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 — Accessor procedures

Technologies de l'information — Langages de programmation — Fortran — Procédures d'accès aux structures de données

Contents

0 Introduction . . . 1
- 0.1 History . . . 1
- 0.2 The problem to be solved . . . 1
- 0.3 What this report proposes . . . 1

1 General . . . 1
- 1.1 Scope . . . 1
- 1.2 Normative References . . . 1

2 Requirements . . . 2
- 2.1 General . . . 2
- 2.2 Summary . . . 2
- 2.3 Syntax of declaration of objects of type SECTION . . . 3
- 2.4 Section part designator . . . 4
- 2.5 Expressions of type SECTION . . . 4
- 2.6 Accessor definition syntax . . . 4
- 2.7 Invocation of an accessor . . . 6
- 2.8 Accessor interface bodies . . . 7
- 2.9 Reference to accessors . . . 7
- 2.10 Compatible extension of substring range . . . 8
- 2.11 Compatible extension of subscript triplet . . . 8
- 2.12 Compatible extension of vector subscript . . . 8
- 2.13 LOWER_BOUNDED (A) . . . 9
- 2.14 SECTION_AS_ARRAY (A) . . . 9
- 2.15 UPPER_BOUNDED (A) . . . 9
- 2.16 Existing intrinsic functions as accessors . . . 10

3 Required editorial changes to ISO/IEC 1539-1:2010(E) . . . 11

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

© ISO/IEC 2010 – All rights reserved

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.

 © ISO/IEC 2010 – All rights reserved

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.

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).

© ISO/IEC 2010 – All rights reserved

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.

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*

© ISO/IEC 2010 – All rights reserved

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.

2.2 Summary

2.2.1 General

1 This technical report defines a new form of subprogram called an *accessor*. An accessor defines two kinds of procedures, a function and a new kind of procedure called an *updater*. It can be invoked in a data reference context, in which case its function is executed and its result is a value. It can also be invoked in a variable definition context, in which case its updater is executed and the value is passed to the updater. 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 dual nature of invocation 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.

2 This technical report defines a new intrinsic data type called a *section* that has the same properties as a section subscript, and a constructor that has the same syntax as a section triplet. This allows variables and procedure dummy arguments that have those properties, which in turn allows the representation of an object to be changed between a function and updater, and an array, without changing the syntax of references to it.

2.2.2 Objects of type SECTION

1 Variables, structure components, function result values, or named constants can be of type SECTION. Objects of type SECTION have three parts, each represented by an integer, all of the same kind, which kind is specified in the object declaration. The parts are the lower bound, the upper bound, and the stride. There is additional data in the representation to indicate whether the lower or upper bound is specified. Therefore objects of type SECTION cannot be storage associated with objects of other types.

2.2.3 SECTION type declaration

1 The type declaration statement is extended to provide for declarations of objects of type SECTION.

2.2.4 SECTION structure component declaration

1 The data component declaration is extended to provide for declarations of structure components of type SECTION.

2.2.5 Constructor for values of type SECTION

1 The constructor for values of type SECTION is the same as *subscript-triplet*.

2.2.6 Reference to parts of SECTION objects

1 The lower bound part, upper bound part, and stride part of an object of type SECTION can be accessed using the same syntax as for structure component selection, or parts of a complex variable. The names of these parts are LBOUND, UBOUND, and STRIDE.

 © ISO/IEC 2010 – All rights reserved

2 Intrinsic procedures named LOWER_BOUNDED and UPPER_BOUNDED are provided to determine whether the lower and upper bound parts of an object of type SECTION are specified. If the stride is not specified its value is 1.

2.2.7 Constructing an array from a SECTION object

1 An intrinsic procedure named SECTION_AS_ARRAY is provided to produce a rank-one integer array whose values are the elements denoted by an object of type SECTION.

2.2.8 Definition of accessor subprograms

1 A new subprogram entity called an ACCESSOR is defined. An accessor has a *function part* and an *updater part*, each of which declares a subprogram that defines a procedure. When an accessor is referenced to provide the value of a primary during evaluation of a function, the function part is invoked, and the result value is provided in the same way as by a function subprogram. When an accessor is referenced in a variable definition context, the updater part is invoked, and the value to be defined is transferred to the updater in the acceptor variable.

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

2.2.9 Syntax of reference to accessor procedures

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

NOTE 2.1

For example, an accessor reference can appear within an expression, 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 accessor is referenced using an extension of the syntax of a function reference. The extended syntax is the same as is used to reference an array or a character substring, which in turn allows the representation of an object to be changed between a function and updater, and a character scalar or an array, without changing the syntax of references to it.

