Date: 19 August 1997
To: J3
From: Van Snyder
Subject: Edits for procedure pointers
References: 97-147 97-169 97-174r1 97-190 97-218r2 (syntax)

Changes are w.r.t. 97-007r1. Page and line numbers are displayed in the margin. There are several J3 notes to which special attention should be directed.

[part of R207] **or** *proc-declaration-stmt* [10:9+]

[part of R425] **or** PROCEDURE ([*proc-interface*]) ■ [38:40+]
■ , POINTER :: *proc-identity-list*

5.2 Procedure declaration statement [59:12+]

[Note to editor: re-number subsequent sections]

A procedure declaration statement declares a procedure pointer, a dummy procedure, an external procedure, or a module procedure.

[Note to editor: Syntax rule numbers are to be inserted between present rules R519 and R520.]

R519A *proc-declaration-stmt* **is** PROCEDURE ([*proc-interface*]) ■
■ [[, *proc-attr-spec*] ... ::] *proc-identity-list*

R519B *proc-interface* **is** *abstract-interface-name*
or *type-spec*

Constraint: *abstract-interface-name* must be the name of an abstract interface (12.3.2.1.3)

R519C *proc-attr-spec* **is** *access-spec*
or INTENT (*intent-spec*)
or POINTER
or SAVE
or OPTIONAL

Constraint: If a *proc-identity* has an accessibility attribute, or an INTENT attribute, or the SAVE attribute, the POINTER attribute shall also be specified for that *proc-identity*.

R519D *proc-identity* **is** name [=> NULL()]

Constraint: If => NULL() appears then the POINTER attribute shall be specified for the corresponding name.

Constraint: *proc-identity* shall not be the name of an accessible module procedure.

Constraint: *proc-identity* shall not appear in a type declaration statement. *Why?*

The only attributes allowed for *proc-identity* are *access-spec*, INTENT, POINTER, SAVE and

OPTIONAL.

There appears to be no place where the allowed attributes for named things are listed. If there is such a place, should the previous paragraph be in that place? Do we need the previous paragraph at all? *J3 note*

The following table indicates the category of entity named by *proc-identity*:

Is the POINTER attribute specified for *proc-identity*?	Is *proc-identity* the name of a dummy argument?	Then *proc-identity* is:
Yes	Yes or No	Procedure pointer
No	Yes	Dummy procedure
No	No	External proceure

Appearance of an intrinsic procedure name in a PROCEDURE statement causes that name to become the name of an external procedure and thus the intrinsic procedure of the same name is not available in the scoping unit.

Appearance of *proc-identity* in a PROCEDURE statement specifies the EXTERNAL attribute (12.3.2.3) for that name.

If *proc-interface* consists of *abstract-interface-name*, *proc-identity* has explicit interface, and shall be used only to identify procedures having characteristics given by the named abstract interface.

If *proc-interface* consists of *type-spec*, *proc-identity* has implicit function interface, and shall be used only to identify functions that have the result type given by *type-spec*.

If *proc-interface* is absent, *proc-identity* has ambiguous interface (12.3.1), and shall be used only to identify subroutines, or dummy procedures that are used as actual arguments to procedures with implicit interface, or as actual arguments to procedures with explicit interface in which the corresponding dummy procedure is a subroutine or has ambiguous interface.

It is not possible to use a PROCEDURE statement to identify a BLOCK DATA subprogram. *Note 5.a*
Why not?
Note 5.b

```
! Using abstract procedure definitions in Note 12.x:
!-- Some external or dummy procedures with explicit interface.
PROCEDURE (REAL_FUNC) :: BESSEL, GAMMA
PROCEDURE (SUB) :: PRINT_REAL

!-- Some procedure pointers with explicit interface,
!-- one initialized to NULL().
PROCEDURE (REAL_FUNC), POINTER :: P, R => NULL()
PROCEDURE (REAL_FUNC), POINTER :: PTR_TO_GAMMA

!-- A derived type with a procedure pointer component ...
TYPE STRUCT_TYPE
```

```
    PROCEDURE (REAL_FUNC), POINTER :: COMPONENT
  END TYPE STRUCT_TYPE

  !-- ... and a variable of that type.
  TYPE(STRUCT_TYPE) :: STRUCT

  !-- An external or dummy function with implicit interface
  PROCEDURE (REAL) :: PSI
```

[sentence that begins "This also..."] [59:17]

.... This also applies to PROCEDURE, EXTERNAL and INTRINSIC statements.

