unit shall have the EXCLUDE_-
 5 ARITHMETIC attribute. If only one operand is of real type, it shall have UNITLESS units. The units
 6 of the result of the operation are the units of the operand of real type.
- 7 7 If a *level-2-expr* includes a *mult-op* and the unit of one of the operands has the EXCLUDE_ARITH-
 8 METIC attribute, the other operand shall not be unitless.

NOTE 5.13

<code>X + Y</code>	<code>! prohibited</code>
<code>Z + A(UNITLESS(Y))</code>	<code>! has units A because units of Y are coerced to A</code>

- 9 8 If both operands of *level-4-expr* are of type real they shall have equivalent units. If only one operand is
 10 of type real its units shall be the intrinsic unit UNITLESS.

5.9 Units in assignments

- 12 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.

- 13 2 If only one of the *variable* or *expr* in an intrinsic assignment is of real type it shall have UNITLESS
 14 units.
- 15 3 If the *data-pointer-object* and *data-target* in a pointer assignment statement are of real type, they shall
 16 have equivalent units (5.3.5).

5.10 Units during formatted input/output**5.10.1 Unit name format descriptors**

- 19 1 A unit name edit descriptor suffix `U[w][.s]` is provided for D, E, EN, ES, F, or G edit descriptors.
- 20 2 If a D, E, EN, ES, F, or G edit descriptor has a U format descriptor suffix, and it corresponds to an
 21 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

- 23 1 If an effective list item is of real type, its units expression shall consist of *unit-name*.

NOTE 5.15

<code>unit :: LENGTH, AREA = LENGTH**2</code>	
<code>real, unit(length) :: X</code>	
<code>real, unit(area) :: Y</code>	
<code>write (*, *) X*X</code>	<code>! prohibited</code>

NOTE 5.15 (cont.)

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

- 1 2 If an output list item appears in an output statement in which a *format* other than asterisk appears,
 2 and it corresponds to an edit descriptor that has a U[w][.s] suffix, the value of *w* to be used to output
 3 the unit name is specified by that suffix. The width of the field is *w* if *w* appears, and the number of
 4 characters in the unit name otherwise. The value of *w* shall not be zero. If *.s* does not appear, the unit
 5 name shall be separated from the value by one blank. If *.s* appears, the unit name shall be separated
 6 from the value by *s* blanks. The value of *s* shall not be negative. The unit name shall be output if and
 7 only if the list item corresponds to an edit descriptor that has a U[w][.s] suffix. A U[w][.s] edit descriptor
 8 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).

- 9 3 Unit name editing is the same as for character editing using an A[w] edit descriptor. The case of unit
 10 names in output is processor dependent.
- 11 4 For list-directed or namelist output, if the units of a real output item are the intrinsic unit UNITLESS,
 12 the unit name shall not be output. Otherwise the unit name is output after the value, separated from
 13 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

- 15 1 In a formatted READ statement in which a *format* other than asterisk appears, if a real input list item
 16 has units other than the intrinsic unit UNITLESS, the edit descriptor shall have a U[w][.s] suffix, and a
 17 unit name shall appear in the field of width *w*. If *.s* does not appear, the unit name shall be separated
 18 from the value by one blank. If *.s* appears the unit name shall be separated from the value by *s* blanks.
 19 The value of *s* shall not be negative. Unit name editing is the same as for character editing using an
 20 A[w] edit descriptor. If *w* appears it shall not be zero.
- 21 2 If a scalar of real type is input using list-directed or namelist input, and its units are not the intrinsic
 22 unit UNITLESS, a unit name shall follow the value on the same line and be separated from it by one or
 23 more spaces. If the units of the input item are UNITLESS, a unit name shall not appear.
- 24 3 If an array of real type is input using list-directed or namelist input, and its units are not the intrinsic
 25 unit UNITLESS, a unit name shall follow the first value and be separated from it by one or more blank
 26 spaces. Whether a unit name follows other elements is optional. If a unit name is not specified for the
 27 value of an array element, its units are the same as for the previous element. If the units of the array
 28 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.

