Date:	11 August 1998
To:	J3
From:	Van Snyder
Subject:	Edits for type-bound procedures – R6.b Polymorphism
References:	97-230r1, 98-136, 98-152r1

1 Background

Paper 97-230r1 provided specifications for type-bound procedures; papers 98-136 and 98-152r1 proposed syntax for type-bound procedures; paper 98-152r1 was approved at meeting 145. This paper provides partial edits for type bound procedures. Edits for the SELECT KIND construct described in 98-152r1 are not complete.

2 Edits

Edits refer to 98-007r2. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by immediately following text, while a page and line number followed by + indicates that immediately following text is to be inserted after the indicated line. Remarks for the editor are noted in the margin, or appear between [and] in the text.

[39:3+]

[CONTAINS
  [PRIVATE]
  [*proc-binding-construct*]...]

[39:29+]

[Editor: Part of R428]

or *proc-component-def-stmt*

[40:8-9]

[Editor: Delete – Should be part of R428]

[40:22+]

R432a *proc-component-def-stmt* **is** PROCEDURE([*proc-interface*]), ■
■ *proc-component-attr-spec-list* :: ■
■ *proc-decl-list*

R432b *proc-component-attr-spec* **is** POINTER
or PASS_OBJ

Constraint: The POINTER attribute shall be specified.

[Editor: Add **passed-object dummy argument** to the index.]

If PASS_OBJ is specified the *proc-interface* shall have a dummy argument that has the same type as the *type-name*. The first of these is called the **passed-object dummy argument**. It shall be a scalar nonpointer variable. The use of PASS_OBJ is explained in [new section] 12.4.1.1.

R432c *proc-binding-construct* **is** *proc-binding*
or *select-kind-construct*

R432d *proc-binding* **is** *binding-by-name*

R432e *binding-by-name* **is** PROCEDURE [[, *binding-attr*] ::] ■
■ *binding-name* [=> *binding*]

If => *binding* is absent it is assumed to have been present with the same name as *binding-name*.

R432f *binding-attr* **is** PASS_OBJ
or NON_OVERRIDABLE
or *access-spec*

If PASS_OBJ is specified the *proc-interface* shall have a dummy argument that has the same type as the *type-name*. The first of these is called the **passed-object dummy argument**. It shall be a scalar nonpointer polymorphic variable. The use of PASS_OBJ is explained in [new section] 12.4.1.1.

R432g *binding* **is** *procedure-name*
or *deferred-binding*

Constraint: The procedure name shall be the name of an accessible module procedure or external procedure that has explicit interface.

R432h *select-kind-construct* **is** *select-kind-stmt*
[*case-stmt*
proc-binding-construct ...]
end-select-kind-stmt

R432i *select-kind-stmt* **is** SELECT CASE (*scalar-initialization-expr*)

I don't see how this can be differently useful from type-bound generic procedures if *scalar-initialization-expr* is allowed to be any more general than *kind-parameter-name*. *J3 note*

Constraint: The *scalar-initialization-expr* shall be of type integer.

R432j *end-select-kind-stmt* **is** END SELECT

[Editor: Insert new section. Add **type bound procedure** and **binding** to the index.] [44:22+]

4.5.1.5 Type-bound procedures

Each binding specifies a **type-bound procedure**. If a type is accessible the public binding names of its type-bound procedures are accessible. The specific names of procedures bound to the type are not automatically made accessible by accessing the type.

An example of a type and a type-bound procedure Note 4.x

```
TYPE, EXTENSIBLE :: POINT
  REAL :: X, Y
CONTAINS
  PROCEDURE, PASS_OBJ :: LENGTH => POINT_LENGTH
END TYPE POINT
REAL FUNCTION POINT_LENGTH ( A, B )
  CLASS(POINT), INTENT(IN) :: A, B
  POINT_LENGTH = SQRT( (A%X - B%X)**2 + (A%Y - B%Y)**2 )
END FUNCTION POINT_LENGTH
```

4.5.1.5.1 Deferred type-bound procedures

R432k *deferred-binding* **is** NULL([*abstract-interface-name*])

Constraint: The abstract interface name shall be the name of an abstract interface (12.3.2.1.4).

Constraint: The abstract interface name shall be present unless the binding is overriding (4.5.3.2) an inherited (4.5.1.3) binding.

A binding that specifies the NULL intrinsic instead of a procedure name creates a deferred type-bound procedure. The *abstract-interface-name* argument to the NULL intrinsic is required to establish the characteristics of the binding unless they are inherited (4.5.1.3) from the parent type.

An extension of a type that specifies a deferred type-bound procedure shall contain a procedure binding for each inherited (4.5.1.3) deferred type-bound procedure. This new binding may confirm that the type-bound procedure is still deferred, or supply a specific procedure.

It is possible to override (4.5.3.2) an inherited (4.5.3.1) binding with a null binding.

[Editor: Add **inherit** to the index.] [47:39-44]

4.5.3.1 Inheritance

An extended type includes all of the type parameters, components, and procedure bindings of the parent type. These are said to be **inherited** by the extended type from the parent type. Entities inherited by the parent type from its parent type are inherited by an extended type. Inherited entities retain their attributes. Additional type parameters, components, and procedure bindings may be declared in the derived type definition for the extended type.

The order of type parameters for an extended type is the type parameters inherited from the parent type, followed by type parameters declared in the extended type, in the order declared.

