Date: 30 October 1997
To: J3
From: Van Snyder
Subject: Alternative to 97-248 – Edits for procedure pointers
References: 97-147 97-169 97-174r1 97-190 97-218r2 (syntax) 97-248 (edits)

This paper proposes an alternative syntax for procedure pointers, and edits that implement it. It's based exclusively on extending the EXTERNAL attribute and statement.

Yes, I know we considered this and rejected it in /data, and passed something different in plenary, but we weren't writing edits at the time. I think we only narrowly rejected using EXTERNAL in favor of inventing something new. Upon finishing 97-248r1, I realized the only thing that can be done with EXTERNAL that couldn't be done with PROCEDURE is to declare that a block data program unit is part of the program. So it seemed we might want to add that capability to PROCEDURE, and then deprecate EXTERNAL. But if we do that, then why not change the spelling back to EXTERNAL so old codes still work? This paper is the result of taking that thought process to its logical conclusion.

I think this integrates better than 97-248.

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

Add "(5.1.2.10)"after "EXTERNAL attribute" [2:36]

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

[Complete replacement just to alphabetize the right-hand-sides while adding EXTERNAL] [38:41-43]
R426 *component-attr-spec* **is** ASYNCHRONOUS
or DIMENSION (*component-array-spec*)
or EXTERNAL
or POINTER

Constraint: If a component has the EXTERNAL attribute it shall also have the POINTER attribute [39:18+]

Delete ", EXTERNAL," [48:8]

Constraint: If a name has the EXTERNAL attribute, the only other attributes allowed are *access-spec*, INTENT, OPTIONAL, POINTER and SAVE. [48:22+]
Constraint: If an entity has the EXTERNAL atribute, and in addition it has an accessibility attribute, or an INTENT attribute, or the SAVE attribute, it shall also have the POINTER attribute.

Delete ", an external name" [48:34]

Replace "the POINTER" by "either the POINTER or the EXTERNAL" [48:37]

A name is specified to have the **EXTERNAL attribute** if it appears in an EXTERNAL statement (5.2) or in a type declaration statement having the EXTERNAL attribute specifier. [58:28-30]

A name that is specified to have the EXTERNAL attribute must be an external procedure, or a procedure pointer, or a block data subprogram.

A name that has the EXTERNAL attribute and also has an explicit type, or that appears as a function name in a function reference (12.4), or that appears as a function name in an interface body (12.3.2.1), is an external function or dummy function.

A name that is not the name of a block data subprogram and has the EXTERNAL attribute, or is the name of an accessible module procedure, or appears as a procedure name in a procedure reference (12.4) or an interface body, may be used as an actual argument (12.4.1.2), as a procedure name in a procedure reference (12.4), or as the target of a procedure pointer assignment (7.5.2).

If we add "accessible module procedure" to the list in the first paragraph in this section, we needn't mention it in the previous paragraph. *J3 note*

A name for which the EXTERNAL attribute has been declared in a given scoping unit or that is a use-associated entity with the EXTERNAL attribute shall not also appear as a specific procedure name in an interface body that is in or accessible to the scoping unit.

If a name has both the EXTERNAL and the POINTER attributes, it is a **procedure pointer**. *Index term*

If a name has the EXTERNAL attribute, an intrinsic procedure of the same name is not available in the scoping unit.

```
!-- An external or dummy function with implicit interface
REAL, EXTERNAL :: PSI
```
Note 5.c

[Preferred replacement for above. Does this work?] [58:28-30]

A name is specified to have the EXTERNAL attribute if it appears in an EXTERNAL statement, in a type statement with the EXTERNAL attribute specifier, as a procedure name in an interface body, or is the name of an accessible module procedure. The EXTERNAL attribute is implied and may be confirmed by explicit specification if the name is not the name of an internal procedure, and it appears as a procedure name in a procedure reference (5.4).

The EXTERNAL attribute shall be specified by an EXTERNAL statement only if the name is the name of an external or dummy subroutine, or a block data program unit. The EXTERNAL attribute shall be specified by the EXTERNAL attribute specifier in a type statement only if the entity is an external or dummy function.

A name that has the EXTERNAL attribute and also has an explicit type, or appears as a *function-name* in a procedure reference, is a function name.

A name that is not the name of a block data subprogram and has the EXTERNAL attribute may

be used as an actual argument (12.4.1.2), as a procedure name in a procedure reference (12.4), or as the target of a procedure pointer assignment (7.5.2).

If a name has both the EXTERNAL and the POINTER attributes, it is a **procedure pointer**. *Index term*

If a name has the EXTERNAL attribute, an intrinsic procedure of the same name is not available in the scoping unit.

```
!-- An external or dummy function with implicit interface
REAL, EXTERNAL :: PSI
```
Note 5.c

Other simplifications may arise elsewhere by using the alternative wording. *J3 note*

5.2 EXTERNAL statement

[59:12+]

[Note to editor: re-number subsequent sections]

An EXTERNAL statement specifies the EXTERNAL attribute for a list of names, or declares procedure pointers.

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

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

R519B *proc-interface* **is** *abstract-interface-name*

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

R519C *proc-identity* **is** name [=> NULL()]

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

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 procedure or Block data subprogram

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

If *proc-interface* is absent but *proc-identity* appears in a type declaration statement, *proc-identity* has implicit function interface, and shall be used only to identify functions that have the declared result type.

The appearance of the name of a block data program unit in an EXTERNAL statement confirms that the block data program unit is a part of the program.

For explanatory information on potential portability problems with external procedures, see section *Note 5.a*

C.9.1.

```
!-- An external or dummy procedure with unspecified interface, or a
!-- block data subprogram
EXTERNAL :: FOCUS

!-- An external or dummy function with implicit interface
REAL, EXTERNAL :: PSI

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

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

!-- A derived type with a procedure pointer component ...
TYPE STRUCT_TYPE
  EXTERNAL (REAL_FUNC), POINTER :: COMPONENT
END TYPE STRUCT_TYPE

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

Note 5.b

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* is a procedure pointer that has explicit interface, *target* shall have the same characteristics.

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

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 5.b)
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]

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. [199:5+]

Remove the word MODULE. [199:33]

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

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

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.

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

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

[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
```

Delete section 12.3.2.2. Change all references to it to refer to 5.2. [203:19-44]

Change second "procedure" to "procedure or procedure pointer". [204:28]

[add a line to R1210] **or** *variable* ([*actual-arg-spec-list*]) [204:31+]

Constraint: *variable* shall be a procedure pointer, or a structure component that is a procedure pointer.

Constraint: A reference to *variable* shall not appear as a subroutine reference. *Needed?*

Constraint: 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.

Constraint: 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 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.]