- 29 4 The input unit name shall be the same as the declared units of the input list item, or shall be in the
 30 same unit family, without regard to case. If the input unit name is not the same as the units of the input
 31 list item, the sequence of conversions between the input unit and declared unit is applied to convert the
 32 input value to the units of the input list item. If the input unit name is not the same as the units of

- 1 the input list item and not in the same unit family as the units of the input list item, an error condition
 2 exists.
- 3 5 A unit other than the intrinsic unit UNITLESS can be renamed during USE association. The unit name
 4 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 3×10^8 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 5.11 Units of dummy and actual arguments

6 5.11.1 Characteristics, explicit interface

- 7 1 The units of dummy arguments, and if they are abstract the relationship of those units, are characteristics
 8 of the procedure.
- 9 2 If a procedure has a dummy argument that has a unit other than UNITLESS, its interface shall be explicit
 10 in a scoping unit where it is referenced, appears as an actual argument, or appears as a procedure pointer
 11 target.

12 5.11.2 Abstract units

- 13 1 Abstract units may be specified for dummy arguments. Abstract units may be atomic or may be
 14 composite units composed from other atomic units.
- 15 2 If the unit of a dummy argument or function result is abstract and composite, the atomic form of that
 16 unit shall consist only of abstract units of dummy arguments of the same procedure, or a function result
 17 variable of the same procedure.

18 5.11.3 Actual arguments corresponding to dummy arguments with nonabstract units

- 19 1 An actual argument that corresponds to a dummy argument that has nonabstract units shall have units
 20 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.

1 5.11.4 Actual arguments corresponding to dummy arguments with abstract units

- 2 1 An actual argument that corresponds to a dummy argument that has an atomic abstract unit need not
3 have the same units as the dummy argument.
- 4 2 If an actual argument corresponds to a dummy argument that has abstract units, its units shall be
5 related to the units of other actual arguments in the same way that the units of their corresponding
6 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.

7 5.12 Units of function result variables and function references

- 8 1 The units of function results, and if they are abstract their relationship to units of dummy arguments,
9 are a characteristic of the procedure.
- 10 2 Function result variables can have abstract or nonabstract units.
- 11 3 If the units of the function result variable are nonabstract, the units of the function reference are the
12 same as the units of the function result variable.
- 13 4 If the unit of the function result variable is an abstract unit, the atomic form of that unit shall consist
14 only of abstract units of dummy arguments of the same procedure.
- 15 5 If the unit of the function result variable is abstract the units of the result of a function reference are
16 related to the units of the actual arguments in the same way that the units of the result variable are
17 related to the units of the dummy arguments.
- 18 6 If the units of the function result variable are not the intrinsic unit UNITLESS, the interface of the
19 function shall be explicit in a scoping unit where the function name appears.

20 5.13 Procedure pointer assignment

- 21 1 A procedure pointer that has implicit interface shall not have a target that has a dummy argument that
22 has a unit other than UNITLESS. A function procedure pointer shall not have a target that has result
23 units different from the result units of the pointer.

1 5.14 Units and generic resolution

2 1 If all of the dummy arguments of some pair of specific interfaces in a generic interface have the same
3 type, kind and rank, at least one of the dummy arguments of one of the specific procedures shall have
4 units that are not equivalent to the units of the corresponding dummy argument in the other specific
5 procedure, and those two dummy arguments shall not both have abstract units.

6 2 If the characteristics of two specific procedures in a generic interface are identical except that the units
7 of some dummy arguments of the first are not abstract while the units of corresponding units of the
8 second are abstract, the first procedure shall not have any arguments with abstract units.

9 3 If the characteristics of two specific procedures in a generic interface are identical except for the units
10 of some dummy arguments, and those dummy arguments of the first procedure have abstract units, and
11 the corresponding actual arguments of the second have the same units as the dummy argument of the
12 second procedure, the second procedure is invoked.

13 5.15 Intrinsic units

14 1 This technical specification defines an intrinsic atomic unit UNITLESS, an intrinsic units coercion func-
15 tion UNITLESS that coerces an actual argument with any units to a result with UNITLESS units, and
16 an intrinsic units confirmation function UNITLESS.

17 2 This technical specification defines an intrinsic atomic unit RADIAN. The intrinsic unit RADIAN defines
18 unit coercion and confirmation functions as described in subclause 5.3.8 of this technical specification.

19 5.16 UNITLESS (A)

20 1 **Description.** Value of the argument with units UNITLESS.

21 2 **Class.** Elemental function.

22 3 **Argument.** A shall be of real type with any units.

23 4 **Result Characteristics.** Same type and kind as A with units UNITLESS.

24 5 **Result Value.** Same as A.