3 Where a reference appears in a value reference context the function part of the assessor is invoked to produce a value. Where it appears in a variable definition context the updater part is invoked to accept a value. Where it appears as an actual argument associated with a dummy argument with INTENT(IN), the function part 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), the updater part 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, the function part is invoked to produce a value before the procedure to which it is an actual argument is invoked, and the updater part 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 Syntax of declaration of objects of type SECTION

1 The syntax of *intrinsic-type-spec* is extended to provide for declaration of structure components, variables, and named constants of type SECTION.

© ISO/IEC 2010 – All rights reserved

R404 *intrinsic-type-spec* **is** ...
or SECTION [(*kind-type-selector*)]

2 The *kind-type-selector* specifies the kind of the integer parts of the entity declared. If it does not appear the integer parts are of default kind.

2.4 Section part designator

1 A section part designator has a syntax similar to component selection or a complex part designator.

R615a *section-part-designator* **is** *designator* % LBOUND
or *designator* % UBOUND
or *designator* % STRIDE

2 The type of *section-part-designator* is integer with the same kind as *designator*.

3 In a reference, if *section-part-designator* is *designator*%LBOUND, LOWER_BOUNDED(*designator*) shall not be false, and if *section-part-designator* is *designator*%UBOUND, UPPER_BOUNDED(*designator*) shall not be false. In a variable-definition, context LOWER_BOUNDED(*designator*) or UPPER_BOUNDED(*designator*) may be true.

NOTE 2.3

Alternatively, the value of *designator*%LBOUND could be −HUGE(*designator%LBOUND*) when LOWER_BOUNDED(*designator*) is false, and the value of *designator*%UBOUND could be HUGE(*designator*%UBOUND) when UPPER_BOUNDED(*designator*) is false.

2.5 Expressions of type SECTION

1 The syntax of *designator* is extended to include section part designators.

R601 *designator* **is** ...
or *section-part-designator*

2 The syntax of *primary* is extended to include the constructor for objects of type SECTION.

R701 *primary* **is** ...
or *section-constructor*

R424a *section-constructor* **is** [*scalar-int-expr*] : [*scalar-int-expr*] [: *scalar-int-expr*]

3 A *section-constructor* constructs a value of type SECTION. The first *scalar-int-expr* provides the lower bound for the section. If it does not appear, the LOWER_BOUNDED intrinsic function would return false. The second provides the upper bound for the section. If it does not appear, the UPPER_BOUNDED intrinsic function would return false. The third provides the stride. If it does not appear the value of the stride is 1.

4 No intrinsic operations are defined for objects of type SECTION.

2.6 Accessor definition syntax

1 An accessor is a subprogram that consists of two subprograms, a function and an updater, that have the same name. The abstract interfaces of the function and updater are identical, except that the acceptor variable of an updater cannot be a pointer or allocatable. An important addition to the syntax of an accessor definition is *aux-dummy-arg-name*, which allows a reference to have a syntax that is compatible with character substring reference.

 © ISO/IEC 2010 – All rights reserved

R1226a *accessor-subprogram* **is** *accessor-stmt*
[*specification-part*]
function-part
updater-part
[*internal-subprogram-part*]
end-accessor-stmt

R1226b *accessor-stmt* **is** [*prefix*] ACCESSOR *accessor-name* ■
■ ([*dummy-arg-name-list*]) [(*aux-dummy-arg-name*)]

R1226c *end-accessor-stmt* **is** END [ACCESSOR [*accessor-name*]]

R1226d *aux-dummy-arg-name* **is** *dummy-arg-name*

R1226d *function-part* **is** *function-part-stmt*
[*specification-part*]
[*execution-part*]

R1226e *updater-part* **is** *updater-part-stmt*
[*specification-part*]
[*execution-part*]

R1226f *function-part-stmt* **is** FUNCTION PART [RESULT (*result-name*)] ■

R1226g *updater-part-stmt* **is** UPDATER PART [ACCEPT (*acceptor-name*)]

C1251a (R1226b) If *aux-dummy-arg-name* appears it shall be scalar, have INTENT(IN), and be of type SECTION.

C1251b (R1226f) If RESULT appears, *result-name* shall not be the same as *accessor-name*, and no attributes other than the ALLOCATABLE or POINTER attributes shall be specified for the *result-name* in the scoping unit of the function part of the accessor.

C1251c (R1226g) If ACCEPT appears, *acceptor-name* shall not be the same as *accessor-name*, and no attributes other than the VALUE and INTENT(IN) attributes shall be specified for the *acceptor-name* in the scoping unit of the updater part of the accessor.

C1251d (R1226a) An ENTRY statement shall not appear within the accessor.

C1251e (R1226a) An internal accessor subprogram shall not contain an *internal-subprogram-part*.

