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

Date: 2010-2-24

Reference number of document: **ISO/IEC TR 99999:2010(E)**

Committee identification: ISO/IEC JTC1/SC22

Secretariat: ANSI

Information technology — Programming languages — Fortran — Updater procedures

Technologies de l'information — Langages de programmation — Fortran — Procédures pour mettre à jour les structures de données

Contents

Foreword

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.

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

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.

ISO/IEC TR 99999:2010(E) was prepared by Joint Technical Committee ISO/IEC/JTC1, *Information technology*, Subcommittee SC22, *Programming languages, their environments and system software interfaces*.

This technical report 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).

It is the intention of ISO/IEC JTC1/SC22/WG5 that the semantics and syntax specified by this technical report be included in the next revision of the Fortran International Standard 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.

0 Introduction

0.1 History

1 After high-level programming languages had been in use for about a decade, it was realized that programs are difficult to maintain and modify because the details of the implementation of each data structure were exposed in the syntax used to reference the representation of the data.

2 Two fundamentally different solutions were proposed for the problem.

3 In 1970 Douglas T. Ross proposed that the same syntax ought to be used to refer to every kind of data object, and to procedures.

4 Charles M. Geschke and James G. Mitchell repeated this proposal in 1975.

5 In 1972 David Parnas proposed that this could be achieved almost completely by encapsulating all operations on a data structure in a family of related procedures.

6 No major programming language has been revised to incorporate the principles advocated by Ross, Geschke and Mitchell.

7 Rather, it has apparently been judged that the problem can be adequately solved by program authors employing the principles advocated by Parnas.

1. Charles M. Geschke and James G. Mitchell, *On the problem of uniform references to data structures*, **IEEE Transactions on Software Engineering SE-2**, 1 (June 1975) 207-210.
2. David Parnas, *On the criteria to be used in decomposing systems into modules*, **Comm. ACM 15**, 12 (December 1972) 1053-1058.
3. D. T. Ross, *Uniform referents: An essential property for a software engineering language*, in **Software Engineering 1** (J. T. Tou, Ed.), Academic Press, (1970) 91-101.

0.2 The problem to be solved

1 There are two problems with the Parnas agenda.

2 First, it is difficult and costly to apply completely and consistently. If it hasn't been applied carefully and completely during the original development of a program, the program is difficult to modify.

3 Second, it is potentially inefficient, because all operations on data structures are encapsulated within procedures. Awareness of this potential is an incentive not to use it carefully and completely.

0.3 What this report proposes

1 This technical report extends the programming language Fortran so that the representation of a data abstraction can be changed between a data object and a procedure without changing the syntax of any references to it, unless the reference is a character substring.

2 The facility specified by this technical report is compatible to the computational facilities of Fortran as standardized by ISO/IEC 1539-1:2010(E).

Information technology – Programming Languages – Fortran

Technical Report: Accessors

1 General

1.1 Scope

1 This technical report specifies an extension to the programming language Fortran. The Fortran language is specified by International Standard ISO/IEC 1539-1:2010(E) : Fortran. The extension allows the representation of a data object to be changed between an array and a procedure, or between a structure component and a procedure, without changing the syntax of references to that data object, unless the reference is a character substring.

2 Clause 2 of this technical report contains a general and informal but precise description of the extended functionalities. Clause 3 contains detailed instructions for editorial changes to ISO/IEC 1539-1:2010(E).

1.2 Normative References

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

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

2 Requirements

2.1 General

1 The following subclauses contain a general description of the extensions to the syntax and semantics of the Fortran programming language to provide that the representation of a data object can be changed between an array and a procedure, or between a structure component and a procedure, without changing the syntax of references to that data object, unless the reference is a character substring.

2.2 Summary

2.2.1 General

1 This technical report defines a new form of subprogram called an *updater*. An updater subprogram defines an updater procedure. It can be invoked in a variable definition context, in which case the value is passed to its acceptor variable. There is presently nothing comparable in Fortran, but it has been provided in other languages such as Mesa and POP-2 (but not in any widely-used language). This allows the representation of a data abstraction to be changed between function and updater procedures, and a data object, without changing the syntax of references to it, unless the reference is a character substring.

2.2.2 Definition of updater subprograms

1 A new subprogram entity called an UPDATER is defined. An updater declares a subprogram that defines an updater procedure. When an updater is referenced in a variable definition context, the value to be defined is considered to be an actual argument, and is associated with the updater's acceptor variable, which is considered to be a dummy argument.

2 Updater subprograms can be type bound procedures and can be procedure pointer targets.

2.2.3 Syntax of reference to updater procedures

1 A reference to an updater is permitted where a variable is permitted to appear in a variable definition context.

NOTE 2.1

For example, an updater reference can appear as the *variable* in an intrinsic assignment statement, in an input/output list in either a READ or WRITE statement, in place of a *variable* in a control information list. . . .