25 5.17 RATIONAL_POWER (X, N, D)

26 1 **Description.** Raise a number to a rational exponent.

27 2 **Class.** Elemental function.

28 3 **Arguments.**

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

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

34 D shall be of integer type. If X is real and the units of the actual argument associated with
35 X are not the intrinsic unit UNITLESS then the actual argument associated with D shall
36 be a constant expression.

37 4 **Result Characteristics.** Default real if X is of integer type; otherwise the same type and kind as X.

1 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.

2 5 **Result Value.** The result has a value equal to a processor-dependent approximation to X raised to the
3 rational power N/D.

4 6 **Example.** Assume there is an atomic unit LENGTH, a unit VOLUME=LENGTH**3, and a unit
5 AREA=LENGTH**2. The value of the result of RATIONAL_POWER (VOLUME(8.0), 2, 3) is
6 $8.0^{2/3} = 4.0$ (approximately) and the units of the result are $VOLUME^{2/3} = LENGTH^{**2}$, which is
7 equivalent to AREA.

8 **5.18 Units relationship to other intrinsic procedures**

9 1 If an intrinsic function has any arguments of real type but the result is not of real type, the units of the
10 arguments are the intrinsic unit UNITLESS. If an intrinsic function has a real result but no arguments
11 of real type, the units of the result are the intrinsic unit UNITLESS.

12 2 With the following exceptions, the units of real arguments of an intrinsic function with real result are
13 the same abstract unit, and the units of the result are that abstract unit.

14 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		

15 4 The arguments of ATAN2, or ATAN with two arguments, have the same abstract unit. The result of
16 ATAN2 or ATAN with two arguments has the intrinsic unit UNITLESS.

17 5 Trigonometric functions have arguments with either UNITLESS or RADIAN units, and UNITLESS
18 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.

19 6 Where the following intrinsic functions have real arguments, their arguments have atomic abstract units
20 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.

21 7 Where the arguments of MOD or MODULO are real they have different abstract units, and the units

- 1 of the result are those of the first argument.
- 2 8 The argument of RANDOM_NUMBER has an abstract unit.
- 3 9 The argument of the SQRT function has an abstract unit A and the unit of the result is $A^{1/2}$.
- 4 10 Where the first argument of RATIONAL_POWER is real, that argument has an abstract unit A, and
5 the unit of the result is $A^{N/D}$, where N and D are the values of the second and third arguments.

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.

- 6 11 The unit of the result of the UNITLESS intrinsic function is the intrinsic unit UNITLESS.

6 Functionality not requested at this time

- 1
2 1 If the type parameter system were extended to units, it would be possible for components to have
3 abstract units. Where objects are declared, their unit parameter values would have to be related to
4 each other in the same way that abstract units of corresponding type parameters are related, similar to
5 the requirement in this technical specification concerning the relationship between actual and dummy
6 arguments. Where unit parameters in a type definition are not abstract, objects' corresponding unit
7 parameter values would have to match exactly.
- 8 2 This would allow a procedure to have a dummy argument of a derived type that has abstract unit
9 parameters, with the unit parameter values of the dummy argument being abstract. For example, in a
10 procedure for solution of differential equations, a dummy argument might have parameters that specify
11 abstract units, say state and independent variable. Users of the library procedure would have their own
12 units, say meters and seconds.
- 13 3 Components could have composite abstract units, such as derivative = state / independent variable. It
14 would be desirable to allow to define abstract units, especially composite abstract units, within a type
15 definition.
- 16 4 It is possible in principle to allow entities of complex type to have units. The real and imaginary parts
17 could have independent units. The descriptions would be complicated.
- 18 5 It might be possible in principle to allow entities of integer type to have units. The definition of the
19 units of results of expressions involving integer division would be controversial.
- 20 6 It is unlikely to be useful to allow entities of derived type to have units. It is sufficient that their
21 components have units, especially if those units can be type parameters.
- 22 7 It does not seem to be possible, even in principle, to allow entities of logical or character type to have
23 units.
- 24 8 Unit conversions could be performed during argument association, implying copy-in/copy-out argument
25 passing. Additional changes to the rules for distinguishing generic procedures would be necessary .

1 statement defines the name of a unit, what category of unit it is, if and how it is related to other units,
 2 and units conversion, coercion, and confirmation functions. Each unit defined by a UNIT definition
 3 statement is a different unit, even if it has the same name as a unit defined in a different scoping unit.

