Subject: Problems with FCD
From: Van Snyder

1 Incorrect terminology regarding "companion processor"

"A companion processor" in the last two lines of Note 16.18 at [422:15+5-6] is not quite correct. The "companion processor" could be the Fortran processor itself. For the same reason, "means other than Fortran" isn't quite right.

[Editor: "companion processor" ⇒ "binding label".] 422:15+5-6

2 Unnecessary duplication regarding I/O and C interop

[403:27-28] and [180:11-12] are identical. There is no need to repeat it. Besides, the topic of [403:27-28] (connection of a file to a unit) is not even remotely related to the topic of 15.4 (interoperation with C functions), the subclause in which it appears. Editor: Delete "If a procedure ... processsor dependent (9.4.3)." 403:27-28

3 Incorrect/imprecise terminology regarding interface bodies

[Editor: "in an interface block" ⇒ "by an interface body".] 285:3

4 Accessing private module procedures through a C-interop back door

This is a subquestion of a larger question: Should private module procedures be accessible by way of a binding label?

Assuming the answer is "no," there are at least two forms of syntax to prevent it:

(1) Specify that a private module procedure has no default binding label, and prohibit the syntax that explicitly specifies a binding label from appearing. This is similar to the way that dummy procedures, procedure pointers, and abstract interfaces work.

(2) Require an affirmative specification that a private module procedure has no binding label. This is remotely similar to the way that external and public module procedures work.

4.1 No default binding label

[Editor: Insert "a private module procedure or" after "of".] 279:28

[Editor: "not a dummy procedure" ⇒ "is an external procedure or a public module procedure".] 403:35

[Editor: Add "Otherwise the procedure has no binding label." at the end of the paragraph.] 405:36

4.2 Require affirmative specification that there is no binding label

---

C1236$\frac{1}{2}$ (R1225) A NAME= specifier with the *scalar-char-initialization-expr* having no nonblank characters shall appear if the subprogram defines a private module procedure. 279:29+

[Editor: Insert "and the value of the expression contains any nonblank characters," before "the procedure".] 403:32

[Editor: Insert "If the value of the expression has no nonblank characters the procedure has no binding label." after "specifier."] 403:33

5 Minor technical oversights regarding PROTECTED

We already say at [84:4] that a nonpointer variable with the PROTECTED attribute is not definable outwith the module in which it is declared. The first following change extends that concept to pointer association status. The constraint confines the effectiveness of the TARGET attribute to the module where the variable is declared.

The second change plugs a hole that needs plugging no matter what we do about the relation between PROTECTED and TARGET.

C723$\frac{1}{2}$ (R739) If *variable* has the PROTECTED attribute, the pointer assignment statement shall appear in the module in which *variable* is declared. 143:31+

[Editor: Insert ", PROTECTED" after "INTENT".] 161:19

6 Minor oversight regarding construct association

Either we don't need to say INTENT at [161:19], or we also need to say VALUE. Just to be safe, let's add VALUE instead of deleting INTENT.

[Editor: Insert "VALUE," before "VOLATILE".] 161:19

7 Mistake / oversight regarding pending I/O identifier

In the FCD [206:9] we specify the ID clause for the WAIT statement as

ID = *scalar-int-variable*

which looks like it was just copied from [187:2]. However, Bill Long thinks the text at [206:9] should be

ID = *scalar-int-expr*

In other cases we use *variable* when the object is intent(out) and *expr* when it is intent(in). For READ and WRITE statements, ID is intent(out), but for the WAIT statement it is intent(in). The current spec for ID in WAIT is not consistent with the other I/O options. This seems unnecessarily confusing for users, and there is no good reason to prohibit expressions for ID in the WAIT case. The same issue exists with the INQUIRE statement.

[Editor: "*scalar-int-variable*" ⇒ "*scalar-int-expr*".] 206:9

[The one on [533] is TeX-O-matic].

[Editor: "variable" ⇒ "expression".] 206:18

[Editor: "*scalar-int-variable*" ⇒ "*scalar-int-expr*".] 210:21

[Editor: "variable" ⇒ "expression".] 213:2

8 Proposed technical change to procedure binding labels

Module procedures ought not to have a default binding label equal to the name of the procedure in lower case if BIND(C) is specified, i.e., without a NAME= clause. This causes interoperable module procedures that have the same name in different modules to have the same binding label, which violates 16.1. It is not good language design for the default behavior specified in one section to have significant possibility to violate the provisions of another section, especially if the violation cannot be detected by the processor, and the consequences are potentially catastrophically expensive, no matter how convenient it might be.

To write a module full of callback routines, i.e., routines that will be called from C only by way of function pointers rather than by mentioning their binding labels, one shouldn't need to invent binding labels one hopes will be unique.

8.1 No default binding labels for module or external procedures

Having default binding labels for external procedures but not for module procedures or dummy procedures is a needless irregularity. Nonetheless, one frequently does want a binding label that is the same as the procedure name in lower case, especially for external procedures. Procedures that have BIND(C), i.e. without a NAME= clause, ought not to have a binding label, and a simple syntax, e.g., NAME, ought to be provided to specify that the binding label is the same as the procedure name in lower case. There is no need to do this for nonprocedure entities.

A special syntax that says "no binding label" such as BIND(C,name="") is just too bizarre: Saying nothing gets a default that is frequently harmful, while one has to say something in order to get nothing.

