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

The following editorial changes, if implemented, would provide the facilities described in foregoing clauses of ISO/IEC TR 29113. Descriptions of how and where to place the new material are enclosed in braces . Edits to different places within the same clause are separated by horizontal lines.

In the edits, except as specified otherwise by the editorial instructions, underwave (underwave) and strike-out (~~strike-out~~) are used to indicate insertion and deletion of text.

***J3COMMENT: This makes it much easier to read what is being changed; I am unsure whether this is suitable for the ISO document... probably a good idea if we can get away with it.

***J3COMMENT: Comments to J3 in this paper begin with ***J3COMMENT. These are not intended to form part of the TR.

J3 DOCUMENT ONLY: Page and line number references to 10-007r1 are in square brackets [], and references indicating which previous subclause gives rise to the edit are between asterisks **.

6.1 Edits to clause 1

{Insert new term definitions before term **1.3.9 attribute** [4:1-] *TR 29113 1.3.1*}

1.3.8a
assumed rank
⟨dummy variable⟩ the property of assuming the rank from its effective argument (5.3.8.7, 12.5.2.4)

1.3.8b
assumed type
⟨dummy variable⟩ being declared as TYPE (*) and therefore assuming the type and type parameters from its effective argument (4.3.1)

{Insert new term definition before **1.3.20 character context** [5:1- *TR 29113 1.3.3*}

1.3.19a
C descriptor
struct of type CFI_cdesc_t defined in the header `ISO_Fortran_binding.h`

{Insert new subclause before 1.6.2 Fortran 2003 compatibility [24:8-] *TR 29113 1.4.1*}

1.6.1a Fortran 2008 compatibility

This part of ISO/IEC 1539 is an upward compatible extension to the preceding Fortran International Standard, ISO/IEC 1539-1:2010(E). Any standard-conforming Fortran 2008 program remains standard-conforming under this part of ISO/IEC 1539.

***J3COMMENT: No need to mention the new intrinsic and EXTERNAL, that is already covered by general language in 1.6.1.

6.2 Edits to clause 4

{In 4.3.1.1 Type specifier syntax, insert additional production for R403 *declaration-type-spec* after the one for CLASS (*) [51:21+] *TR 29113 2.1p1*}

or TYPE (*)

{In 4.3.1.2 TYPE, edit the first paragraph as follows [51:32]}

A TYPE type specifier is used to declare entities that are assumed type, or of an intrinsic or derived type.

{In 4.3.1.2 TYPE, insert new paragraphs at the end of the subclause [52:3+] *TR 29113 2.1p2-p4*}

An entity that is declared using the TYPE(*) type specifier has assumed type and is an unlimited polymorphic entity (4.3.1.3). Its dynamic type and type parameters are assumed from its associated effective argument.

C407a An assumed-type entity shall be a dummy variable that does not have the ALLOCATABLE, CODIMENSION, POINTER or VALUE attributes.

An assumed-type variable shall not appear as a designator or expression except as an actual argument associated with a dummy argument that is assumed-type, or the first argument to the intrinsic and intrinsic module functions IS_CONTIGUOUS, LBOUND, PRESENT, RANK, SHAPE, SIZE, UBOUND, or C_LOC.

Unresolved Technical Issue TRnn+1

Should the above paragraph be a constraint?

***J3COMMENT: In the fullness of time I would expect to break type compatibility out into a separate subclause 4.3.1.4, which will avoid confusion.

6.3 Edits to clause 5

{In 5.3.7 CONTIGUOUS attribute, edit C530 as follows [93:6]}

C530 An entity with the CONTIGUOUS attribute shall be an array pointer, ~~or~~ an assumed-shape array, or have assumed rank.

{In 5.3.7 CONTIGUOUS attribute, edit paragraph 1 as follows [93:7]}

The CONTIGUOUS attribute specifies that an assumed-shape array can only be argument associated with a contiguous effective argument, ~~or~~ that an array pointer can only be pointer associated with a contiguous target, or that an assumed-rank object can only be argument associated with a scalar or contiguous effective argument.