2 An updater is referenced using the same syntax as a function reference.

2.2.4 Accessor generic interface

1 A generic interface that specifies both functions and updaters is an accessor generic interface.

2 Where an accessor reference appears in a value reference context a function is invoked to produce a value. Where it appears in a variable definition context an updater is invoked to accept a value. Where it appears as an actual argument associated with a dummy argument with INTENT(IN), a function is invoked to produce a value before the procedure to which it is an actual argument is invoked. Where it appears as an actual argument associated with a dummy argument with INTENT(OUT), an updater is invoked to accept a value after the invoked procedure completes execution. Where it appears as an actual argument associated with a dummy argument with INTENT(INOUT) or unspecified intent, a

function is invoked to produce a value before the procedure to which it is an actual argument is invoked, and an updater is invoked to accept a value after the invoked procedure completes execution.

NOTE 2.2

If an accessor reference appears as an actual argument, copy-in, copy-out or copy-in/copy-out argument passing is required.

2.3 Updater definition syntax

1 An updater subprogram defines an updater procedure.

R1236a *updater-subprogram* **is** *updater-stmt*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-updater-stmt

R1236b *updater-stmt* **is** [*prefix*] UPDATER *updater-name* ■
■ ([*dummy-arg-name-list*]) [ACCEPT (*acceptor-name*)]

R1236c *end-updater-stmt* **is** END [UPDATER [*updater-name*]]

C1261a (R1236g) If ACCEPT appears, *acceptor-name* shall not be the same as *updater-name*.

C1261b (R1236a) An ENTRY statement shall not appear within the updater.

C1261c (R1236a) An internal updater subprogram shall not contain an *internal-subprogram-part*.

C1261d (R1236c) If *updater-name* appears in the *end-updater-stmt*, it shall be identical to the *updater-name* specified in the *updater-stmt*.

C1261e (R1236a) The acceptor variable name shall not be specified to have the ALLOCATABLE or POINTER attribute within the scoping unit of the updater. No INTENT attribute other than INTENT(IN) shall be specified for the acceptor variable name within the scoping unit of the updater.

2 The name of the updater is *updater-name*.

3 The type and type parameters of the updater name may be specified by a type specification in the UPDATER statement or by the updater name appearing in a type declaration statement in the *specification-part* of the scoping unit of the updater subprogram. They shall not be specified both ways. If they are not specified either way, they are determined by the implicit typing rules in force within the scoping unit of the updater. If the updater is an array, this shall be specified by specifications of the name of the updater variable within the scoping unit of the updater.

4 The acceptor variable is considered to be a dummy argument. Unless the VALUE attribute is specified for it within the updater part, it has the INTENT(IN) attribute, and this may be confirmed by explicit specification. The specifications of the acceptor variable attributes, the specification of dummy argument attributes, and the information in the UPDATER statement, collectively define the characteristics of the accessor (12.3.1).

NOTE 12.40a

An acceptor variable cannot be a pointer or allocatable.

5 If ACCEPT appears, the name of the acceptor variable of the updater is *acceptor-name* and all oc-

currences of the updater name in *execution-part* statements in the scoping unit of the updater refer to the updater itself. If ACCEPT does not appear, the acceptor variable name is *updater-name* and all occurrences of the updater name in *execution-part* statements in the scoping unit are references to the updater variable.

6 The characteristics of the updater where it is referenced in a variable definition context are the characteristics of the acceptor variable.

2.4 Invocation of an updater

1 When an updater is invoked, the following events occur in the order specified.

(1) All actual argument expressions and the expression (if any) defining the value to be accepted are evaluated. The value might be provided by other than an expression, for example by an associated INTENT(OUT) dummy argument, from a keyword in an input/output control list, in an input list, ...

(2) The actual arguments are associated with their corresponding dummy arguments. The value to be accepted is considered to be an actual argument, and is associated with the acceptor variable.

(3) All specification expressions within the *specification-part* of the scoping unit of the updater are evaluated.

(4) Control is transferred to the first executable construct of the *execution-part* of the updater.

2 When the updater is invoked the value of the acceptor variable is the accepted value and the updater shall not change the value of the acceptor variable unless it has the VALUE attribute.

3 As is the case with functions, if an updater is pure all dummy arguments shall have the INTENT(IN) attribute or the VALUE attribute.

4 Updaters are not interoperable; therefore the UPDATER statement does not include a *proc-language-binding-spec*.

2.5 Generic accessors

1 A generic interface bloc that specifies both functions and updaters specifies a generic accessor. The syntax of interface blocks is extended to define accessor generic interfaces.