For purposes of intrinsic input/output (9.4.2) and value construction (4.5.6), the order of the components of an extended type is the components inherited from the parent type, followed by the components declared in the derived type definition of the extended type, in the order declared.

Editor: Replace “component” by “component or type parameter” twice.] [48:16-17]

[Editor: Start a new paragraph, add **override** to the index.] [48:28+]

4.5.3.2 Type-bound procedure overriding

If a binding has the same binding name as one inherited from the parent type:

- It shall have PASS_OBJ specified if PASS_OBJ is specified for the binding inherited from the parent type.
- It shall not have PASS_OBJ specified if PASS_OBJ is not specified for the binding inherited from the parent type.
- It shall be pure if the binding inherited from the parent type is pure.
- It shall be elemental if the binding inherited from the parent type is elemental.
- It shall not be elemental if the binding inherited from the parent type is not elemental.
- It shall have the same number of dummy arguments as the binding inherited from the parent type. Each dummy argument other than the passed-object dummy argument shall have the same characteristics (12.2.1) and dummy argument name as the binding inherited from the parent type.
- It shall be a subroutine if the binding inherited from the parent is a subroutine.
- It shall be a function having the same result characteristics (12.2.2) as the binding inherited from the parent type if the the binding inherited from the parent type is a function.
- The binding inherited from the parent type shall not have NON_OVERRIDABLE specified.
- The binding declared in the type **overrides** the one inherited from the parent. The binding inherited from the parent is not accessible in objects of the type.

An example of procedure over-riding. See example 4.x. Note 4.y

```
TYPE, EXTENDS(POINT) :: POINT_3D
  REAL :: Z
CONTAINS
  PROCEDURE, PASS_OBJ :: LENGTH => POINT_3D_LENGTH
END TYPE POINT_3D
REAL FUNCTION POINT_3D_LENGTH ( A, B )
  CLASS(POINT_3D), INTENT(IN) :: A, B
  IF ( EXTENDS_TYPE_OF(B,A) ) THEN
    POINT_3D_LENGTH = SQRT( (A%X-B%X)**2 + (A%Y-B%Y)**2 + (A%Z-B%Z)**2 )
    RETURN
  END IF
  PRINT *, 'In POINT_3D_LENGTH, dynamic type of argument is incorrect'
  STOP
END FUNCTION POINT_3D
```

4.5.3.3 Procedure binding accessibility

The default accessibility of procedure bindings is PUBLIC, independently from the accessibility of components. The default accessibility may be changed by an explicit PRIVATE statement following the CONTAINS, and may be overridden by an accessibility attribute.

---

or *type-bound-proc-name* ([*actual-arg-spec-list*]) [224:8+]

---

type-bound-proc-name **is** *data-ref* % *binding-name* [224:10+]

Constraint: The *binding-name* shall be the name of a procedure binding (4.5.1.5) to the declared type of the *data-ref*.

The procedure binding named by *type-bound-proc-name* is determined by the dynamic type of the *data-ref*. The procedure binding to the dynamic type of the *data-ref* shall not be deferred (4.5.1.5.1).

---

or CALL *type-bound-proc-name* ■
■ [([*actual-arg-spec-list*])] [224:12+]

[Editor: add "that does not refer to a type-bound procedure for which PASS_OBJ is specified," after "function reference,"] [225:11]

[Editor: Add a new section. Add **passed-object dummy argument** to the index.] [225:36+]

12.4.1.1 The effect of PASS_OBJ on argument association

In a reference to a type-bound procedure for which the binding includes the PASS_OBJ annotation, the *data-ref* is associated, as an actual argument, to the passed-object dummy argument (4.5.1). In a procedure reference that uses a structure component that is a procedure pointer that has the PASS_OBJ annotation, the penultimate *part-ref* is associated, as an actual argument, to the passed-object dummy argument (4.5.1). The actual argument list identifies the correspondence between the actual arguments supplied and the remaining dummy arguments. In the absence of an argument keyword, an actual argument is associated to the dummy argument occupying what would be the corresponding position in the dummy argument list if the passed-object dummy argument were removed. If an argument keyword is present, the actual argument is associated to the dummy argument whose name is the same as the argument keyword. The passed-object dummy argument shall not be identified by an argument keyword.

[Editor: Add "binding name" to the list.] [303:38]

[Editor: Add "(12.4.1)" after "reference".] [307:4]

[Editor: Add "(12.4.1)" after "reference".] [307:9]

14.1.2.5 Components, type parameters, subobjects and bindings [308:33]

A binding name has the scope of the derived type definition. Outside of the type definition, it may appear only within a call statement or function reference. If the type is accessible in another scoping unit by use association or host association and the type does not contain the PRIVATE statement (4.5.1), the binding name is accessible for use in a call statement or function reference in that scoping unit. [308:47+]

---

binding (4.5.1.5): An association, declared within a derived type definition, of a specific procedure to a name. [342:3+]

[Editor: Add in the same paragraph] [346:37+]

(4.5.3)If a procedure is bound to an extensible type by the same *binding name* as one that would be inherited from the parent type, it *overrides* the one that would be inherited from the parent type.

---

passed-object dummy argument (4.5.1): The first argument of a specific procedure that is bound to a type by a procedure binding that has the PASS_OBJ annotation, and that has the same type as the type to which the procedure is bound. [347:10+]

---

type-bound procedure (4.5.1.5): A procedure that is declared to be associated to a type. It is invoked using a binding name. It is accessed if the type to which it is bound is accessed, and its accessibility is public. [349:26+]