4 4.4a.2 Categories of units

5 4.4a.2.1 General

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

7 4.4a.2.2 Atomic units

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

9 4.4a.2.3 Composite units

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

12 4.4a.2.4 Conversion units

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

15 4.4a.2.5 Abstract units

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

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

22 4.4a.3 Syntax of UNIT definition statement

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

24 R424b *unit-attr* is ABSTRACT
 25 or EXCLUDE_ARITHMETIC
 26 or *access-spec*

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

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

Unresolved Technical Issue C425b

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

30 C425b (R424b) A *unit-attr* shall not be specified more than once for a unit name, no matter how
 31 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.

- 1 R424d *unit-expr* is *unit-conversion-expr*
 2 or *unit-composition-expr*
- 3 R424e *unit-conversion-expr* is *mult-operand* * *unit-name* [*mult-op mult-operand*] ■
 4 ■ [*add-op add-operand*]
 5 or [*mult-operand* *] *unit-name mult-op mult-operand* ■
 6 ■ [*add-op add-operand*]
 7 or *unit-name add-op add-operand*
 8 or [*mult-operand* *] (*unit-conversion-expr*) ■
 9 ■ [*mult-op mult-operand*] [*add-op add-operand*]
- 10 C425c (R424e) The *unit-name* shall be defined previously within the same scoping unit, or accessible
 11 by use or host association. It shall not be the name of a conversion unit or abstract unit, or
 12 specify the intrinsic unit UNITLESS.
- 13 C425d (R424e) Each *add-operand* and *mult-operand* shall be a unitless constant expression of real or
 14 integer type.
- 15 C425e (R424e) The *unit-name* shall not have the EXCLUDE_ARITHMETIC attribute.
- 16 C425f (R424e) After substitution of values for named constants, and algebraic simplification, *unit-*
 17 *conversion-expr* is always of the form $a \times \textit{unit-name} + b$ where a and b are unitless numeric con-
 18 stants. If no *mult-operand* appears the value of a is 1. If no *add-operand* appears, the value of
 19 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.

- 20 R424f *unit-composition-expr* is *unit-factor* [*mult-op unit-composition-expr*]
- 21 R424g *unit-factor* is *unit-factor* ** *int-literal-constant*
 22 or *unit-factor* ** (*signed-int-literal-constant*)
 23 or RATIONAL_POWER (*unit-factor*, ■
 24 ■ *signed-int-literal-constant*, *int-literal-constant*)
 25 or *unit-name*
 26 or (*unit-composition-expr*)
- 27 C425h (R424f) Either *mult-op* or *int-literal-constant* shall appear.
- 28 C425i (R424g) The *unit-name* shall be defined previously within the same scoping unit, or accessible
 29 by use or host association.
- 30 C425j (R424g) The *unit-name* shall be the name of an abstract unit if and only if the *unit-name* being
 31 defined by the *unit-definition-stmt* is an abstract unit.
- 32 C425k (R424g) If *unit-name* has the EXCLUDE_ARITHMETIC attribute, the unit being defined shall
 33 have the EXCLUDE_ARITHMETIC attribute.
- 34 C425l (424h) Neither *int-literal-constant* shall be zero.
- 35 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

1 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,
3 or the ABSTRACT attribute is specified for it by an ABSTRACT statement.

4 4.4a.4 Excluded operations

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

9 4.4a.5 Equivalence of units

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

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

NOTE 4.14d

A unit might have different names in different scoping units.

14 The atomic form of a composite unit is produced by replacing each composite unit in its defining
15 expression by that unit's atomic form. If a unit appears more than once the several appearances are
16 combined into one by adding their exponents, in the usual manner of algebraic simplification. This
17 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
18 a_i is an atomic unit or a conversion unit, no two units in the expression are equivalent, and each e_i is
19 either an integer or the ratio of two integers that have no common factors.

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

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

23 C425n In the atomic form of a composite unit, if several a_i are conversion units in the same family
24 (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.

25 Although each unit definition defines a different unit, composite units can be equivalent. Composite
26 units, whether abstract or not, are equivalent if and only if their atomic forms are equivalent. If the
27 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
28 $m = n$, there is a one-to-one correspondence between a_i and b_j such that a_i and b_j are equivalent, and
29 $e_i = e_j$ where a_i and b_j are equivalent. The unit names a_i and b_j are equivalent if they refer to the same
30 unit definition.