R1205 *interface-body* **is** ...
or *updater-stmt*
[*specification-part*]
end-updater-stmt

2.6 Specific accessors

1 An ACCESSOR statement is provided to specify that a function and an updater are grouped together to form an accessor.

R1217a *accessor-declaration-stmt* **is** ACCESSOR (*accessor-interface*, *accessor-interface*) ■
■ [[, *accessor-attr-spec*] ... ::] *accessor-decl-list*

R1217b *accessor-interface* **is** *name*

R1217c *accessor-attr-spec* **is** *access-spec*
or INTENT (*intent-spec*)
or OPTIONAL

or POINTER
or SAVE

R1217d *accessor-decl* **is** *accessor-entity-name* [=> *accessor-pointer-init*]

R1217e *accessor-pointer-init* **is** *lbracket initial-accessor-target*, *initial-accessor-target rbracket*
or (/ *initial-accessor-target*, *initial-accessor-target* /)

R1217f *initial-accessor-target* **is** *procedure-name*

C1222a (R1217c) The *name* shall be the name of a nonintrinsic function or updater, or the name of an abstract interface of a nonintrinsic function or updater that has explicit interface. If is declared by a *procedure-declaration-stmt* it shall be previously declared.

C1222b (R1217a) One *accessor-interface* shall specify a function and the other *accessor-interface* shall specify an updater. The updater shall specify the same dummy arguments as the function, with the same names. The names and all attributes of corresponding dummy arguments shall be identical. The characteristics of the result variable of the function and the acceptor variable of the updater shall be identical, except that the function result value may be allocatable or a pointer.

NOTE 12.13a

An acceptor variable cannot be allocatable or a pointer.

C1222c (R1217d) If => appears in *accessor-decl*, the accessor entity shall have the POINTER attribute.

C1222d (R1217f) The *procedure-name* shall be the name of a nonelemental external or module function with explicit interface, or a nonelemental external or module updater.

C1222e (R1217e) One *initial-accessor-target* shall specify a function and the other shall specify an updater. The characteristics of the function shall be identical to the characteristics of the *accessor-interface* that specifies a function, except that the function may be pure even if the *accessor-interface* is not, and the characteristics of the updater shall be identical to the characteristics of the *accessor-interface* that specifies an updater, except that the updater may be pure even if the *accessor-interface* is not.

C1222f (R1217b) If *accessor-entity-name* is neither a pointer nor an actual argument, *accessor-interface* shall not specify an abstract interface.

2 If *accessor-entity-name* is neither a pointer nor an actual argument the *accessor-interface* specifications specify a specific function and a specific updater that correspond to the accessor. If *accessor-entity-name* is either a pointer nor an actual argument the *accessor-interface* specifications specify the characteristics of the functions and updaters that can correspond to the accessor.

2.7 Reference to updaters

1 A reference to an updater is permitted where definition of a variable is permitted.

R1218a *updater-reference* **is** *procedure-designator* [([*actual-arg-spec-list*])]

R1218b *actual-args* **is** ([*actual-arg-spec-list*])

C1223a (R1218a) The *procedure-designator* shall designate an updater.

2 If an updater name appears without *actual-args* it nonetheless specifies invocation of the updater unless it is an actual argument associated with a dummy procedure, or a *proc-target* in a pointer assignment

statement. For this reason, a procedure shall have explicit interface where it is invoked if it has an updater dummy procedure argument. If it is desired to invoke the updater when it appears in these contexts, *actual-args* shall appear.

2.8 Extension of function reference syntax

1 The syntax of *function-reference* is changed so as not always to require () if there are no actual arguments.

R1219 *function-reference* **is** *procedure-designator* [*actual-args*]

C1223a (R1219) If *actual-args* does not appear, *function-reference* shall not be a *proc-target* in a procedure pointer assignment statement.

NOTE 2.3

If a *procedure-designator* appears as the *proc-target* in a procedure pointer assignment statement, and *actual-args* does not appear, it is not a *function-reference*; it designates the function.

C1223b (R1219) If *actual-args* does not appear, *function-reference* shall not be an actual argument if the invoked procedure does not have explicit interface, or the invoked procedure has explicit interface and the corresponding argument is a function (with or without the POINTER attribute).

2 If *function-reference* is the *proc-target* in a procedure pointer assignment statement, or an actual argument, and the designated function returns a pointer result, and the result of the function is the intended pointer target or actual argument, *actual-args* shall appear.

2.9 Reference to accessors

1 A reference to a generic accessor is permitted where a reference to or definition of a variable is permitted. Where a generic accessor reference appears as a primary in an expression it is considered to be a reference to a function in its interface. Where a generic accessor appears in a variable definition context it is considered to be a reference to an updater in its interface. The usual generic resolution rules apply, with one extension: Since the value to be accepted is considered to be an actual argument, the acceptor variable characteristics contribute to generic resolution.