{In 5.3.7 CONTIGUOUS attribute, paragraph 2, item (3) [93:12]}

Change first “array” to “or assumed-rank dummy argument”,
change second “array” to “object”.

{In 5.3.8.1 General, edit paragraph 1 as follows [94:3-4]}

The DIMENSION attribute specifies that an entity has assumed rank or is an array. An assumed-rank entity has the rank and shape of its associated actual argument; otherwise, the~~The~~ rank or rank and shape is specified by its *array-spec*.

{In 5.3.8.1 General, insert additional production for R515 *array-spec*, after *implied-shape-spec-list* [94:10+] *TR 29113 2.2p1*}

or *assumed-rank-spec*

{At the end of 5.3.8, immediately before 5.3.9, insert new subclause [96:31+] *TR 29113 2.2p2*}

5.3.8.7 Assumed-rank entity

An assumed-rank entity is a dummy variable whose rank is assumed from its effective argument; this rank may be zero. An assumed-rank entity is declared with an *array-spec* that is an *assumed-rank-spec*.

R522a *assumed-rank-spec* **is** ..

C535a An assumed-rank entity shall be a dummy variable that does not have the CODIMENSION or VALUE attribute.

An assumed-rank variable shall not appear as a designator or expression except as an actual argument corresponding to a dummy argument that is assumed-rank, the argument of the C_LOC function in the ISO_C_BINDING intrinsic module, or the first argument in a reference to an intrinsic inquiry function. The RANK inquiry intrinsic may be used to inquire about the rank of an array or scalar object.

Unresolved Technical Issue TRnn+2

Should the above paragraph, except the final sentence, be a constraint?

Unresolved Technical Issue TRnn+3

An assumed-rank entity is neither scalar nor an array, so the remark about RANK is pointless. RANK is also doubtless wrong. The remark about RANK should be more specific, and should probably be a note.

6.4 Edits to clause 6

{In 6.5.4 Simply contiguous array designators, paragraph 2, edit the second bullet item as follows [125:4]}

- an *object-name* that is not a pointer, not ~~or~~ assumed-shape, and not assumed-rank,

---

{In 6.7.3.2 Deallocation of allocatable variables, append to paragraph 6 [131:9]}

When a Fortran procedure that has an INTENT (OUT) allocatable dummy argument is invoked by a C function and the corresponding argument in the C function call is a C descriptor that describes an allocated allocatable variable, the variable is deallocated on entry to the Fortran procedure. When a C function is invoked from a Fortran procedure via an interface with an INTENT (OUT) allocatable dummy argument and the corresponding actual argument in the reference of the C function is an allocated allocatable variable, the variable is deallocated on invocation (before execution of the C function begins).

6.5 Edits to clause 12

{In 12.3.2.2, edit paragraph 1 as follows [278:17,22]}

The characteristics of a dummy data object are its type, its type parameters (if any), its shape (unless it is assumed-rank), its corank, its codimensions, its intent (5.3.10, 5.4.10), whether it is optional (5.3.12, 5.4.10), whether it is allocatable (5.3.3), whether it has the ASYNCHRONOUS (5.3.4), CONTIGUOUS (5.3.7), VALUE (5.3.18), or VOLATILE (5.3.19) attributes, whether it is polymorphic, and whether it is a pointer (5.3.14, 5.4.12) or a target (5.3.17, 5.4.15). If a type parameter of an object or a bound of an array is not a constant expression, the exact dependence on the entities in the expression is a characteristic. If a rank, shape, size, type, or type parameter is assumed or deferred, it is a characteristic.

{In 12.4.2.2 Explicit interface, after item (2)(c) insert new item [279:27+]}

(c2) has assumed rank,

{In 12.5.2.4 Ordinary dummy variables, append to paragraph 2 [293:5]}

If the actual argument is of a derived type that has type parameters, type-bound procedures, or final subroutines, the dummy argument shall not be assumed type.

