Date: 11 July 1998
To: J3
From: Van Snyder
Subject: Thoughts on generic type-bound procedures
References: 97-230r1, 98-136, 98-140

1 Background

Paper 97-230r1 provided specifications for type-bound procedures; paper 98-136 proposed syntax for type-bound procedures; paper 98-140 was a discussion paper for generic type-bound procedures. This paper proposes specifications for generic type-bound procedures similar to 97-230r1 and 98-140, but with syntax developed partly from 98-140, and partly by extending 98-136. Tentative edits are proposed.

2 Introduction

A generic procedure name can be thought of as the name of a row-vector, in which each element is a specific procedure. The column is selected by the argument characteristics, using generic resolution rules in section 14.1.2.3.

A type-bound procedure name can be thought of as the name of a column vector, in which each element is null or a specific procedure. When constructing the vector, the row is selected by the type to which the procedure name is bound. There are three possibilities for each value:

- Null, if the procedure is declared to be abstract, or
- the same as the element in the row indexed by the parent type, if the type is an extended type and the type-bound procedure name is inherited from the parent type, or
- the specific procedure declared to over-ride the one inherited from the parent type (consider the inherited specific procedure to be null if there is no parent type).

When referencing the vector, the row is selected by the dynamic type (5.1.1.8) of the part name immediately preceeding the type-bound procedure name.

A generic type-bound procedure name can be thought of as the name of a matrix, in which each element is a specific procedure. The column is selected as for a generic procedure name, and the row is selected as for a type-bound procedure name. The values in a column have the same possibilities as for a column vector that represents a type-bound procedure name. In addition to being null because a procedure with the characteristics that select the column is declared to be abstract for a particular type, an element could be null if no specific procedure with the characteristics that select the column is bound by the type-bound procedure name to the type or any of its ancestors.

For example, in `CALL A%B%C (D, E)`, `C` is the name of the matrix, the type of `A%B` is used to select the row (declared type when creating the matrix, dynamic type when referencing it), and the declared (not dynamic) characteristics of `(D,E)` are used to select the column.

In the absence of a `PASS_OBJ` declaration, the characteristics used to select a column include all arguments. In the presence of a `PASS_OBJ` declaration, the characteristics include all dummy

arguments except the dummy argument to which the object that appears before **%** would be associated.

3 Specifications – Summary of 98-140?

Allow several specific procedures to be associated to a single type-bound procedure name, defined operator, or equals sign.

If a procedure is bound to a type by the same name as one inherited from the parent type, the same operator symbol, or the equals sign, and it has the same characteristics as a specific procedure associated to the type-bound procedure name, operator symbol, or equals sign, inherited from the parent type, it overrides that specific procedure's association to the type-bound name, operator symbol, or equals sign, inherited from the parent type. Otherwise it extends the generic collection of specific procedures associated to the name, operator symbol, or equals sign, by which it is bound to the type.

The specific procedure denoted by a type-bound procedure reference is resolved by using the type-bound procedure resolution rules proposed in papers 97-230r1 and 98-136, and the generic resolution rules in section 14.1.2.3.

The semantics proposed here may be slightly different from semantics proposed in paper 98-140. This paper proposes that one may over-ride a subset of the procedures bound to a type, as specified by their characteristics, while inheriting those bound to the same name that are not over-ridden.

4 Syntax – Slightly simpler than 98-140

The *derived-type-def* is augmented to include a procedure bindings section introduced by a `CONTAINS` statement, after all of the component definitions. After the `CONTAINS` statement, specific module procedures, or abstract interfaces, are specified to be bound to

- a name by using a statement of the form `PROCEDURE [[, PASS_OBJ] :: ]` *binding-name* => *binding-list*, or
- a defined operator symbol by using a statement of the form `OPERATOR(`*defined-operator*`)` => *binding-list*, or
- the equals symbol by using a statement of the form `ASSIGNMENT(=)` => *binding-list*.

Within a single derived type definition, each binding name, each defined operator symbol, and each equals symbol establishes a generic interface. If several type-bound procedures have the same binding name, or several defined operations use the same symbol, or several defined assignments are specified, the effect is as if each were established by a single statement (parallel to the case for multiple generic interface blocks having the same name).

Each *binding* shall be the specific name of an accessible procedure pointer, external procedure, dummy procedure, or module procedure, or a declaration that the type-bound procedure, defined operation, or defined assignment for a particular characteristic is *abstract* by being of the form `NULL(` *abstract-interface-name* `)`, where *abstract-interface-name* is the name of an abstract interface, as described in section 12.3.2.1.4.

The syntax proposed here is slightly different from the syntax proposed in paper 98-140, wherein it is proposed to use a `GENERIC` keyword if more than one specific procedure is to be bound to a name. This paper proposes to use the keyword `PROCEDURE` in both cases.

5 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.

[39:4+]

[CONTAINS
[*procedure-binding*]...]

[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[, PASS_OBJ]
or PASS_OBJ, POINTER

Constraint: If PASS_OBJ is specified the *proc-interface* shall have a dummy argument that has the same type as the *type-name*, or an ancestor type (4.5.3) thereof.

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

If PASS_OBJ is specified, the first dummy argument of *proc-interface* that has the same type as the *type-name*, or an ancestor type (4.5.3) thereof, is called the **passed-object dummy argument**. It shall not be a dummy function procedure. The use of PASS_OBJ is explained in [new section] 12.4.1.1.