NOTE 2.4

The acceptor variable is considered to be a dummy argument and the value to be accepted is considered to be an actual argument.

A reference to a specific accessor is permitted where a reference to or definition of a variable is permitted. Where a specific accessor reference appears as a primary in an expression it is considered to be a reference to its function. Where a specific accessor appears in a variable definition context it is considered to be a reference to its updater.

R1218c *accessor-reference* **is** *procedure-designator* [(*actual-args*)]

C1223b (R1218c) The *procedure-designator* shall designate an accessor.

2 Unlike a reference to a function, if an accessor name appears without *actual-args* it nonetheless specifies invocation of the accessor unless it is an actual argument associated with a dummy procedure, or a *proc-target* in a pointer assignment statement. For this reason, a procedure shall have explicit interface where it is invoked if it has an accessor dummy procedure argument. If it is desired to invoke the accessor when it appears in these contexts, *actual-args* shall appear.

2.10 Reference to accessors and updaters in variable-definition contexts

1 The syntax of *designator* is extended to allow references to accessors in value-providing and variable definition contexts.

R601 *designator* **is** ...
or *accessor-reference*

2 The syntax of *variable* is extended to allow reference to updaters in variable-definition contexts.

R602 *variable* **is** *designator*
or *updater-reference*
or *expr*

3 The syntax of intrinsic assignment already allows reference to an accessor in its variable-definition context.

R732 *assignment-stmt* **is** *variable* = *expr*

4 If *assignment-stmt* is *accessor-reference* = *expr*, an updater in the accessor's interface is invoked.

5 The following intrinsic functions could be defined to be accessors. When a reference appears in a variable-definition context

- REAL(X) with complex X is equivalent to X%RE,
- AIMAG(X) with complex X is equivalent to X%IM,
- ABS(X) with numeric X changes the modulus without changing the phase,
- FRACTION(X) with real X changes the fraction, and
- EXPONENT(X) with real X changes the exponent.

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

The following editorial changes to ISO/IEC 1539-1:2010(E), if implemented, would provide the facilities described in foregoing clauses of this report. Descriptions of how and where to place the new material are enclosed between square brackets. Page and line numbers refer to ANSI/INCITS/PL22.3 standing document 10-007r1.

[2:10+ 1.3.1+] Editor: Insert new subclauses:

1.3.1a
acceptor variable
variable that transfers a value into an updater invoked in a variable definition context (12.6.2.3a)

1.3.1b
accessor
generic interface, procedure pointer, or dummy procedure that allows a function to be invoked by an expression or an updater (12.6.2.3a) to be invoked in a variable definition context

1.3.1b.1
specific accessor
accessor that is specified by a PROCEDURE statement

[5:3+ 1.3.20+] Editor: Insert two new subclauses:

"**1.3.20a**
characteristics
(acceptor variable) properties listed in 12.3.2a

[8:34+ 1.3.61+] Editor: Insert a new subclause:

"**1.3.62.1a**
dummy accessor
dummy argument that is an accessor"

[8:37 1.3.62] Editor: After "SUBROUTINE" insert ", UPDATER".

[9:37 1.3.66] Editor: Before "*end-block-data-stmt*" insert "*end-updater-stmt*, ".

[15:15+ 1.3.120+] Editor: Insert a new subclause:

"**1.3.120a**
accessor reference
appearance of a procedure designator for an accessor, or operator symbol in a context requiring execution of a function from an accessor interface during expression evaluation (12.5.3)"

[15:22 1.3.120.2] Editor: Append a clause at the end: "; an accessor reference that results in execution of a function part from the accessor iterface is a function reference".

[20:35+ 1.3.153+] Editor: Insert a new subclause:

"**1.3.153a**
updater
procedure that is invoked in a variable definition context

[27:18+ R203] Editor: Add an alternative for R203 *external-subprogram*:

or *updater-subprogram*

[27:28+ R203+] Editor: Add quotations of the new syntax rule R1236a:

R1236a *updater-subprogram* **is** *updater-stmt*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-updater-stmt

[28:30+ R211] Editor: Add an alternative for R211 *internal-subprogram*:

or *updater-subprogram*

[28:34+ R1108] Editor: Add an alternative for R1108 *module-subprogram*:

or *updater-subprogram*

[29:31+] Editor: Add an alternative for R214 *action-stmt*:

or *end-updater-stmt*

[30:8 C201] Editor: Delete "or"; After "*end-subroutine-stmt*" insert ", or *end-updater-stmt*, ".

[30:14 2.2.1p2] Editor: Replace "or" by a comma; after "subroutine" insert ", or an updater subprogram".

[30:26 2.2.3p1] Editor: Replace "either a function or a subroutine" by "a function, a subroutine, or an updater".