{In 12.5.2.4 Ordinary dummy variables, paragraphs 3 and 4 [293:8-9,13]}

Change "not assumed shape" to "explicit-shape or assumed-size" (twice).

{In 12.5.2.4 Ordinary dummy variables, paragraph 9 [294:7]}

After "dummy argument is a scalar"
Change "or" to ", has assumed rank, or is".

{In 12.5.2.4 Ordinary dummy variables, insert new paragraph after paragraph 14 [294:34+]}

An actual argument of any rank may correspond to an assumed-rank dummy argument. The rank and shape of the dummy argument are the rank and shape of the corresponding actual argument. If the rank is nonzero, the lower and upper bounds of the dummy argument are those that would be given by the intrinsic functions LBOUND and UBOUND respectively if applied to the actual argument.

{In 12.6.2.2 Function subprogram, edit C1255 as follows [306:30-33] *TR 29113 2.3*}

C1255 (R1229) If *proc-language-binding-spec* is specified for a procedure, each of the procedure's dummy arguments shall be an ~~nonoptional~~ interoperable variable (15.3.5, 15.3.6) that does not have both the OPTIONAL and VALUE attributes, or an ~~nonoptional~~ interoperable procedure (15.3.7). If *proc-language-binding-spec* is specified for a function, the function result shall be an interoperable scalar variable.

6.6 Edits to clause 13

{In 13.5 Standard generic intrinsic procedures, Table 13.1, LBOUND and UBOUND intrinsic functions [321,323]}

Delete " of an array" (twice).

{In 13.5 Standard generic intrinsic procedures, Table 13.1 [322]}

Insert new entry into the table, alphabetically

RANK (A) I Rank of a data object.

{In 13.7.86, IS_CONTIGUOUS, edit paragraph 3 as follows [359:4]}

Argument. ARRAY may be of any type. It shall be an array or an assumed-rank object. If it is a pointer it shall be associated.

{In 13.7.86, IS_CONTIGUOUS, edit paragraph 5 as follows [359:6]}

Result Value. The result has the value true if ARRAY has rank zero or is contiguous, and false otherwise.

{In 13.7.90 LBOUND, edit paragraph 1 as follows [359:30]}

Description. Lower bound(s) ~~of an array~~.

{In 13.7.90 LBOUND, edit paragraph 3, ARRAY argument, as follows [359:30]}

ARRAY shall be an array or assumed-rank object of any type. It shall not be an unallocated allocatable variable or a pointer that is not associated.

{In 13.7.93 LEN, paragraph 3 [361:10]}

Change "a type character scalar or array"
to "of type character".

{Immediately before subclause 13.8.138 REAL, insert new subclause [381:17-]}

13.7.137a RANK (A)

Description. Rank of a data object.

Class. Inquiry function.

Argument. A shall be a data object of any type.

Result Characteristics. Default integer scalar.

Result Value. The result is the rank of A.

Example. If X is declared as REAL X (:, :, :), the result has the value 3.

{In 13.7.149 SHAPE, edit paragraph 5 as follows [386:23]*TR 29113 3.4.1*}

Result Value. The result has a value equal to [(SIZE(SOURCE, *i*, KIND), *i*=1, RANK(SOURCE))].

{In 13.7.156 SIZE, edit paragraph 3, argument ARRAY, as follows [388:19]}

ARRAY shall be an array or assumed-rank object of any type. It shall not be an unallocated allocatable variable or a pointer that is not associated. If ARRAY is an assumed-size array, DIM shall be present with a value less than the rank of ARRAY.

{In 13.7.156 SIZE, edit paragraph 5 as follows [388:29-30]*TR 29113 3.4.2*}

Result Value. If ARRAY is an assumed-rank object associated with an assumed-size array and DIM is present with a value equal to the rank of ARRAY, the result is −1; otherwise, if DIM is present, the result has a value equal to the extent of dimension DIM of ARRAY. If DIM is not present, the result has a value equal to PRODUCT([(SIZE(ARRAY, *i*, KIND), *i*=1, RANK(ARRAY))]).