Should we allow or prohibit procedure pointers to appear in COMMON blocks? [71:2+]

J3 note

Constraint: Each *allocate-object* shall be a non-procedure pointer or an allocatable array. [81:36]

Constraint: Each *allocate-object* shall be a non-procedure pointer or an allocatable array. [84:23]

[Note to editor: Replace first sentence with this stuff, then start a new paragraph with "If the *target* is not ...."] [113:7]

If *pointer-object* is a procedure pointer, *target* shall be the name of an accessible external, module, dummy or intrinsic procedure, a procedure pointer, a reference to a function that returns a procedure pointer, or a reference to the NULL intrinsic function. The only intrinsic procedures permitted are those listed in 13.13 and not marked with a bullet (•). If the specific intrinsic procedure name is also a generic name, only the specific intrinsic procedure is associated with *pointer-object*.

If *pointer-object* has explicit interface, *target* shall have the same characteristics.

If *pointer-object* has function interface, *target* shall have the same result type.

If *pointer-object* has ambiguous interface, *target* shall be the name of a subroutine, an external or dummy procedure, or a procedure pointer, that does not have function interface.

If *pointer-object* is not a procedure pointer, *target* shall have the same type parameters as *pointer-object*.

[inside note 7.46] [113:34+]

```
! P is a procedure pointer and BESSEL is a
! procedure with compatible interface (see note 5b)
P => BESSEL

! Likewise for a structure component
STRUCT % COMPONENT => BESSEL
```

Constraint: A variable that is an input item shall not be a procedure pointer. [151:12+]

Constraint: An expression that is an output item shall not have a value that is a procedure pointer. [151:14+]

Change title to **12.2.1.1 Characteristics of dummy data objects other than procedure pointers.** [198:12]

Change title to **12.2.1.2 Characteristics of dummy procedures and dummy procedure pointers.** [198:18]

Change title to **12.3.1 Implicit, explicit and ambiguous interface.** [198:37]

The interface for an external procedure, dummy procedure or procedure pointer is **ambiguous**, in that one cannot know whether it is a function or subroutine if [199:5+]

1. It is defined only by an EXTERNAL statement, or by a PROCEDURE statement having no *proc-interface*, and
2. It does not appear in a type statement, and
3. A reference to its name does not appear as a function or subroutine reference.

A procedure or procedure pointer has **function interface** if it has explicit interface and is a function, or its name is explicitly typed, or a reference to its name appears as a function reference.

Remove the word MODULE. [199:33]

[replace second line of R1202] **or** *procedure-stmt* [199:40]

[add a line to R1203] **or** INTERFACE ■ [199:41+]
■ PROCEDURE ()

Constraint: If *interface-stmt* is INTERFACE PROCEDURE(), each *interface-specification* shall be an *interface-body*.

[replace R1206] [200:8]
R1206 *procedure-stmt* **is** [MODULE] ■
■ PROCEDURE ■
■ *procedure-name-list*

Constraint: A *procedure-name* shall have explicit interface and shall refer to an accessible procedure pointer, external procedure, dummy procedure or module procedure. [200:22-24]

Constraint: If MODULE appears, *procedure-name* shall refer to an accessible module procedure.

At present the only "chinks in the armor" of the "no host association into interface blocks" policy are that module, external and dummy procedure names are nonetheless visible inside them. The *J3 note*

present changes would also allow procedure pointers. It would also be valuable to allow type names to be visible inside them. In any case, it would be valuable to list exactly what is and is not visible inside them, either in 12.3.2.1 or 14.6.1.3.

Constraint: A *procedure-stmt* is allowed only if the interface block has a *generic-spec*.

Constraint: In any specification part, A *procedure-name* shall not be specified more than once in a *procedure-stmt*, nor be the same as a function name or subroutine name that appears in a *function-stmt* or *subroutine-stmt* in all interface blocks that have the same generic identifier.

[after note 12.6] [201:46+]

An interface block introduced by INTERFACE PROCEDURE() is an **abstract interface block**; it defines **abstract interfaces**.