C1251f (R1226c) If *accessor-name* appears in the *end-accessor-stmt*, it shall be identical to the *accessor-name* specified in the *accessor-stmt*.

C1251g (R1226a) The acceptor variable name shall not be specified to have the ALLOCATABLE or POINTER attribute within the scoping unit of the accessor or the scoping unit of the updater part of the accessor.

C1251h (R1226a) No characteristic of the function or updater subprograms defined by the accessor, except whether the result of the function has the ALLOCATABLE or POINTER attribute, shall be specified within the *specification-part* of the scoping unit of the function part or updater part of the accessor.

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

© ISO/IEC 2010 – All rights reserved

3 The type and type parameters of the accessor name may be specified by a type specification in the ACCESSOR statement or by the accessor name appearing in a type declaration statement in the *specification-part* of the scoping unit of the accessor 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 accessor. If the accessor is an array, this shall be specified by specifications of the name of the accessor within the scoping unit of the accessor. If the result variable is a pointer or allocatable, this shall be specified by specifications of the name of the result variable within the scoping unit of the function part of the accessor.

4 The acceptor variable is considered to be a dummy argument. It has all the attributes specified for the accessor name. 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 result variable has all the attributes specified for the accessor name. Within the function part, the POINTER or ALLOCATABLE attribute may be specified for it. The specifications of the result and acceptor variable attributes, the specification of dummy argument attributes, and the information in the ACCESSOR 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 RESULT appears, the name of the result variable of the function part of the accessor is *result-name* and all occurrences of the accessor name in *execution-part* statements in the scoping unit of the function part of the accessor refer to the accessor itself. If RESULT does not appear, the result variable name is *accessor-name* and all occurrences of the accessor name in *execution-part* statements in the scoping unit of the function part of the accessor are references to the result variable.

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

7 The characteristics (12.3.3) of the result of the accessor where it is referenced to produce the value of a primary within an expression are the characteristics of the result variable. The characteristics of the accessor where it is referenced in a variable definition context are the characteristics of the acceptor variable.

2.7 Invocation of an accessor

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

(1) The actual arguments are associated with their corresponding dummy arguments. If the accessor is invoked to accept a value the value to be accepted is considered to be an actual argument, and is associated with the acceptor variable.

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

(3) If the accessor is invoked to produce a value the function part is executed, else the updater part is executed.

2 When the accessor is invoked to produce a value

- if the result variable is a pointer, its pointer association status is undefined when execution of the accessor begins, the shape and association status of the result are determined by the shape of the result variable when execution of the accessor is completed, and the association status of the result variable shall not be undefined when execution of the accessor is completed;

 © ISO/IEC 2010 – All rights reserved

- if the result variable is not a pointer, its value is undefined when execution of the accessor begins, and it shall be defined by the accessor.

3 When the accessor is invoked to accept a value the value of the acceptor variable is the accepted value and the accessor shall not change the value of the acceptor variable unless it has the VALUE attribute.

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

5 Accessors are not interoperable; therefore the ACCESSOR statement does not include a *proc-language-binding-spec*.

2.8 Accessor interface bodies

1 The syntax of interface blocks is extended to allow accessor interface bodies.

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

2.9 Reference to accessors

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

2 The syntax of an accessor reference is an extension of the syntax of a function reference. The extension makes it compatible with a scalar reference, an array element reference, a whole array reference, or a character substring reference, which in turn allows the representation of an object to be changed between an accessor and a data object without changing the syntax of references to it.

R1218a *accessor-reference* **is** *procedure-designator* [*actual-args*] [(*aux-actual-arg*)]

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

R1218c *aux-actual-arg* **is** *actual-arg-spec*

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

3 Unlike a reference to a function, if an accessor name appears without either *actual-args* or *aux-actual-arg* 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, either *actual-args* or *aux-actual-arg* shall appear.

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

© ISO/IEC 2010 – All rights reserved

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

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

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

6 If *assignment-stmt* is *accessor-reference* = *expr*, the type and kind type parameter values of *expr* shall be the same as the transfer variable of the accessor. Either the rank of *accessor-reference* and *expr* shall be the same, or the accessor shall be elemental.

2.10 Compatible extension of substring range

1 The type SECTION is provided to allow a dummy argument of type SECTION, so that an accessor can replace an array or character variable without requiring change to the references. It seems pointless to restrict this only to actual arguments, so it makes sense to allow variables other than dummy arguments of type SECTION. Having a variable of type section and not allowing it to be used as a *substring-range* would be silly.

R610 *substring-range* **is** *scalar-section-expr*

R610a *scalar-section-expr* **is** *scalar-expr*

C608a (R610a) The *scalar-expr* shall be an expression of type SECTION.