[31:18+2 Table 2.1] Editor: After "PROGRAM" insert ", UPDATER".

[32:11,13 2.3.3p1] Editor: After "*end-subroutine-stmt*" insert ", *end-updater-stmt*, " twice

---

[34:17 2.4.1.2p1] Editor: Replace "and function results" by ", function results, and acceptor values,"

[34:29+ 2.4.3.1p2+] Editor: Insert a new paragraph:

"A data entity that is passed to an updater that is invoked in a variable definition context is called the acceptor value."

[45:24+ 3.3.2.2p3] Editor: In the table of adjacent keywords where separating blanks are optional, insert "END UPDATER" in alphabetical order.

[52:3+ 4.3.1.2p2+] Editor: Insert a new paragraph:

If the data entity is an acceptor variable in an updater, the derived type may be specified in the UPDATER statement provided the derived type is defined within the body of the updater or is accessible there by use or host association. If the derived type is specified in the UPDATER statement and is defined within the body of the updater, it is as if the acceptor variable were declared with that derived type immediately after the *derived-type-def* of the specified derived type.

[73:31+ C741+] Editor: Insert a note:

NOTE 4.42a

If *generic-spec* is *generic-name* and *binding-name-list* includes both functions and updaters, one should probably provide a function with the same dummy argument characteristics and result variable type, kind, and rank as each updater's dummy argument characteristics and acceptor variable type, kind, and rank and vice versa.

[78:15 4.5.7.3p2] Editor: Replace "or" by a comma; append a phrase at the end of the sentence: ", or both shall be updaters for which all acceptor variables have the same characteristics (12.3.2a)".

[87:11+ 5.1p3+] Editor: Insert a new paragraph:

An updater has a type and rank and may have type parameters and other attributes that determine the uses of the updater. The type, rank, and type parameters are the same as those of its acceptor variable.

[89:20 C515] Editor: After "for" insert "an acceptor variable or for".

[91:17 C523] Editor: Before "a function" insert "an acceptor variable and not".

[91:20 C525] Editor: Before "and" insert "shall not be an acceptor variable,".

[97:9 C538] Editor: "or" by a comma; at the end insert ", or an acceptor variable".

[97:11+ C539+] Editor: Insert a new constraint:

C539a (R523) An entity with the INTENT(OUT) or INTENT(INOUT) attribute shall not be an acceptor variable.

[99:5 C543] Editor: Replace "or" by a comma; after "subroutine" insert ", or all be updaters".

[101:7 C554] Editor: Replace "a function result" by "an acceptor variable, a function result variable".

[103:6+ R527+] Editor: Insert a new constraint:

C563a (R527) An *object-name* shall not be an acceptor variable.

[104:30 C567] Editor: After "function result name" insert ", an acceptor variable".

[107:20+ C579+] Editor: Insert a new constraint:

C579a (R551) An *object-name* shall not be an acceptor variable.

[109:24 5.5p4] Editor: Add a sentence at the end of the paragraph: "An explicit type specification in an UPDATER statement overides an IMPLICIT statement for the name of the acceptor variable of that updater subprogram."

[117:3+ R601] Editor: Insert an additional alternative for syntax rule R601 *designator*:

or *accessor-reference*

[117:12+ R602] Editor: Insert an additional alternative for syntax rule R602 *variable*

or *updater-reference*

[125:15-16 6.5.4p3] Editor: Replace "reference to a function" by "function reference".

[150:5 7.1.11p1] Editor: At the end of the sentence insert "or UPDATER statement (12.6.2.1a)".

[150:22 7.1.11p2(9)] Editor: delete "function" (it's not needed to qualify "argument" in any of the other list items).

[158:30+ R740] Editor: Insert an additional alternative for R740 *proc-target*, and an additional syntax rule:

or *accessor-target*

R740a *accessor-target* **is** *accessor-name*
or *lbracket procedure-name*, *procedure-name rbracket*
or (/ *procedure-name*, *procedure-name* /)

[158:33 C729] Editor: After "pointer" insert ", a specific accessor".

[159:3+ C730+] Editor: Insert new constraints:

C729b (R738, R740) If *proc-pointer-name* or *proc-component-ref* is an accessor, *proc-pointer-object* shall have explicit interface.

C729c (R740) If *proc-target* is *accessor-target*, *proc-pointer-object* shall be a specific accessor, its function interface shall be the same as the interface of the function specified by *accessor-target*, and its updater interface shall be the same as the interface of the updater specified by *accessor-target*.

[159:3+ C730+] Editor: Insert new constraints:

C730a (R740a) The *accessor-name* shall be the name of a specific accessor.

C730b (R740a) One *procedure-name* shall specify a function and the other shall specify an updater.

C730c (R740a) The *procedure-name* shall not be a pointer.