[after note 12.9] [203:18+]

12.3.2.1.3 Abstract interfaces

The name given in a *subroutine-stmt* or *function-stmt* in an abstract interface block is the name of an abstract interface. Abstract interface names are in the same class as type names (14.1.2)

Note 12.x

```
! Example abstract interfaces.
INTERFACE PROCEDURE()
  ! REAL_FUNC IS ABSTRACT INTERFACE NAME
  FUNCTION REAL_FUNC (X)
    REAL, INTENT(IN) :: X
    REAL :: REAL_FUNC
  END FUNCTION REAL_FUNC
  ! SUB IS ABSTRACT INTERFACE NAME
  SUBROUTINE SUB (X)
    REAL, INTENT(IN) :: X
  END SUBROUTINE SUB
END INTERFACE
```

[Move 203:33-34 here.] [203:28-44]

[Move notes 12.10 and 12.11 (203:39-44) here].

Appearance of an intrinsic procedure in an EXTERNAL statement causes that name to become the name of an external or dummy procedure and thus the intrinsic procedure of the same name is not available in the scoping unit.

It is generally better practice to declare an external or dummy procedure by using a PROCEDURE statement, as this allows the interface to be specified in the same place. *Note 12.y*

Is *external-name* allowed to be the name of an accessible module procedure (see [194:14-17])? *J3 note*

12.3.2.3 EXTERNAL attribute

[Note to editor: Re-number subsequent sections.]

A name is specified to have the EXTERNAL attribute if it appears in an EXTERNAL statement, a PROCEDURE statement, or as a specific procedure name in an interface block.

A name may be specified to have the EXTERNAL attribute only once in a scoping unit.

Or is the name of a module procedure? [204:31+]

[add a line to R1210] **or** *variable* ■
■ ([*actual-arg-spec-list*])

Constraint: *variable* shall be a procedure pointer, or a structure component that is a procedure pointer. A reference to *variable* shall not appear as a subroutine reference. The pointer association status of *variable* shall not be undefined.

[add a line to R1211] **or** CALL *variable* ■
■ ([*actual-arg-spec-list*]) [204:33+]

Constraint: *variable* shall be a procedure pointer, or a structure component that is a procedure pointer. The pointer association status of *variable* shall not be undefined.

Examples of procedure reference using procedure pointers. [205:23+]
Note 12.14a

```
P => BESSEL
WRITE (*, *) P(2.5)         ! -- BESSEL(2.5)

S => PRINT_REAL
IF (ASSOCIATED(S)) CALL S(3.14)
```

12.4.1.2 Actual arguments associated with dummy prodedures or dummy procedure pointers [208:16:30]

If the dummy argument is a procedure pointer, the associated actual argument shall be a procedure pointer.

If the dummy argument is a dummy procedure, the associated actual argument shall be the specific name of an external, module, dummy, or intrinsic procedure, or a procedure pointer. The only intrinsic procedures permitted are those listed in 13.13 and not marked with a bullet (•). If the specific name is also a generic name, only the specific procedure is associated with the dummy argument.

If an external procedure name or a dummy procedure name is used as an actual argument, its interface shall be explicit or it shall be declared in an EXTERNAL or PROCEDURE statement.

If the dummy argument has explicit interface, the characteristics listed in 12.2 shall be the same for the associated actual argument and the corresponding dummy argument, except that an actual argument having an interface to a pure procedure may be associated with a dummy argument having an interface to a procedure that is not pure, and an actual argument having an interface to

an elemental intrinsic procedure may be associated with a dummy argument having an interface to a procedure that is not elemental.

If the dummy argument has implicit interface and either the name of the dummy argument is explicitly typed or the dummy argument is referenced as a function, the dummy argument shall not be referenced as a subroutine, and the actual argument shall have ambiguous interface, or function interface and the same result type as the dummy argument.

If the dummy argument has implicit interface, and a reference to the dummy argument appears as a subroutine reference, the actual argument shall not have function interface.

POINTER	shall be a pointer and may be of any type, or a procedure pointer. Its pointer association status shall not be undefined.	[238:17-21]
TARGET (optional)	shall be a pointer or target. If POINTER is a data entity, TARGET shall have the same type, type parameters and rank as POINTER. If POINTER is a procedure pointer, TARGET shall be a procedure, or procedure pointer, for which pointer assignment (7.5.2) to POINTER would be permitted. If TARGET is a pointer then its association status shall not be undefined.	

Case (ii): If POINTER is a procedure pointer and TARGET is an external procedure, module procedure, intrinsic procedure or dummy procedure, the result is true if POINTER is associated with TARGET. [238:25+]

Case (iii): If POINTER is a procedure pointer and TARGET is a procedure pointer, the result is true if POINTER and TARGET are associated with the same procedure.

[Note to editor: re-number subsequent cases.]