2 The value of the stride of *scalar-section-expr* shall be 1.

Unresolved Technical Issue 1

Does this introduce a syntax ambiguity?

2.11 Compatible extension of subscript triplet

1 Having a variable of type section and not allowing it to be used as a *subscript-triplet* would be silly.

R621 *subscript-triplet* **is** *scalar-section-expr*

NOTE 2.4

Since no operations are defined on objects of type SECTION, the only possible expressions of type SECTION are section constructors, variables of type SECTION, references to functions or accessors of type SECTION, or such an expression enclosed in parentheses. Thus `A((1:10))` is a newly-allowed syntax having the same meaning as `A(1:10)`.

2.12 Compatible extension of vector subscript

1 Having an array of type section and not allowing it to be used as a *vector-subscript* would be silly.

R623 *vector-subscript* **is** *expr*

C627 (R623) A *vector-subscript* shall be an array expression of rank one and type integer or SECTION.

2 If *vector-subscript* is of type SECTION the effect is as if the elements appeared as arguments to a sequence of references to the SECTION_AS_ARRAY intrinsic function in an array constructor, in array element order.

 © ISO/IEC 2010 – All rights reserved

NOTE 2.5

For example, if A is an array with two elements having values 1:5:2 and 5:1:-2, the effect is as if the subscript were [SECTION_AS_ARRAY(A(1)), SECTION_AS_ARRAY(A(2))], which has the value [1, 3, 5, 5, 3, 1].

2.13 LOWER_BOUNDED (A)

1 **Description.** Whether a lower bound is specified for a section.

2 **Class.** Elemental function.

3 **Argument.** A shall be of type SECTION.

4 **Result Characteristics.** Default logical.

5 **Result Value.** The result value is true if and only if A has a lower bound.

2.14 SECTION_AS_ARRAY (A)

1 **Description.** An array having element values of all elements of a section.

2 **Class.** Transformational function.

3 **Argument.** A shall be a scalar of type SECTION. Neither LOWER_BOUNDED(A) nor UPPER_BOUNDED(A) shall be false. The value of A%STRIDE shall not be zero.

4 **Result Characteristics.** Rank one array of type integer and the same kind as A. The size of the result is the number of elements denoted by the section, which is MAX(0, (A%UBOUND − A%LBOUND + A%STRIDE) / A%STRIDE).

5 **Result Value.** The result value is the same as the expression [(I, I = A%LBOUND, A%UBOUND, A%STRIDE)] where I is an integer of the same kind as A.

NOTE 2.6

The description of the result value makes it clear that SECTION_AS_ARRAY is not really needed; it is pure syntactic sugar.

6 **Examples.** The value of SECTION_AS_ARRAY (5:1:-2) is [5, 3, 1]. The value of SECTION_AS_ARRAY (5:1:2) is [ ] and the size of the result value is zero.

2.15 UPPER_BOUNDED (A)

1 **Description.** Whether an upper bound is specified for a section.

2 **Class.** Elemental function.

3 **Argument.** A shall be of type SECTION.

4 **Result Characteristics.** Default logical.

5 **Result Value.** The result value is true if and only if A has an upper bound.

© ISO/IEC 2010 – All rights reserved

2.16 Existing intrinsic functions as accessors

1 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.

 © ISO/IEC 2010 – All rights reserved

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 accessor invoked in a variable definition context (12.6.2.1a)

1.3.1b
accessor
procedure that can be invoked by an expression or in a variable definition context (12.6.2.1a)

[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: Replace "a FUNCTION" by "an ACCESSOR, a FUNCTION".

[9:37 1.3.66] Editor: Before "*end-block-data-stmt*" insert "*end-accessor-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 the function part of the accessor 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 the function part of the accessor is a function reference".

[15:42 1.3.124] Editor: Before "BLOCK" insert "Accessor, ".

[18:15 1.3.143] Editor: Before "*function-subprogram*" insert "*accessor-subprogram* (R1226a), "; insert a comma before "or".

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

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

© ISO/IEC 2010 – All rights reserved

[27:17 R203] Editor: Add an alternative for R203 *external-subprogram*:

R203 *external-subprogram* **is** *accessor-subprogram*
or *function-subprogram*

[27:18 R203+] Editor: Add quotations of the new syntax rule R1226a:

R1226a *accessor-subprogram* **is** *accessor-stmt*
[*specification-part*]
function-part
updater-part
[*internal-subprogram-part*]
end-accessor-stmt

[28:29 R211] Editor: Add an alternative for R211 *internal-subprogram*:

R211 *internal-subprogram* **is** *accessor-subprogram*
or *function-subprogram*

[28:33 R1108] Editor: Add an alternative for R1108 *module-subprogram*:

R203 *module-subprogram* **is** *accessor-subprogram*
or *function-subprogram*

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

or *end-accessor-stmt*

[30:8 C201] Editor: Before "*end-function-stmt*" insert "*end-accessor-stmt*, ".

