Subject: Semantics of the select kind construct are not described, and it appears to be a mess to use
From: Van Snyder
References: 98-179, 00-179, 00-195

1 Introduction

The select kind construct, apparently intended to be used within a derived type definition to select different specific procedures to invoke using an object of derived type, depending on the kind parameters, is not described further than providing its syntax. In particular, the relation between select kind, inheritance, and procedure overriding is not described.

Furthermore, if I understand it correctly, it is quite cumbersome to use. Suppose one has a type with three kind parameters, and one anticipates three values for each of those parameters. If one procedure is needed for each combination of kind type parameter values, this results in a requirement to bind 27 procedures to the type. It appears to require 92 statements to do so, using the select kind construct: Three nested select kind constructs are needed. The inner ones needs 8 statements each – the SELECT CASE and END SELECT statements, 3 CASE statements, and 3 procedure declaration statements. Each middle one encloses three of these, and adds five more statements, for a total of 29 statements per middle level case. The outer one has three middle ones, and adds five more statements, for a total of 92 statements. The proposal here would allow to use one statement – albeit perhaps using more than one line, but not 92 lines.

As I understand it, this is a very clumsy explicit replacement for the automatic generic resolution mechanism. (Actually, the intent is to specify how to generate dispatch tables, but the generic mechanism could do that more clearly.)

I propose in this paper to replace the select kind construct with the already-developed generic resolution mechanism.

This strategy has a simple extension to type-bound defined assignment, type-bound defined operators, type-bound derived-type input/output procedures (see 00-179), and type-bound final procedures (see 00-195).

2 Specifications

Several specific procedures may be bound to a type by using one binding name. The specific procedures bound to (not inherited into) a single type-bound procedure name shall be distinguishable according to the rules for unambiguous generic procedure reference (14.1.2.3).

The PASS_OBJ declaration applies to the binding name, and thereby to all of the specific procedures bound to the type, and all of its extensions, by that name, so we don't need to worry about the case that a binding name has PASS_OBJ in the parent type but not in the type being declared, or vice-versa.

The rules for overriding are not much more difficult than in the case of nongeneric type-bound procedures. We don't have an explanation for the semantics of the select kind construct, but I don't think it will be similer than this: If a specific procedure to be bound to a type by a particular binding name is not distinguishable from one bound to the parent by the same name, by using the rules of section 14.1.2.3, it overrides the one inherited from the parent. Otherwise, it

extends the set of procedures accessible by applying the generic procedure resolution mechanism to the binding name.

Now consider procedure invocation. Define the *effective set of procedures* for a type and binding name to be the set of procedures inherited for that binding name from the parent of the type, minus the overridden ones, plus the ones declared in the type. Each procedure in an effective set has a corresponding one in each effective set for each extension type – either the same procedure or one that overrides it. First, one procedure is selected from the effective set of procedures for the declared type of the invoking object and specified binding name, according to the generic resolution rules. Then the corresponding procedure from the effective set for the dynamic type of the invoking object and the same binding name is invoked. From an implementors point of view, there is a separate dispatch table for each distinct generic resolution of a binding name.

3 Syntax

The proposed syntax to specify generic type-bound procedures is to specify non-generic procedure bindings by using the PROCEDURE statement, and generic bindings by using a new GENERIC statement. This has the disadvantage of requiring a new statement, and the advantage that the processor can detect one case in which one mistakenly extends the generic set instead of overriding a non-generic binding – the case when the name is already non-generic.

The PROCEDURE statement is unchanged, and the *proc-binding* is extended to include

R440 *proc-binding* **is** <as at present>
or GENERIC[(*proc-interface-name*)] ■
■ [[, *binding-attr-list*] ::] *binding-name* ■
■ => NULL()
or GENERIC [[, *binding-attr-list*] ::] ■
■ *binding-name* => *procedure-name-list*

A *binding-name* specified in a PROCEDURE statement shall not be the same as any other binding name specified within the same derived type definition, no matter whether specified in a PROCEDURE or GENERIC statement; if it is the same as an inherited one, the present overriding rules apply – no extension of a generic set is permitted. A binding name specified in a GENERIC statement may be the same as the binding name specified in another GENERIC statement, having the same effect as if the *procedure-name-lists* were combined in a single statement.

4 Edits

Edits refer to 00-007r1. 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.

---

proc-binding 44:12-13
[*proc-binding*] ...