31 4.4a.6 Unit families

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

3 There is a sequence of conversions between units W ... Z if there is a set of definitions of conver-
4 sion units $W = f_{WX}(X)$, $X = f_{XY}(Y)$, ... $Y = f_{YZ}(Z)$, where each f is a mathematical function
5 defined by the *unit-conversion-expr*. Denote the composition of these definitions by $W = f_{WZ}(Z)$.
6 The conversion relation between w and z , where w and z are values with units W and Z, respectively,
7 is $w = f_{WX}(f_{XY}(\dots f_{YZ}(z)\dots)) = f_{WZ}(z)$. Since each *unit-conversion-expr* is linear and the *mult-*
8 *operand* is nonzero, there are also inverse conversion relations $Z = f_{ZY}(Y) = f_{YZ}^{-1}(Y)$, ... $X = f_{XW}(W)$
9 $= f_{WX}^{-1}(W)$, and a sequence of conversions $z = f_{YZ}(\dots f_{YX}(f_{XW}(w))) = f_{ZW}(w) = f_{WZ}^{-1}(w)$.

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

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

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

22 If all conversion units in the tree that represents the atomic form of one composite unit can be put
23 into one-to-one correspondence with all the conversion units in a tree that is isomorphic to the one
24 that represents the atomic form of another composite unit, such that the units in every such pair of
25 corresponding units are in the same unit family, then there is a conversion between the composite units
26 that is constructed by applying the sequence of conversions between members of each corresponding
27 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.

28 4.4a.7 Unit conversion functions

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

1 4.4a.8 Unit coercion and confirmation functions

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

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

7 The second is a **units confirmation function**. Its dummy argument shall have units equivalent to the
8 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.

9 4.4a.9 ABSTRACT statement

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

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

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

3 4.4a.10 EXCLUDE ARITHMETIC statement

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

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

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

8 4.4a.11 Intrinsic units

9 This part of ISO/IEC 1539 defines an intrinsic atomic unit UNITLESS, an intrinsic units coercion
10 function UNITLESS that coerces an actual argument with any units to a result with UNITLESS units,
11 and an intrinsic units confirmation function UNITLESS. This part of ISO/IEC 1539 defines an intrinsic
12 atomic unit RADIAN. The intrinsic unit RADIAN defines unit coercion and unit confirmation functions
13 (4.4a.8).

14 [66:13+ R437] Editor: Add an alternative for R439 *component-attr-spec*:

15 *or unit-attr-spec*

16 [69:20 4.5.4.6p2] Editor: After first “target” insert “, if the pointer variable or component is of real type
17 its units are equivalent to the units of *initial-data-target* (4.4a.5)”

18 [88:10+ C505+] Editor: Insert a constraint:

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

21 [101:19- Note 5.24+] Editor: Insert a subclause:

22 5.3.17a UNIT attribute

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

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

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

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

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

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

1 [102:28+ R525] Editor: Add an alternative for R525 *access-stmt*:

2
$$\text{or } \textit{unit-name}$$

3 [102:32+ C563+] Editor: Add a constraint:

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

6 [108:18- Note 5.36+] Editor: Insert a subclause:

7 **5.4.15a UNIT statement**

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

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

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

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

14 [134:18+ R709+] Editor: Insert constraints and paragraphs:

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

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

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

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

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

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

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

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

- 1 UNITLESS the units of the result are the inverse of the units of the second operand.
- 2 If a *level-2-expr* consists of *add-operand* or *add-op add-operand*, the units of the *level-2-expr* are the
3 units of the *add-operand*.
- 4 If a *level-2-expr* consists of *level-2-expr add-op add-operand*, the units of the result are the units of the
5 operand of real type.
- 6 C703c (R704) The atomic form of the units expression of a *mult-operand* shall satisfy the requirements
7 specified in 4.4a.5.