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

[30:17+ 2.2.1p2+] Editor: Insert a new note:

NOTE 2.1a

An accessor subprogram defines a function subprogram and 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 ", ACCESSOR".

[32:11,13 2.3.3p1] Editor: Before "*end-function-stmt*" insert "*end-accessor-stmt*, " twice

---

[33:8,11 2.3.5p2] Editor: Divide 2.3.5p2 into three paragraphs at "When" and "With". Then insert a new paragraph between the first two:

"When an accessor is invoked the specification expressions within the *specification-part* of the accessor, if any, are evaluated in a processor dependent order, followed by execution of either the function or updater part."

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

 © ISO/IEC 2010 – All rights reserved

[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."

[36:6,7 2.4.3.4p1] Editor: Before "(12.3.3)" insert "or accessor".

[36:31+2 Note 2.12] Editor: After "function" insert "or accessor".

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

[49:26 4.1.4p1] 4.1.4p1] Editor: After "functions" insert "and accessors".

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

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

[53:6+] Editor: Insert an additional alternative for syntax rule R404:

or SECTION [*kind-selector*]

[60:20+] Editor: Insert a new subclause:

4.4.5 Section type

4.4.5.1 General

The section type consists of three parts, all objects of integer type of the same kind. The processor shall provide a kind of type section corresponding to each integer kind. The kind type parameter of an object of section type is returned by the intrinsic function KIND (13.7.80).

The type specifier for the section type uses the keyword SECTION.

The keyword SECTION with no *kind-selector* specifies type section with the same kind as default integer kind.

The parts of an object of type section are the lower bound, the upper bound, and the stride.

No intrinsic operations are defined for objects of type section.

4.4.5.2 Construction of values of section type

R424a *section-constructor* **is** [*scalar-int-expr*] : [*scalar-int-expr*] [: *scalar-int-expr*]

A *section-constructor* constructs a value of type SECTION. The first *scalar-int-expr* provides the lower bound for the section. If it does not appear, the LOWER_BOUNDED intrinsic function would return false. The second provides the upper bound for the section. If it does not appear, the UPPER_BOUNDED intrinsic function would return false. The third provides the stride. If it does not appear the value of the stride is 1.

© ISO/IEC 2010 – All rights reserved

[73:15-16 C465] Editor: After “or” insert “ accessor,”; after “interface” insert “, or an external accessor subprogram”.

[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 accessors for which all acceptor variables and function result variables have the same characteristics (12.3.2a)”.

[80:27+2-3 Note 4.58] Editor: After “*function-reference*” insert “or *accessor-reference*” twice.

[87:7+ 5.1p1+] Editor: Insert a new paragraph:

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

[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 “functions” by “accessors, all be functions,”

[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: Replace “a function name, a function result” by “an acceptor variable, a function name, a function result 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 ACCESSOR statement overides an IMPLICIT statement for the name of the result variable of the function part of that accessor subprogram and the acceptor variable of the updater part of that accessor subprogram.”

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

or *accessor-reference*

[117:8+ R601] Editor: Insert an additional alternative for syntax rule R601 *designator*:

 © ISO/IEC 2010 – All rights reserved

or *section-part-designator*

[120:13-] Editor: Insert a new subclause:

6.4.4a Parts of objects of type section

R615a *section-part-designator* **is** *designator* % LBOUND
or *designator* % UBOUND
or *designator* % STRIDE

621a (R615a) The *designator* shall be of type section.

The type of *section-part-designator* is integer with the same kind as *designator*.

If *section-part-designator* is *designator*%LBOUND it designates the lower bound part. If it is *designator*%UBOUND it designates the upper bound part. If it is *designator*%STRIDE it designates the stride part.

In a reference, if *section-part-designator* is *designator*%LBOUND, a reference to the intrinsic function LOWER_BOUNDED(*designator*) shall not return false, and if *section-part-designator* is *designator*%UBOUND, a reference to the intrinsic function UPPER_BOUNDED(*designator*) shall not return false.

NOTE 6.6a

In a variable-definition, context LOWER_BOUNDED(*designator*) or UPPER_BOUNDED(*designator*) may be true.

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

[146:14 7.1.6.1p2] Editor: After "d_2" append ", or the function is the function part of an accessor and the ACCESSOR statement (12.3.2.1a) specifies one dummy argument d_2".