[Editor: Insert "and the value of the expression contains any nonblank characters," before "the procedure".] 403:32

[Editor: "no ... procedure" ⇒ "a NAME specifier and no expression".] 403:34-35

[Editor: Add a sentence at the end of the paragraph: "Otherwise the procedure has no binding label."] 403:36

8.2 Default binding labels only for external procesures

R1225 *proc-language-binding-spec* **is** BIND (C [, NAME [= *scalar-char-initialization-expr*]]) 279:26

C1235$\frac{1}{2}$ (R1225) The *scalar-char-initialization-expr* shall be of default character kind.

[Editor: Delete "=".] 279:27

[Editor: Insert "and the value of the expression contains any nonblank characters," before "the procedure".] 403:32

[Editor: "no ... specifier" ⇒ a NAME specifier and no expression"; "not a dummy procedure" ⇒ "is an external procedure".] 403:34-35

[Editor: Add a sentence at the end of the paragraph: "Otherwise the procedure has no binding label."] 403:36

The proposal in 8.1 makes it easier to create a binding label for a module procedure that is the same as the procedure name in lower case.

If we decide in the future to allow interoperable internal procedures we almost certainly will not want them to have any binding labels. We might want to allow interoperable internal procedures if we allow internal procedures to be actual arguments and/or procedure pointer targets.

Either change leaves 15.4.1 in a state that will not need alteration if we allow interoperable internal procedures that don't have binding labels.

9 Problems with pointer undefinition

At [415:8] we see "A procedure is terminated...." This doesn't have a defined meaning.

At [415:12-16] we see

(c) Is in a named common block that appears in at least one other scoping unit that is in execution,

(d) Is in the scoping unit of a module if the module also is accessed by another scoping unit that is in execution,

(e) Is accessed by host association,

Scoping units are not executable. The term "scoping unit that is in execution" has no defined meaning. Furthermore, scoping units don't overlap.

If a named common block is declared in the scoping unit of a module (which is nonexecutable), and a procedure in that module is in execution but doesn't contain a declaration of the named common block, does the pointer become undefined? Does (e) save you?

Item (e) is prima-facie absurd. Consider the case of the same common block and procedure as in the previous paragraph, but the procedure is NOT executing. Host association is defined in terms of subprograms, not procedures, so it is clearly a static text-only thing, not a dynamic only-if-the-procedure-is-executing thing. I.e., the pointer is still accessed by host association, so does the pointer never become undefined? Consider the case of an interface body that contains an IMPORT statement that names the pointer. Does this prevent undefinition of that pointer's association status — everywhere?

If an outer scoping unit contains an inner one that is in execution, is the outer one considered to be "in execution" even if it's not a procedure? Or vice-versa? I can't find any support for this (nor even for a procedure being "in execution"). The vice-versa would be absurd. Suppose you put a common block declaration in an interface body in a procedure, but not in the procedure itself. Does this prevent a pointer declared in the common block from losing its association status so long as the procedure in which the interface block appears is executing? Probably not. Further, suppose the common block isn't really used to specify the characteristics of the interface. [415:12-13] says "appears in," not "appears in unless it doesn't count because it doesn't contribute to the characteristics of the interface."

We can fix (c-e) by adding a new paragraph and then rewriting (c-e):

A scoping unit is **in execution** if an executable statement within it is being executed. 15:25+ New ¶

(c) Is in a named common block that is declared in or accessed by host or use association in at least one other scoping unit that is in execution, 415:12-16

(d) Is declared in or accessed by host or use association in at least one other scoping unit that is in execution,

[This will cover submodules, while "within" instead of "host association" would not.]

---

in execution (2.3.4) : A scoping unit is in execution if an executable statement within it is being executed. 430:26+

10 Incorrect terminology

[Editor: In the first line of note 12.9 "operators" ⇒ "operations". 262:21+2

11 Minor problem with syntax of Function subprogram.

In the light of [280:38-40], the *execution-part* cannot be optional at [279:17]. This is an old problem.

---

execution-part 279:17

12 Require the TARGET attribute for a dummy procedure to be a pointer target

Do we want to reqire the TARGET attribute in order for a dummy procedure that is not a procedure pointer to be a *proc-target*? The reason to do so is that we may in the future want to allow internal subprograms as actual arguments, but not as procedure pointer targets. If we specify now that a dummy procedure that is not a procedure pointer has to have the TARGET attribute in order to be a *proc-target* we could later allow an internal subprogram to be an actual argument so long as the interface of the proceedure to which it is passed is explicit, and the associated dummy argument does not have the TARGET attribute. If we don't say so now, we can never say so in the future, which means that if we allow internal subprograms to be actual arguments, we will have to allow them to be procedure pointer targets as well.

Van says: "Even though I am the author of this question, I am opposed to making the following change. I just want to get it on the table now so nobody can legitimately use the failure to do so as an argument in the future against allowing internal subprograms as actual arguments."

Proposed change:

[Editor: "object" ⇒ "object or dummy procedure" twice.] 84:21

[Editor: "*object-name*" ⇒ "*entity-name*" twice.] 92:4-5

C570$\frac{1}{2}$ (R546) If *entity-name* is not an object name, it shall be a dummy procedure that does not have the POINTER attribute. 92:5+

C727$\frac{1}{2}$ (R742) If *procedure-name* is a dummy procedure that does not have the POINTER attribute, it shall have the TARGET attribute. 144:6+

or TARGET 264:18+

C1218$\frac{1}{2}$ (R1213) If TARGET appears every *procedure-entity-name* shall be the name of a dummy procedure that does not have the POINTER attribute. 264:35+

[Since we don't allow the POINTER attribute to be specified in an EXTERNAL statement, We shouldn't allow to specify the TARGET attribute there, either.]