- 8 [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

```

- 9 [135:19+ R713+] Editor: Insert a constraint:
- 10 C703d (R712) If both operands of *level-4-expr* are of real type they shall have equivalent units. If only
11 one operand is of real type its units shall be the intrinsic unit UNITLESS.
-
- 12 [153:21,24,24+ 7.2.12p1(7-8)] Editor: Delete “and” at the end of item (7); insert list items:
- 13 (9) if the *variable* and *expr* in an intrinsic assignment are of real type they shall have equivalent
14 units (4.4a.5), and
- 15 (10) if only one of the *variable* or *expr* in an intrinsic assignment is of real type its units shall be
16 the intrinsic unit UNITLESS.
-
- 17 [157:36+ C714+] Editor: Insert a constraint:
- 18 C714a (R733) If *data-target* is of real type, the units of *data-pointer-object* and *data-target* shall be
19 equivalent (4.4a.5)
-
- 20 [160:13+ 7.2.2.4p5+] Editor: Insert a paragraph:
- 21 If the pointer object has an implicit interface, *proc-target* shall not be a procedure that has a dummy
22 argument or result with units other than the intrinsic unit UNITLESS.
-
- 23 [160:15 7.2.2.4p6] Editor: Append a sentence to the paragraph:

1 “The units of the result of the pointer object shall not be different from the units of the result of the
2 pointer target.”

3 [170:33 8.1.3.2p1] Editor: Before “If” insert a sentence: “If the selector is of real type the units of the
4 associating entity are assumed from the units of the selector.”

5 [217:16+ C935+] Editor: Insert a constraint:

6 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
```

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

9 **or** $Fw.d[U[w][.s]]$
10 **or** $Ew.d[Ee][U[w][.s]]$
11 **or** $ENw.d[Ee][U[w][.s]]$
12 **or** $ESw.d[Ee][U[w][.s]]$
13 **or** $Gw.d[Ee][U[w][.s]]$
14 **or** $Dw.d[U[w][.s]]$
15

16 [247:15+ R1011] Editor: Insert a syntax rule:

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

18 [259:8+ 10.7.5.4+] Editor: Insert a subclause:

19 **10.7.5a Unit name editing**

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

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

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

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

1 [263:14 10.10.3p4] Editor: Append sentences: “If the item is a real scalar and has units other than the
2 intrinsic unit UNITLESS, a unit name shall follow the value, separated from the value by one or more
3 spaces. If the item is a real array and has units other than the intrinsic unit UNITLESS, a unit name
4 shall follow the first value, separated from the value by one or more spaces, and may follow other values
5 provided the unit specified for each value after the first is equivalent to the unit for the previous value
6 or related to it by conversion. If a unit is not specified for the value of an array element the unit of the
7 previous value is used. If the units of the item are the intrinsic unit UNITLESS, a unit name shall not
8 appear. The unit specified in the input shall be related to the unit of the item as specified in 10.7.5a. If
9 the relationship is a conversion, conversion shall be applied as specified in 10.7.5a.”

10 [265:19 10.10.4p5] Editor: Append sentences: “If the item is scalar and has units other than the intrinsic
11 unit UNITLESS, its unit name shall follow the value, separated from the value by one or more spaces
12 and be edited as if by a U edit descriptor without *w*. If the item is an array and has units other than the
13 intrinsic unit UNITLESS, its unit name shall follow the value of the first element of the array, separated
14 from that value by one or more spaces, and edited as if by a U edit descriptor without *w*.”

15 [278:17 12.3.2.2p1] Editor: After “(if any)” insert “its nonabstract units (if any), the relationship of its
16 units to the units of other dummy arguments if its units are abstract”

17 [278:30 12.3.3p1] Editor: After “(if any)” insert “its nonabstract units (if any), the relationship of its
18 units to the units of dummy arguments if its units are abstract”

19 [279:21- 12.4.2.2p1(1)(a-)] Editor: Insert a list item:

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

22 [279:24- 12.4.2.2p1(2)(a-)] Editor: Insert a list item:

23 “(a’ has units other than the intrinsic unit UNITLESS,”

24 [279:31- 12.4.2.2p1(3)(a-)] Editor: Insert a list item:

25 “(a’ has units other than the intrinsic unit UNITLESS,”

26 [279:35+ 12.4.2.2p1+] Editor: Insert a paragraph:

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

30 [285:15-16 12.4.3.4.5p2] Editor: After first “kind” insert “; units”; replace “TKR” by “TKUR”; after
31 “second” insert “, if they are of type real the units of the first are equivalent to the units of the second
32 or one has abstract units”; change the index item to “TKUR compatible”.

33 [286:3 12.4.3.4.5p3 second item] Editor: Replace “TKR” by “TKUR”.