[175:15 C816] Editor: After "*end-subroutine-stmt*" insert ", *end-accessor-stmt*".

[175:28 C818] Editor: After "*end-subroutine-stmt*" insert ", *end-accessor-stmt*".

[181:5 C828] Editor: After "*end-subroutine-stmt*" insert ", *end-accessor-stmt*".

[271:3 11.1p1] Editor: Before "MODULE" insert "UPDATER, ".

[272:10 R1108] Editor: Add an alternative for R1108:

or *updater-subprogram*

[277:7 12.1p2] Editor: After "FUNCTION" insert ", UPDATER".

[277:12 12.2.1p1] Editor: Replace "or a subroutine" by ", a subroutine, or an updater".

[277:15 12.2.1p1] Editor: Append a sentence at the end of the paragraph:

"A reference to an updater appears in a variable definition context."

[277:28 12.2.2p4] Editor: Replace "the" by "its"; replace "or FUNCTION" by ", FUNCTION, or UPDATER".

[278:9 12.3.1p1] Editor: Replace "or subroutine" by ", subroutine, or updater".

[278:11 12.3.1p1] Editor: Replace "and" by a comma. Append a new clause at the end of the sentence: ", and the characteristics of its acceptor variable if it is an updater".

[278:28+ 12.3.3-] Editor: Insert a new subclause:

"**12.3.2a Charactistics of acceptor variables**

Acceptor variables are considered to be dummy data objects."

[279:13 12.4.2.1p1] Editor: Replace "subroutine or a function" by "subroutine, a function". After "result name" insert ", or an updater with a separate acceptor variable name".

[279:23+ 12.4.2.2p1(2)+] Editor: Insert a list subitem:

"(a′) is a dummy accessor or updater procedure,"

[279:33+ 12.4.2.2p1(4-5)] Editor: Delete "or" on item (4), replace the period at the end of item (5) by ", or", and insert a list subitem:

"(6) the procedure is defined by an updater subprogram."

[280:4 12.4.3.1p1] Editor: After "SUBROUTINE" insert ", ACCESSOR".

[280:21+] Editor: Add an additional alternative for *interface-body*:

or *updater-stmt*
[*specification-part*]
end-updater-stmt

[281:2 C1203] Editor: Replace "or" by a comma. Before "shall" insert ", or the *updater-name* in the *updater-stmt*".

[281:18-19 12.4.3.2p3] Editor: Replace "or" by a comma; after "*subroutine-stmt*" insert ", or the *updater-name* in the *updater-stmt*".

[282:1- Note 12.4+] Editor: Insert a new note:

NOTE 12.4a

If *generic-spec* is *generic-name* and interfaces are provided for both functions and updaters, one should probably provide a a function with the same dummy argument characteristics and result variable type, kind, and rank as each updater's dummy argument characteristics and acceptor variable type, kind, and rank, and vice versa.

[283:9 12.4.3.4.1p2] Editor: After "function" insert "or updater".

[284:8 12.4.3.4.2p1] Editor: Replace "function" by "dummy".

[284:11-14 12.4.3.4.2p2] Editor: Delete "function's". Delete "of the function".

[286:3 12.4.3.4.5p3] Editor: After "functions" insert "or updaters".

[286:4-5 12.4.3.4.5p3] Editor: Delete "or", change period to ", or" insert a list item:

- one is an updater with nonzero rank and the other is not known to be a function.

[286:15 C1215] Editor: Replace "or both be functions" by ", both be functions, both be updaters, or one shall be a function and the other shall be an updater".

[286:38 12.4.3.4p5] Editor: After "functions" insert "or updaters,".

[289:1- 12.4.3.6+] Editor: Insert a new subclause:

1.2.4.3.6a Specific accessors

An ACCESSOR statement specifies that a function and an updater are grouped together to form a specific accessor.

R1217a *accessor-declaration-stmt* **is** ACCESSOR (*accessor-interface*, *accessor-interface*) ■
■ [[, *accessor-attr-spec*] ... ::] *accessor-decl-list*

R1217b *accessor-interface* **is** *name*

R1217c *accessor-attr-spec* **is** *access-spec*
or INTENT (*intent-spec*)
or OPTIONAL
or POINTER
or SAVE

R1217d *accessor-decl* **is** *accessor-entity-name* [=> *accessor-pointer-init*]

R1217e *accessor-pointer-init* **is** *lbracket initial-accessor-target*, *initial-accessor-target rbracket*
or (/ *initial-accessor-target*, *initial-accessor-target* /)

R1217f *initial-accessor-target* **is** *procedure-name*

C1222a (R1217c) The *name* shall be the name of a nonintrinsic function or updater, or the name of an abstract interface of a nonintrinsic function or updater that has explicit interface. If is declared by a *procedure-declaration-stmt* it shall be previously declared.

