Date 13 August 1997
To: J3
From: /data
Subject **Procedure pointer syntax**

/data has reached consensus on both declaration and operational syntax for changeable procedure identities.

Shaded parts are new.

In syntax rules, where we write "Add syntax" we frequently take the liberty to repeat the left-hand-side and "is" instead of using "or" (without the left-hand-side) or re-writing the entire rule, so the reader will be in context. The original content of the syntax rule is not to be removed.

Named abstract interface declarations

An abstract interface is a variation on an interface block

Add syntax:
R1207 *generic-spec* **is** PROCEDURE()

Constraint
If *generic-spec* is PROCEDURE() then each *interface-specification* shall be an *interface-body*.

Add explanatory text:
The name given in a *subroutine-stmt* or *function-stmt* in an *interface-body* in an *interface-block* with *generic-spec* of PROCEDURE() is the name of an abstract interface. Abstract interface names are in the same class as type names (14.1.2).

Add note:
```
! Example abstract interface.
interface procedure()
     function real_func(x)     ! real_func is abstract
                               ! interface name
          real, intent(in) :: x
          real :: real_func
     end function real_func
     subroutine sub(x)    ! sub is abstract interface
                          ! name
          real, intent(in) :: x
     end subroutine sub
end interface
```

Procedure identifier declarations

Three varieties of procedure identifiers may be declared by using a PROCEDURE statement:

1. External procedures (not changeable)
2. Dummy procedures (not changeable)
3. Procedure pointers (changeable procedure identities)

Add syntax:

R207	*declaration-construct*	**is**	*procedure-declaration-stmt*

R425	*component-def-stmt*	**is**	*component-proc-decl-stmt*

R427A	*component-proc-decl-stmt*	**is**	PROCEDURE ([*procedure-interface*]) & & , POINTER :: *procedure-identity-list*

R50w	*procedure-declaration-stmt*	**is**	PROCEDURE ([*procedure-interface*]) & & [[, *procedure-attr-spec*] ... ::] & & *procedure-identity-list*

R50x	*procedure-interface*	**is**	*abstract-interface-name*
		or	*type-spec*

Constraint:

abstract-interface-name shall be the name of an abstract interface.

R50y	*procedure-attr-spec*	**is**	*access-spec*
		or	INTENT (*intent-spec*)
		or	POINTER
		or	SAVE

Constraint:

If *access-spec* or INTENT or SAVE is specified then POINTER shall also be specified.

R50z	*procedure-identity*	**is**	*name* [=> NULL()]

Constraint:

If `=> NULL()` appears the POINTER attribute shall be specified.

Add explanatory text:

If POINTER is not specified the names declared are external procedures or dummy procedures. If POINTER is specified the names declared are procedure pointers.

If *procedure-interface* consists of *abstract-interface-name* then the *procedure-identity* has explicit specific interface given by the named abstract interface.

If *procedure-interface* consists of *type-spec* then the *procedure-identity* identifies a function that has implicit interface and the specified return type.

If *procedure-interface* is absent then the *procedure-identity* identifies a subroutine that has implicit interface.

It is not possible to use a PROCEDURE statement to identify a procedure that is ambiguous concerning whether it is a subroutine or function.

It is not possible to use a PROCEDURE statement to identify a BLOCK DATA subprogram.

Add note:

```
!-- Some external or dummy procedures with explicit
!-- specific interface.
procedure(real_func) :: bessel, gamma
procedure(sub) :: print_real

!-- Some procedure pointers with explicit specific
!-- interface, one initialized to null.
procedure(real_func), pointer :: p, r => null()
procedure(real_func), pointer :: ptr_to_gamma
procedure(sub), pointer :: s

!-- A derived type with a procedure pointer component...
type struct_type
     integer :: some_int
     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
```

Procedure identifers can be in generic interface blocks

Consistent with the possibility to put dummy procedures into generic interface blocks, procedure identifiers can be referenced (not declared) in generic interface blocks. Add syntax:

R1202 *interface-specification* **is** PROCEDURE [::] *procedure-name-list*

Constraint:

procedure-name shall have an explicit interface and shall be a procedure pointer, external procedure, dummy procedure or module procedure.