34 [286:18 C1215(1)(a)] Editor: Replace “TKR” by “TKUR”.

35 [286:33+ C1215+] Editor: Insert a constraint:

36 C1215a If the only difference between the characteristics of the dummy arguments of two procedures
37 within the scope of a generic identifier is that one has an argument of real type with nonabstract

1 units while the other has a corresponding argument with abstract units, the first procedure shall
 2 have no arguments with abstract units.

3 [288:24+ 12.4.3.6p9+] Editor: Insert a paragraph:

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

6 [293:14+ 12.5.2.4p4+] Editor: Insert paragraphs and notes:

7 An actual argument that corresponds to a dummy argument that has nonabstract units shall have units
 8 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.

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

11 If an actual argument corresponds to a dummy argument that has abstract units, its units shall be
 12 related to the units of other actual arguments in the same way that the units of their corresponding
 13 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.

14 The units of the dummy argument of a units conversion function shall be in the same units family
 15 (4.4a.6) as the units of the function result. The units of the dummy argument of a units coercion
 16 function (4.4a.8) shall be the intrinsic unit UNITLESS. The units of the dummy argument of a units
 17 confirmation function (4.4a.8) shall be equivalent (4.4a.5) to the units of the function result.

18 [302:6 12.5.3p1] Editor: Before “If” insert a sentence: “If the function result variable has abstract units,
 19 the units of the result value of the function reference are related to the units of the actual arguments
 20 in the same way that the units of the function result variable are related to the units of the dummy
 21 argument.”

22 [303:27,32 12.5.5.2p1,p2] Editor: Replace “one such specific procedure” by “two such specific procedures,
 23 and only one of those shall have arguments with abstract units. If the reference is consistent with the
 24 procedure with arguments with nonabstract units, the reference is to that procedure; otherwise the
 25 reference is to the procedure with arguments with abstract units” twice.

26 [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.

1 [305:21 12.6.2.1p1] Editor: Append a sentence: “A unit definition defines conversion (4.4a.7), coercion
2 and confirmation (4.4a.8) functions of the same name.”

3 [306:35+ C1256+] Editor: Insert a constraint:

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

6 [307:8 12.6.2.2p3] Editor: After “result” insert “has units other than the intrinsic unit UNITLESS, or”

7 [316:29+ 13.2.4p2+] Editor: Insert a subclause

8 13.3a Units of intrinsic function arguments and results

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

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

14 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		

15 The arguments of ATAN2 or ATAN with two arguments have the same abstract unit. The result of
16 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

NOTE 13.3a (cont.)

can coerce the result units to RADIAN.

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

3 Where the following intrinsic functions have real arguments, their arguments have atomic abstract units
4 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.

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

7 The argument of RANDOM_NUMBER has an abstract unit.

8 The argument of the SQRT function has an abstract unit A and the unit of the result is $A^{1/2}$.

9 Where the first argument of RATIONAL_POWER is real, that argument has an abstract unit A, and
10 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.

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

12 [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

13 [381:16+ 13.7.137p6+] Editor: Insert a subclause

13.7.137a RATIONAL_POWER (X, N, D)

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

16 **Class.** Elemental function.

Arguments.

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

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

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

4 **Result Characteristics.** Default real if X is of integer type; otherwise the same type and kind as X.
 5 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.

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

8 **Example.** Assume there is an atomic unit LENGTH, a unit VOLUME=LENGTH**3, and a unit
 9 AREA=LENGTH**2. The value of the result of RATIONAL_POWER (VOLUME(8.0), 2, 3) is
 10 $8.0^{2/3} = 4.0$ (approximately) and the units of the result are $VOLUME^{2/3} = LENGTH^{**2}$, which is
 11 equivalent to AREA.

12 [395:26+ 13.7.173p6+] Editor: Insert a subclause

13 **13.7.173a UNITLESS (A)**

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

15 **Class.** Elemental function.

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

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

19 **Result Value.** Same as A.

20 [440:7 16.3.1p1(1)] Editor: After “derived types” insert “, units”.

21 [444:4+ 16.5.1.4p2(1)+] Editor: Insert a list item:

22 “(1a) a *unit-name* in a *unit-definition-stmt*,”

23 [461:44+ A.2] Editor: Insert a list item

24 • “ the case of unit names in output (10.7.5a);”

25 [513:29 C.9.6p24] Editor: Replace “TKR” by “TKUR”.