C1222b (R1217a) One *accessor-interface* shall specify a function and the other *accessor-interface* shall specify an updater. The updater shall specify the same dummy arguments as the function, with the same names. The names and all attributes of corresponding dummy arguments shall be identical. The characteristics of the result variable of the function and the acceptor variable of the updater shall be identical, except that the function result value may be allocatable or a pointer.

NOTE 12.13a

An acceptor variable cannot be allocatable or a pointer.

C1222c (R1217d) If => appears in *accessor-decl*, the accessor entity shall have the POINTER attribute.

C1222d (R1217f) The *procedure-name* shall be the name of a nonelemental external or module function

with explicit interface, or a nonelemental external or module updater.

C1222e (R1217e) One *initial-accessor-target* shall specify a function and the other shall specify an updater. The characteristics of the function shall be identical to the characteristics of the *accessor-interface* that specifies a function, except that the function may be pure even if the *accessor-interface* is not, and the characteristics of the updater shall be identical to the characteristics of the *accessor-interface* that specifies an updater, except that the updater may be pure even if the *accessor-interface* is not.

C1222f (R1217b) If *accessor-entity-name* is neither a pointer nor an actual argument, *accessor-interface* shall not specify an abstract interface.

If *accessor-entity-name* is neither a pointer nor an actual argument the *accessor-interface* specifications specify a specific function and a specific updater that correspond to the accessor. If *accessor-entity-name* is either a pointer nor an actual argument the *accessor-interface* specifications specify the characteristics of the functions and updaters that can correspond to the accessor.

[289:13+ 12.5.1] Editor: Insert new syntax rules and constraints, and a new paragraph:

R1218a *accessor-reference* **is** *procedure-designator* [*actual-args*]

R1218b *actual-args* **is** ([*actual-arg-spec-list*])

C1223a (R1218a) The *procedure-designator* shall designate an accessor.

Unlike a reference to a function, if an accessor name appears without *actual-args* it nonetheless specifies invocation of the accessor unless it is an actual argument associated with a dummy procedure, or a *proc-target* in a pointer assignment statement. For this reason, a procedure shall have explicit interface where it is invoked if it has an accessor dummy procedure argument. If it is desired to invoke the accessor when it appears in these contexts, *actual-args* shall appear.

NOTE 12.15a

If a dummy argument is a dummy accessor procedure, it is not possible to invoke the associated actual argument before or after invoking the procedure. It is not sensible to do so because the only possible use would be to return a procedure pointer. The acceptor variable of an updater cannot be a pointer, and therefore not a procedure pointer, and therefore the result of the function cannot be a procedure pointer.

[289:15+ 12.5.1] Editor: Insert new syntax rules and constraints

R1219a *updater-reference* **is** *procedure-designator* [*actual-args*]

C1223a (R1218a) The *procedure-designator* shall designate an updater.

[291:3 12.5.2.1p1] Editor: Replace "either a subroutine reference or a function reference" by "a reference to a subroutine, function or accessor".

[292:5 12.5.2.1p1] Editor: Insert a new paragraph before "Exactly":

"In a reference to an updater, the value to be accepted is considered to be an actual argument that corresponds to the acceptor variable, which is considered to be a dummy argument."

[292:10 12.5.2.2p1] Editor: Replace "*function-reference*" by "*updater-reference*, *function-reference*, or"

[302:2 12.5.3p1] Editor: Replace "a *function-reference* or by" by "an *accessor-reference*, a *function-*

reference, or".

[302:18+ 12.5.4p1+] Editor: Insert new subclauses:

12.5.4a Updater reference

An updater is invoked when an *accessor-reference* appears in a variable definition context. The value to be accepted is considered to be an actual argument associated with the acceptor variable. When an updater is invoked, all actual argument expressions are evaluated, then the arguments are associated, and then the updater is executed. When the actions specified by the updater are completed the value in the variable definition context has been accepted. In a reference to an elemental updater, all array arguments shall have the same shape.

12.5.4b Updater reference as an actual argument

When a subroutine or function completes execution, if an updater reference corresponds to an actual argument that does not have INTENT(IN), the updater is invoked to accept the value of that actual argument before a branch resulting from an alternate return occurs.

12.5.4c Accessor reference as an actual argument

When a subroutine, function, or updater is invoked, if any dummy argument that does not have INTENT(OUT) corresponds to an accessor reference, the function specified by the accessor is invoked to evaluate the actual argument to be associated with the dummy argument before the procedure is executed. If any dummy argument that does not have INTENT(IN) corresponds to an accessor reference, the updater specified by the accessor is invoked to accept the value of the dummy argument after the procedure completes execution and before a branch resulting from an alternate return occurs.