Functions that return procedure identifiers

Functions can return procedure identifiers, by defining the function or result name to be a procedure pointer.

Assigning values to procedure pointers

Values are assigned to procedure pointers by using pointer assignment.

Add a constraint after R737:

> If *pointer-object* is a procedure pointer then *target* must have an interface compatible to the interface for *pointer-object*, and must be the name of an accessible external, module, dummy or intrinsic procedure (the same list of intrinsics as are allowed to be actual arguments), a procedure pointer or a reference to a function that returns a procedure pointer.

Add a note:

```
!-- Give p a non-null value.  p must be a procedure
!-- pointer
p => bessel

!-- Likewise for a structure component.
struct%component => bessel
```

Testing procedure pointers

Procedure pointers can be tested using the ASSOCIATED intrinsic function.

```
!-- Test for equality.
if (associated(p,struct%component)) &
write(*,*) 'This should print.'

!-- Test for NULL
if (.not. associated(r)) &
write(*,*) 'This should print.'
```

Add text to the description of ASSOCIATED that allows TARGET to be the same kinds of things allowed for *target* in a *pointer-assignment-stmt* (including accessible procedures).

Invoking procedures defined by procedure pointers

Procedures defined by procedure pointers are invoked by using a CALL statement or *function-reference*.

Add syntax:

R1210 *function-reference* **is** *variable* ([*actual-arg-spec-list*])

Constraint:

> *variable* shall be a procedure pointer to a function, or a structure component that is a procedure pointer to a function.

Add syntax:

R1211 *call-stmt* **is** CALL *variable* [([*actual-arg-spec-list*])]

Constraint:

> *variable* shall be a procedure pointer to a subroutine, or a structure component that is a procedure pointer to a subroutine.

Add note:

```
!-- Evaluate functions.
write (*,*) p(2.5)                    !-- bessel(2.5)
write (*,*) struct%component(2.5)   !-- Also bessel(2.5)

!-- Some subroutine operations.
s => print_real
if (associated(s)) call s(3.14)
```

Using procedure identities as actual arguments

All procedure identities (external procedure identities, dummy procedure identities, or procedure pointers) and procedure values (results of function evaluation) can be used as actual arguments. Add text in 12.4.1.2 to allow this. Add a section 12.4.1.3 describing the case when the dummy argument is a dummy procedure pointer (and re-number existing 12.4.1.3 ff).

Add note:

```
!-- Pass as an actual argument.
call integrate (p, 1., 2.)

!-- Invoke a function returning a proc value.
ptr_to_gamma => gamma
r => select_func(2, p, ptr_to_gamma)  !-- r is now gamma

!-- A fairly complicated composition.
     call integrate (select_func(1,p,r), 1., 2.)
! ...
contains

     subroutine integrate (func, from, to)
          procedure(real_func), intent(in) :: func
          real, intent(in) :: from, to

          if (.not. associated(func)) &
               call abort('Oops.')
          write (*,*) 'End values are ', &
                func(from), func(to)
          return
     end subroutine integrate

     function select_func(n, proc1, proc2)
          integer, intent(in) :: n
          procedure(real_func), intent(in) :: proc1
          procedure(real_func), intent(in) :: proc2
          procedure(real_func), pointer :: select_func

          select case(n)
          case(1)
               select_func => proc1
          case(2)
               select_func => proc2
          case default
               select_func => null()
          end select
          return
      end function select_func
```

Dummy arguments

Dummy arguments may be procedure pointers. The actual argument must also be a procedure pointer.

Input / Output

Intrinsic input/output of procedure identities is prohibited. No text is needed w.r.t. derived type I/O because of the presence of the POINTER attribute. Somewhere after R918 add

Constraint:
A variable that is an input item or output item shall not be a procedure pointer.

An expression that is an output item shall not have a value that is a procedure pointer.