[146:30 7.1.6.1p5] Editor: After "d_2" append ", or the function is the function part of an accessor and the ACCESSOR statement (12.3.2.1a) specifies two dummy arguments, d_1 and d_2".

[150:5 7.1.11p1] Editor: before "a FUNCTION" insert "an ACCESSOR statement (12.6.2.1a) or".

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

[158:32-33 C729] Editor: After "dummy" insert "accessor or"; after "external" insert "accessor or".

[159:3+ C730+] Editor: Insert a new constraint:

C729a (R738, R740) If *procedure-name* or *proc-component-ref* is an accessor, *proc-pointer-object* shall have explicit interface.

[175:14 C816] Editor: Before "*end-function-stmt*" insert "*end-accessor-stmt*, ".

[175:27 C818] Editor: Before "*end-function-stmt*" insert "*end-accessor-stmt*, ".

[181:4 C828] Editor: Before "*end-function-stmt*" insert "*end-accessor-stmt*, ".

© ISO/IEC 2010 – All rights reserved

[218:1- Note 9.34+] Editor: Insert a list item:

- A list item of SECTION type shall be processed by a defined input/output procedure (9.6.4.8).

Unresolved Technical Issue UTI 1

List items of SECTION type are processed by a defined input/output procedure to avoid discussing what to do about the cases of `.not. LOWER_BOUNDED (` *expr* `)` and `.not. UPPER_BOUNDED (` *expr* `)`

[223:23 9.6.4.8.1p1] Editor: Before "objects" insert "or SECTION type".

[224:17+ R921+] Editor: Add an alternative for R921 *dtv-type-spec*:

or SECTION [*kind-selector*]

[228:6 9.6.4.8.4p2] Editor: Replace "derived-type" by "derived type or SECTION type".

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

[272:9 R1108] Editor: Add an alternative for R1108:

R1108 *module-subprogram* **is** *accessor-subprogram*
or *function-subprogram*

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

[277:12 12.2.1p1] Editor: Replace "a function or a subroutine" by "an accessor, a function, 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 as a reference to an accessor in a variable definition context."

[277:28 12.2.2p4] Editor: Replace "a procedure for the SUBROUTINE or FUNCTION statement" by "a subroutine procedure for its SUBROUTINE statement, a function procedure for its FUNCTION statement, or defines a function procedure and an updater procedure for its ACCESSOR statement".

[278:9 12.3.1p1] Editor: Replace "function or subroutine" by "function, 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 Characteristics of acceptors**

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:

 © ISO/IEC 2010 – All rights reserved

"(a′) is a dummy accessor 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 accessor subprogram."

[280:4 12.4.3.1p1] Editor: Before "FUNCTION" insert "ACCESSOR, ".

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

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

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

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

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

[283:15 12.4.3.4.1p4] Editor: After "functions" insert "or accessors".

[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 accessors".

[286:5 12.4.3.4.5p3] Editor: After "function" insert "or accessor" twice.

[286:15 C1215] Editor: Replace "or both be functions" by ", or both be functions or accessors".

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

[287:35+ C1222+] Editor: Insert a new constraint:

C1222a (R1216) If *initial-proc-target* is an accessor its interface shall be explicit, and the interface of *proc-entity-name* shall be explicit. The interfaces of *initial-proc-target* and *proc-entity-name* shall specify the same characteristics.

[288:8+ 12.4.3.6p4+] Editor: Insert a new note

NOTE 12.13a

The interface of an accessor is required to be explicit where it is referenced or used as a procedure pointer target, and the interface of a procedure pointer is required to be explicit where it is associated with an accessor.

© ISO/IEC 2010 – All rights reserved

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

R1218a *accessor-reference* **is** *procedure-designator* [*actual-args*] [(*aux-actual-arg*)]

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

R1218c *aux-actual-arg* **is** *actual-arg-spec*

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

Unlike a reference to a function, if an accessor name appears without either *actual-args* or *aux-actual-arg* 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 a dummy accessor procedure argument.

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.

[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".

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

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

[302:5 12.5.3p1] Editor: Replace "The characteristics" by "If the function is defined by an accessor the characteristics of the function result (12.3.3) are determined by the interface of the accessor and additional specifications, if any, of the function result variable; otherwise, the characteristics".

[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. The characteristics of the acceptor are determined by the interface of the accessor that defines the updater. In a reference to an elemental updater, all array arguments shall have the same shape.

12.5.4b 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.

 © ISO/IEC 2010 – All rights reserved

[303:38-39 12.5.5.2p4] Editor: Replace “a function name” by “an accessor name or a function name,”

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

[305:20 12.6.2.1p1] Editor: Replace “or FUNCTION” by “, ACCESSOR, or FUNCTION”.