[303:38-39 12.5.5.2p4] Editor: After "a function name" by "an accessor name, a function name,"; delete "or"; after "subroutine name" insert ", or an updater name".

[304:14 12.5.5.4p2] Editor: Replace "a function or" by "an accessor, a function, or a"

[305:20 12.6.2.1p1] Editor: Replace "or FUNCTION" by ", FUNCTION, or UPDATER".

[305:23 12.6.2.1p2] Editor: Replace "or FUNCTION" by ", FUNCTION, or UPDATER".

[305:35 C1247] Editor: Replace "or *subroutine-stmt*" by "*subroutine-stmt*, or *updater-stmt*".

[308:17+ 12.6.2.3+] Editor: Insert a new subclause:

12.6.2.3a Updater subprogram

1 An updater subprogram is a subprogram that has an UPDATER statement as its first statement.

2 An updater subprogram defines an updater procedure.

R1236a *updater-subprogram* **is** *updater-stmt*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-updater-stmt

R1236b *updater-stmt* **is** [*prefix*] UPDATER *updater-name* ■
■ ([*dummy-arg-name-list*]) [(ACCEPT *acceptor-name*)]

R1236c *end-updater-stmt* **is** END [UPDATER [*updater-name*]]

C1261a (R1236g) If ACCEPT appears, *acceptor-name* shall not be the same as *updater-name*.

C1261b (R1236a) An ENTRY statement shall not appear within the updater.

C1261c (R1236a) An internal updater subprogram shall not contain an *internal-subprogram-part*.

C1261d (R1236c) If *updater-name* appears in the *end-updater-stmt*, it shall be identical to the *updater-name* specified in the *updater-stmt*.

3 The name of the updater is *updater-name*.

4 The type and type parameters of the updater name may be specified by a type specification in the UPDATER statement or by the acceptor variable name appearing in a type declaration statement in the *specification-part* of the scoping unit of the updater subprogram. They shall not be specified both ways. If they are not specified either way, they are determined by the implicit typing rules in force within the scoping unit of the updater. If the updater is an array, this shall be specified by specifications of the name of the acceptor variable within the scoping unit of the accessor.

5 The acceptor variable is considered to be a dummy argument. Unless the VALUE attribute is specified for it within the updater part, it has the INTENT(IN) attribute, and this may be confirmed by explicit specification. The specifications of the acceptor variable attributes, the specification of dummy argument attributes, and the information in the UPDATER statement, collectively define the characteristics of the updater (12.3.1).

NOTE 12.40a

An acceptor variable cannot be a pointer or allocatable.

6 If ACCEPT appears, the name of the acceptor variable is *acceptor-name* and all occurrences of the updater name in *execution-part* statements in the scoping unit of the updater refer to the updater itself. If ACCEPT does not appear, the acceptor variable name is *updater-name* and all occurrences of the updater name in *execution-part* statements in the scoping unit of the updater are references to the acceptor variable.

7 The characteristics of the updater where it is referenced in a variable definition context are the characteristics of the acceptor variable.

[309:7-8 12.6.2.5p1] Editor: Delete “or”; Before “Its interface” insert “, or by an *updater-subprogram* whose initial statement contains the word MODULE,”.

[312:18 C1276] Editor: After “function” insert “or updater”.

[314:14 12.8.3p1] Editor: Delete the first sentence: “An elemental subroutine... actual arguments” because its first part repeats C1289, and its second part doesn’t say anything new.

[314:20+ 12.8.3+] Editor: Insert a new subclause:

12.8.4 Elemental updater actual arguments

An elemental updater has only scalar dummy arguments, but may have array actual arguments. All actual arguments shall be conformable. If an actual argument is an array the effect is the same as would be obtained if the updater were applied separately, in array element order, to corresponding elements of each array actual argument.

NOTE 12.51

The acceptor value is considered to be a dummy argument. The value to be accepted is considered to be an actual argument.

[425:8+ 15.1p1+] Editor: Insert a note:

NOTE 15.0a

Updaters are not interoperable.

[441:20+ 16.3.3p1+] Editor: Insert a subclause:

16.3.3a Updater acceptor variables

For each UPDATER statement there is an acceptor variable. If there is no ACCEPT clause, the acceptor variable has the same name as the updater subprogram; otherwise the acceptor variable has the name specified in the ACCEPT clause.

[441:17-20 16.3.3] Editor: In lieu of the previous edit, delete subclause 16.3.3 because it duplicates 12.6.2.2p4 [307:12-20], and 16.3.3a would duplicate 12.6.2.3a paragraph 5.

[444:14 16.5.1.4p2(10)] Editor: After "in a *subroutine-stmt*" insert ", in an *updater-stmt*,"

[444:15 16.5.1.4p2(11+)] Editor: Insert a list item:

"(11a)an *acceptor-name* in an *updater-stmt*,"