[Editor: Delete.] 44:15-16

R440 *proc-binding* **is** PROCEDURE(*proc-interface-name*) ■ 44:17-20
■ [[, *binding-attr-list*] ::] *binding-name* ■
■ => NULL()
or PROCEDURE [[, *binding-attr-list*] ::] ■
■ *binding-name* => *binding*
or GENERIC(*proc-interface-name*) ■
■ [[, *binding-attr-list*] ::] *binding-name* ■
■ => NULL()
or GENERIC [[, *binding-attr-list*] ::] ■
■ *binding-name* => *binding-list*

Constraint: The binding name shall not be the same as a binding name in the parent type that is declared to be NON_OVERRIDABLE.

Constraint: If a binding name is inherited (4.5.3.2) from the parent type, then the binding name inherited from the parent type and the one being declared shall both be declared with GENERIC or both be declared with PROCEDURE.

Constraint: If the binding name is the same as one inherited from the parent type, PASS_OBJ shall be specified if and only if it is specified for the binding of the same name in the parent type. 44:31+

Constraint: If PASS_OBJ is specified for a binding name in one procedure binding within the derived type declaration, it shall be specified in all procedure bindings for that binding name within the derived type declaration.

Constraint: If NON_OVERRIDABLE is specified for a binding name in one procedure binding within the derived type declaration, it shall be specified in all procedure bindings for that binding name within the derived type declaration.

Constraint: If an *access-spec* is specified for a *binding-name*, and it specifies an accessibility different from the default accessibility, the same *access-spec* shall be specified for every GENERIC statement that specifies the same *binding-name* within the type definition.

---

or NULL(*procedure-name*) 44:45
or NULL(*procedure-pointer-name*)

[Editor: Replace "procedure that has" by "procedure. The *procedure-pointer-name* shall be the name of an accessible procedure pointer. The procedure or procedure pointer shall have". After the second "procedure" insert "or procedure pointer".] 44:47

[Editor: Delete.] 45:4-13

[Editor: Replace "deferred" with "a **deferred procedure binding**".] 49:26

---

may override (4.5.3.2) the inherited deferred binding with another deferred binding. 49:29

The same binding name may be used in several GENERIC procedure binding statements within a single type definition. The effect is as if all of the NULL() bindings were specified by NULL(*procedure-pointer-name*) with *procedure-pointer-name* specifying a procedure pointer with the same interface as the *proc-interface-name*, and then all the bindings were specified by a single statement. 49:30+

A procedure binding declared within a derived type definition overrides one inherited from the parent type if: 54:11-16

(1) The binding declared in the type has the same binding name as one inherited from the

parent type, and

(2) it is declared using GENERIC and the specific or deferred procedure to be bound to the type by a particular binding name is not distinguishable, by using the rules of section 14.1.2.3, from one inherited from the parent and bound to the same binding name.

Otherwise, it extends the set of procedures accessible by applying the generic procedure resolution mechanism (14.1.2.4.2$\frac{1}{2}$) to the binding name. If a binding overrides one inherited from the parent, it and the inherited one shall match in the following ways:

[Editor: Delete "in that interface block".] 345:7

[Editor: Delete "in that interface block".] 345:12-13

[Editor: Add a new section. The term *effective set of procedures* is defined here, but not used anywhere other than in this section. I've set it in italic instead of bold face, with the intention that it's not worth putting in the index and glossary. If you want to set it in bold face and put it in the index, that's fine, too. If you set it in bold face, do I owe you a glossary entry?] 346:22+

14.1.2.4.2$\frac{1}{2}$ Resolving type bound procedure references

The *effective set of procedures* for a type and binding name is the set of procedures inherited for that binding name from the parent of the type, minus the overridden ones, plus the ones declared in the type. Each procedure in an effective set has a corresponding one in each effective set for each extension type – either the same procedure or one that overrides it. For purposes of generic resolution, the passed-object dummy argument (4.5.1) of a procedure inherited from the parent type is considered to have the extended type into which it is inherited. Each effective set of procedures is a generic interface.

If a type-bound procedure is specified by *data-ref % binding-name* in a function reference or call statement:

(1) One procedure is selected from the effective set of procedures for the *binding-name* and the declared type of the *data-ref*, according to the generic resolution rules (14.1.2.4.1).

(2) The reference is to the procedure from the effective set, for the *binding-name* and the dynamic type of *data-ref*, that corresponds to the procedure selected in step (1).

If the reference is to a deferred binding, an error condition occurs.

---

deferred procedure binding (4.5.1.5): a *type-bound procedure* binding that specifies the NULL() intrinsic. A deferred procedure binding shall not be invoked. 400:17+