{In 13.7.160 STORAGE_SIZE, paragraph 3 [390:5]}

Change "a scalar or array of any type"
to "a data object of any type".

{In 13.7.171 UBOUND, paragraph 1 [394:20]}

Delete " of an array".

{In 13.7.171 UBOUND, paragraph 3, ARRAY argument [394:23]}

After "shall be an array"
insert "or assumed-rank object".

{In 13.7.171 UBOUND, edit paragraph 5 as follows [394:34]*TR 29113 3.4.3*}

Result Value.

Case (i): For an array section or for an array expression, other than a whole array, UBOUND (ARRAY, DIM) has a value equal to the number of elements in the given dimension~~; otherwise,~~.

Case (ii): For an assumed-rank object associated with an assumed-size array, UBOUND(ARRAY, *n*) where *n* is the rank of ARRAY has a value equal to LBOUND(ARRAY, *n*) − 2.

Case (iii): Otherwise, UBOUND(ARRAY, DIM) has a value equal to the upper bound for subscript DIM of ARRAY if dimension DIM of ARRAY does not have size zero and has the value zero if dimension DIM has size zero.

Case (iv): UBOUND (ARRAY) has a value whose i^{th} element is equal to UBOUND (ARRAY, i), for $i = 1, 2, \ldots, n$, where n is the rank of ARRAY.

6.7 Edits to clause 15

{In 15.1 General, at the end of the subclause, insert new paragraph [425:11+]}

The header `ISO_Fortran_binding.h` provides definitions and prototypes to enable a C function to interoperate with a Fortran procedure with an allocatable, assumed character length, assumed-shape, assumed-rank, or pointer dummy data object.

{In 15.3.7 Interoperability of procedures and procedure interfaces, paragraph 2, edit item (5) as follows [433:14-16]}

(5) any dummy argument without the VALUE attribute corresponds to a formal parameter of the prototype that is of pointer type, and either

(a) the dummy argument is interoperable with an entity of the referenced type (ISO/IEC 9899:1999, 6.25, 7.17, and 7.18.1) of the formal parameter,

(b) the dummy argument is a nonallocatable, nonpointer variable of type CHARACTER with assumed length, and corresponds to a formal parameter of the prototype that is a pointer to CFI_desc_t,

(c) the dummy argument is allocatable, assumed-shape, assumed-rank, or a pointer, and corresponds to a formal parameter of the prototype that is a pointer to CFI_cdesc_t, or

5a the dummy argument is assumed-type and not allocatable, assumed-shape, assumed-rank, or a pointer, and corresponds to a formal parameter of the prototype that is a pointer to void,

(6) each allocatable or pointer dummy argument of type CHARACTER has deferred character length, and,

---

{In 15.3.7 Interoperability of procedures and procedure interfaces, insert new paragraphs at the end of the subclause [437:23+]}

If a dummy argument in an interoperable interface is allocatable, assumed-shape, assumed-rank, or a pointer, the corresponding formal parameter is interpreted as a pointer to a C descriptor for the effective argument in a reference to the procedure. The C descriptor shall describe an object of interoperable type and type parameters with the same characteristics as the effective argument.

An absent actual argument in a reference to an interoperable procedure is indicated by a corresponding formal parameter with the value NULL.

{At the end of clause 15 [437:23+]}

Insert subclause 5.2 of ISO/IEC TR 29113 as subclause 15.5, including subclauses 5.2.1 to 5.2.7 as subclauses 15.5.1 to 15.5.7.

6.8 Edits for annex C

{In C.11 Clause 15 notes, at the end of the subclause [519:42+]}

Insert subclauses A.1.1 to A.1.4 as subclauses C.11.6 to C.11.9.

Insert subclause A.2 as C.11.10 with the revised title "Processing assumed-shape arrays in C".