[305:23 12.6.2.1p2] Editor: Replace “or FUNCTION” by “, ACCESSOR, or FUNCTION”.

[305:35 C1247] Editor: Replace “*function-stmt*” by “*accessor-stmt*, *function-stmt*”.

[306:12+ 12.6.2.1+] Editor: Insert a new subclause:

12.6.2.1a Accessor subprogram

1 An accessor is a subprogram that consists of two subprograms, a function and an updater, that have the same name. The abstract interfaces of the function and updater are identical, except that the acceptor variable of an updater cannot be a pointer or allocatable. An important addition to the syntax of an accessor definition is *aux-dummy-arg-name*, which allows a reference to have a syntax that is compatible with character substring reference.

R1226a *accessor-subprogram* **is** *accessor-stmt*
[*specification-part*]
function-part
updater-part
[*internal-subprogram-part*]
end-accessor-stmt

R1226b *accessor-stmt* **is** [*prefix*] ACCESSOR *accessor-name* ■
■ ([*dummy-arg-name-list*]) [(*aux-dummy-arg-name*)]

R1226c *end-accessor-stmt* **is** END [ACCESSOR [*accessor-name*]]

R1226d *aux-dummy-arg-name* **is** *dummy-arg-name*

R1226d *function-part* **is** *function-part-stmt*
[*specification-part*]
[*execution-part*]

R1226e *updater-part* **is** *updater-part-stmt*
[*specification-part*]
[*execution-part*]

R1226f *function-part-stmt* **is** FUNCTION PART [RESULT (*result-name*)] ■

R1226g *updater-part-stmt* **is** UPDATER PART [ACCEPT (*acceptor-name*)]

C1251a (R1226b) If *aux-dummy-arg-name* appears it shall be scalar, have INTENT(IN), and be of type SECTION.

C1251b (R1226f) If RESULT appears, *result-name* shall not be the same as *accessor-name*, and no attributes other than the ALLOCATABLE or POINTER attributes shall be specified for the

© ISO/IEC 2010 – All rights reserved

result-name in the scoping unit of the function part of the accessor.

C1251c (R1226g) If ACCEPT appears, *acceptor-name* shall not be the same as *accessor-name*, and no attributes other than the VALUE and INTENT(IN) attributes shall be specified for the *acceptor-name* in the scoping unit of the updater part of the accessor.

C1251d (R1226a) An ENTRY statement shall not appear within the accessor.

C1251e (R1226a) An internal accessor subprogram shall not contain an *internal-subprogram-part*.

C1251f (R1226c) If *accessor-name* appears in the *end-accessor-stmt*, it shall be identical to the *accessor-name* specified in the *accessor-stmt*.

C1251g (R1226a) The acceptor variable name shall not be specified to have the ALLOCATABLE or POINTER attribute within the scoping unit of the accessor or the scoping unit of the updater part of the accessor.

C1251h (R1226a) No characteristic of the function or updater subprograms defined by the accessor, except whether the result of the function has the ALLOCATABLE or POINTER attribute, shall be specified within the *specification-part* of the scoping unit of the function part or updater part of the accessor.

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

3 The type and type parameters of the accessor name may be specified by a type specification in the ACCESSOR statement or by the accessor name appearing in a type declaration statement in the *specification-part* of the scoping unit of the accessor 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 accessor. If the accessor is an array, this shall be specified by specifications of the name of the accessor within the scoping unit of the accessor. If the result variable is a pointer or allocatable, this shall be specified by specifications of the name of the result variable within the scoping unit of the function part of the accessor.

4 The acceptor variable is considered to be a dummy argument. It has all the attributes specified for the accessor name. 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 result variable has all the attributes specified for the accessor name. Within the function part, the POINTER or ALLOCATABLE attribute may be specified for it. The specifications of the result and acceptor variable attributes, the specification of dummy argument attributes, and the information in the ACCESSOR 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 RESULT appears, the name of the result variable of the function part of the accessor is *result-name* and all occurrences of the accessor name in *execution-part* statements in the scoping unit of the function part of the accessor refer to the accessor itself. If RESULT does not appear, the result variable name is *accessor-name* and all occurrences of the accessor name in *execution-part* statements in the scoping unit of the function part of the accessor are references to the result variable.

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

 © ISO/IEC 2010 – All rights reserved

7 The characteristics (12.3.3) of the result of the accessor where it is referenced to produce the value of a primary within an expression are the characteristics of the result variable. The characteristics of the accessor where it is referenced in a variable definition context are the characteristics of the acceptor variable.

[306:14 12.6.2.2p1] Editor: Before "FUNCTION" insert "FUNCTION PART statement (12.6.2.1a) or".