R432c *procedure-binding* **is** *binding-by-name*
or *binding-by-operator*
or *binding-by-assignment*

R432d *binding-by-name* **is** PROCEDURE [[, PASS_OBJ] ::] ■
■ *binding-name* => *binding-list*

Constraint: If PASS_OBJ is specified each binding shall have a dummy argument that has the same type as the *type-name*, or an ancestor type (4.5.3) thereof.

Constraint: If PASS_OBJ is specified it shall be specified for all procedure bindings, declared within the same type definition or inherited from the parent type, that have the same binding name.

If PASS_OBJ is specified, the first dummy argument of each binding that has the same type as the *type-name*, or an ancestor type (4.5.3) thereof, is the passed-object dummy argument. It shall not be a dummy function procedure. The use of PASS_OBJ is explained in [new section] 12.4.1.1.

R432e *binding-by-operator* **is** OPERATOR(*defined-operator*) => *binding-list*

R432f *binding-by-assignment* **is** ASSIGNMENT(=) => *binding-list*

R432g *binding* **is** *procedure-name*
or NULL(*abstract-interface-name*)

Constraint: Each procedure name shall have an explicit interface and shall refer to an accessible procedure pointer, external procedure, dummy procedure or module procedure.

Constraint: A procedure name that appears in a procedure binding shall not be one that previously had been specified in any procedure binding with the same binding name, defined operation or equals symbol in the same derived type definition or any ancestor type (4.5.3) definition.

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

If two or more bindings have the same identity, the effect is as though they were declared by a single binding.

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

4.5.1.5 Type-bound procedures, operations and assignment

Each binding by name establishes a generic interface ([new section] 12.3.2.1.1), each binding by operator establishes a defined operation ([existing section] 12.3.2.1.1), and each binding by assignment establishes a defined assignment ([existing section] 12.3.2.1.2) for each of the specified procedures.

The term *identity* is probably not ideal. Alternatives are solicited. *Denotation* anyone? *J3 note*

The identity of a binding is the binding name, defined operator, or equals symbol that appears in its declaration. If two procedure bindings have the same identity, the effect is as though all bindings for that identity were specified in a single statement.

The characteristics of a binding are as specified in 12.2; the characteristics of bindings having the same identity shall differ as specified in 14.1.2.3.

The term *interface* is probably not ideal. Alternatives are solicited. *J3 note*

The interface of a binding is its identity and the characteristic of the specific procedure.

Each binding specifies a **type-bound procedure**. If a type is accessible and its components are public, the identities 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. Note – The specific names may be private names.

[Editor: Add the following in the same paragraph. Add **ancestor type** to the index.] [47:36+]

An **ancestor type** of an extended type is its parent type, or an ancestor type of its parent type.

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. Inheritance is transitive: entities inherited by the parent type from its parent type are inherited by an extended type. Additional type parameters, components, and procedure bindings may be declared in the derived type definition for the extended type. [47:39-44]

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.

[An extended type has a subobject name that is the same name and has the same type as its parent type. This is not an additional component; it denotes a subobject that has the parent type and that consists of all of the components inherited from the parent type. If the parent type is an extended type, the first subobject of the subobject denoted by the parent type name is the subobject name of the parent type's parent type. [48:1-12]

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

[Editor: Start a new paragraph, add **override** to the index.] [48:19+]

If a binding (4.5.1.5) has the same identity (4.5.1.5) as one inherited from the parent type, and characteristics indistinguishable according to section 14.1.2.3, the newly specified one **overrides** the one inherited from the parent type; the one inherited from the parent type is not bound to the extended type, or any extension thereof. Other inherited bindings with the same identity are not affected. If a binding has the same identity as one inherited from the parent, but an interface that is distinguishable from any binding inherited from the parent, according to section 14.1.2.3, it extends the generic interface, defined operation, or defined assignment for the identity, and the type in which it is specified.

[Editor: Add a section title] [219:26+]

12.3.2.1.1 Generic interfaces

[Editor: Move to 222:11+] [220:18-20]

[Editor: Delete (mostly moved to 222:20+).] [220:21-22]

An interface block introduced by INTERFACE PROCEDURE() is an **abstract interface block**. [222:20+]

[Editor: set "abstract interface" in bold face type.] [222:22]

---

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*.

---

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 ac-

tual 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: Start a new paragraph] [305:33+]

In a generic interface established by a procedure binding that includes the PASS_OBJ annotation, the dummy argument list is considered not to contain the passed-object dummy argument (5.4.1).

A type-bound procedure is always generic. [306:24+]

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

[Editor: Replace "interface block that provides that" by "generic".] [307:7]

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

[Editor: Replace "interface block that provides that" by "generic".] [307:12-13]

---

ancestor type (4.5.3): The parent type of an extended type, or an ancestor type of its parent type. [341:14+]

---

binding (4.5.1.5): An association, declared within a derived type definition, of a specific procedure to a name, defined operation or equals symbol. [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* or *operator symbol* or equals symbol as one that would be inherited from an ancestor 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 component name, defined operation, or equals symbol. It is accessed if the type to which it is bound is accessed, and the type's components are public. [349:26+]