Subject: Technical issues in the submodules TR
From: Van Snyder

Introduction

The purpose of this paper is to describe my current thoughts concerning the TR on submodules, to stimulate discussion of the issues, and to solicit your opinions.

The issues are

(1) What sort of syntactic device ought to be used to indicate that a procedure's body is in a submodule?
(2) What should be the scoping and association relations between a submodule and its parent, between a procedure body's definition and its declaration, and between a procedure's declaration or definition and its containing module or submodule?
(3) What should be the syntax of a procedure declaration and the definition of its body?

1 Syntactic device

My present thought is that an interface body ought to be used to specify the interface of a procedure for which the body is in a submodule.

The argument for doing this is that if one wants a generic for which the bodies are in submodules, the proposal presented at the London WG5 meeting, N1434, required one to put `module procedure` statements in interface blocks, and then put the interface itself in a different structure. N1434 proposed that the syntax of the interface would be identical to an interface body, but it would be called something different, so as to allow it to access its environment by host association.

N1434 proposed that the submodule containing a procedure's body shall be named in the declaration. This isn't necessary, and in some cases may be undesirable. My present thoughts are that this ought to be optional.

N1434 proposed that a submodule shall specify its parent. This is necessary, so no change is proposed here.

2 Scoping and association relations

N1434 proposed that a submodule would access its parent by host association, while a procedure body would be an extension of its declaration. This leads to confusion, and to ambiguity if we find a way to allow the interface to be redeclared in the body.

A better model is that the relation of a submodule to its parent, and the relation of a procedure body to its declaration, is inheritance association. This prohibits redefinition, which prevents confusion and ambiguity. Not being types, neither a submodule nor a procedure body need worry about a "parent component," so a little bit of word smithing will be needed to adjust the description of inheritance association.

N1434 was silent on the relation of the procedure declaration and body to their containing module and submodule. My thought at the time of writing that paper is that it was obvious this relation ought to be host association; anything else would be inconsistent with current practice.

3 Syntax of procedure headers

N1434 proposed that the procedure header in the declaration of a procedure interface for which the body is in a submodule would include a prefix of the form SUBMODULE (*submodule-name*). This paper proposes that the *submodule-name* specification should be optional.

N1434 proposed that the procedure header in the definition of a procedure body includes a SUBMODULE prefix. No change is proposed here.

One can tell the difference between a procedure declaration and a procedure body, both beginning with SUBMODULE, because the former is in an interface block, and the latter is not.

One consequence of the specification of the submodule name being optional is that one could put the interface declaration and the body definition in the same scoping unit. This is a natural degenerate case that would be more effort to prevent than to allow, both in the standard and in compilers.

4 Host association and interface bodies

I was not a party to the debates in 1986-1990 that resulted in interface bodies not accessing their environments by host association. In retrospect, I believe the controversy arose from failing to realize that there were then two kinds of interface bodies: One for external procedures, and one for dummy procedures. The argument that apparently carried the day is that an interface body for an external procedure ought to have exactly the same interpretation as the same statements would have in the external procedure.

It has become clear that certain quite reasonable things are impossible if interface bodies for dummy procedures do not access their environment by host association, e.g., the dummy procedure cannot have an argument for which the type is private. The argument that the interface body ought to have the same semantics as the procedure it represents doesn't carry through for interfaces for dummy procedures, because they might be module procedures, which *do* access their environment by host association.

Development of procedure pointers and object-oriented programming support led to abstract interfaces. Use of abstract interfaces, to support deferred type-bound procedures or for procedure pointers with the PASS_OBJ attribute, is impossible if abstract interfaces do not access their environment by host association.

This led to development of the IMPORT statement.

It would be a bit bizarre to accept that the interface body for an external procedure ought to have the same semantics as the same statements would have in the external procedure, but that the interface body for a submodule procedure *ought not* to have the same semantics as the declaration of the same procedure when not developed as a submodule procedure.

A better solution would be to specify that interface bodies other than those for external procedures access their environment by host association. This would be a minor change from Fortran 95, in that interface bodies for dummy procedures would have potentially different interpretation. The only opportunity for difference, however, is the case of an interface body containing an implicitly-typed entity that becomes explicitly typed with a type different from what the implicit type would be by accessing an entity that determines its type, by host association.

The IMPORT statement should be retained for use in interface bodies for external procedures, because it provides useful functionality – one wouldn't need to put a derived type into a separate module from the interface body, so that it could be accessed by use association.

5 Example

```
module M
  integer, parameter :: RK = kind(1.0d0)
  interface
    submodule(s1) subroutine R1 ( A1, A2 ) ! Body better be in S1!
      real(rk) :: A1          ! Notice that RK is accessed by host association
      integer :: A2
    end subroutine R1
    submodule subroutine R2 ( B1 ) ! Doesn't say which submodule
      real :: B1                   ! the body is in
    end subroutine R2
    submodule subroutine R3 ( C1 )
      integer :: C1
    end subroutine R3
  end interface
contains
  submodule subroutine R3 ! ( C1 ) ! Ok, but why would you do it?
    print *, 'In R3, C1 = ', C1
  end subroutine R3
end module M

submodule(M) S1                    ! Submodule of M named S1
  integer, save :: MyVar = 0
! integer, parameter :: RK = kind(1.0) ! Illegal redeclaration of RK
  interface
    submodule subroutine Invisible ! Not accessible by use association
    end subroutine invisible
  end interface
contains
  submodule subroutine R1 ! ( A1, A2 )  ! R1 is accessible by use association
    myVar = a2
    print *, 'In R1, A1 =', a1
  end subroutine R1
end submodule S1

submodule(S1) S2                   ! Submodule of S1 named S2
contains
  submodule subroutine R2 ! ( B1 ) ! R2 is accessible by use association
    print *, 'In R2, B1 =', b1, ', MyVar =', myVar
  end subroutine R2
  submodule subroutine Invisible
    print *, 'Got to Invisible subroutine, RK =', rk
  end subroutine Invisible         ! Not accessible by use association
end submodule S2
```

6 Preparing the way

A TR is a terrible place to introduce an incompatibility. It's better to do it in the context of a revision of the standard.

In the edits specified here, page and line numbers refer to 01-007r3.

In Fortran 95, interface bodies did not access their host scoping units by host association. This standard specifies that interface bodies that do not specify external procedures access their host scoping units by host association. The only opportunities for incompatibility occur if the types of entities in the interface of a dummy procedure are determined implicitly.	3:27+ New ¶
[Editor: Insert "for an external procedure" after "*interface-body*".]	242:46
[Editor: "in ... association" ⇒ "by host association within an interface body for an external procedure".]	243:28
In Fortran 95, it was not possible for a module procedure to have dummy procedures with assumed-shape arguments of an opaque type. For example, the following would have been impossible:	244:16-17
[Editor: Delete.]	244:29
[Editor: "without ... statement" ⇒ "in Fortran 95".]	244:38-39
[Editor: Insert ", an interface body for other than an external procedure" after "module subprogram".	373:40
[Editor: Insert "for an external procedure" after "body".]	373:41
[Editor: "by an IMPORT statement" ⇒ "within an interface body".]	374:35-36
[Editor: Insert "for an external procedure" after "body".]	374:45