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

12.6.2.3a Updater subprogram

An updater subprogram is a subprogram that has an UPDATER PART statement (12.6.2.1a) as its first statement.

[309:6 12.6.2.5p1] Editor: Before "*function-subprogram*" insert "by an *accessor-subprogram* whose initial statement contains the word MODULE,".

[309:10 R1237] Editor: Replace the first line of R1237 *separate-module-subprogram*:

R1237 *separate-module-subprogram* **is** *mp-subroutine-or-function*
or *mp-accessor*

R1237a *mp-subroutine-or-function* **is** *mp-subprogram-stmt*

[309:14+ R1237+] Editor: Add a syntax rule to define *mp-accessor*:

R1237b *mp-accessor* **is** *mp-subprogram-stmt*
[*specification-part*]
function-part
updater-part
[*internal-subprogram-part*]
end-accessor-stmt

[309:21+ C1263+] Editor: Insert an additional constraint:

C1263a (R1237b) A *separate-module-subprogram* shall be *mp-accessor* if and only if the *procedure-name* in its *mp-subprogram-stmt* has been declared to be an accessor.

[312:16 C1276] Editor: Before "function" insert "accessor or".

[312:17+ 12.7p2+] Editor: Insert a note:

NOTE 12.46a

If an accessor subprogram is pure, the function and updater subprograms it defines are pure.

[313:15 12.8.1p2] Editor: Insert a note before "The following additional. . . :

NOTE 12.50a

If an accessor subprogram is elemental, the function and updater subprograms it defines are elemental.

[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.

© ISO/IEC 2010 – All rights reserved

[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.

[319 Table 13.1] Editor: Insert three list items in Table 3.1 in alphabetical order:

LOWER_BOUNDED	(A)	E	Query whether a lower bound is specified for a section
SECTION_AS_ARRAY	(A)	T	An array having elements defined by a section
UPPER_BOUNDED	(A)	E	Query whether an upper bound is specified for a section

---

[364:18+ 13.7.102+] Editor: Insert a subclause:

13.7.102a LOWER_BOUNDED (A)

Description. Whether a lower bound is specified for a section.

Class. Elemental function.

Argument. A shall be of type SECTION.

Result Characteristics. Default logical.

Result Value. The result value is true if and only if A has a lower bound.

[384:16+ 13.7.144+] Editor: Insert a subclause:

13.7.144a SECTION_AS_ARRAY (A)

Description. An array having element values of all elements of a section.

Class. Transformational function.

Argument. A shall be a scalar of type SECTION. Neither LOWER_BOUNDED(A) nor UPPER_BOUNDED(A) shall be false. The value of A%STRIDE shall not be zero.

Result Characteristics. Rank one array of type integer and the same kind as A. The size of the result is the number of elements denoted by the section, which is MAX(0, (A%UBOUND − A%LBOUND + A%STRIDE) / A%STRIDE).

Result Value. The result value is the same as the expression [(I, I = A%LBOUND, A%UBOUND, A%STRIDE)] where I is an integer of the same kind as A.

 © ISO/IEC 2010 – All rights reserved

NOTE 3.1

The description of the result value makes it clear that SECTION_AS_ARRAY is not really needed; it is pure syntactic sugar.

Examples. The value of SECTION_AS_ARRAY (5:1:-2) is [5, 3, 1]. The value of SECTION_AS_ARRAY (5:1:2) is [ ] and the size of the result value is zero.

[395:4+ 13.7.173+] Editor: Insert a subclause:

13.7.173a UPPER_BOUNDED (A)

Description. Whether an upper bound is specified for a section.

Class. Elemental function.

Argument. A shall be of type SECTION.

Result Characteristics. Default logical.

Result Value. The result value is true if and only if A has an upper bound.

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

NOTE 15.0a

Accessors are not interoperable.

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

For each FUNCTION PART statement there is a result variable. If there is no RESULT clause, the result variable has the same name as the accessor subprogram containing the function being defined; otherwise the result variable has the name specified in the RESULT clause.

16.3.3a Updater acceptor variables

For each UPDATER PART statement there is an acceptor variable. If there is no ACCEPT clause, the acceptor variable has the same name as the accessor subprogram containing the updater being defined; 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], the new paragraph in 16.3.3 would duplicate 12.6.2.1a paragraph 5, and 16.3.3a would duplicate 12.6.2.1a paragraph 6.

[444:14 16.5.1.4p2(10)] Editor: Before "in a *function-stmt*" insert "in an *accessor-stmt*,"

[444:15+ 16.5.1.4p2(11+)] Editor: Replace "in a *function-stmt*" by "in an *accessor-stmt*, in a *function-stmt*, in a *function-part-stmt*,".

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

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

© ISO/IEC 2010 – All rights reserved