Subject: Edits to implement TR 19767
From: Van Snyder

1 Introduction

The following editorial changes, if implemented, would provide the facilities described in TR 19767. Descriptions of how and where to place the new material are enclosed between square brackets within the body of the text. Page and line numbers in the margin refer to 04-007. If there is any conflict between the instructions in the body of the text and the page and line numbers in the margin, the instructions in the body take precedence.

2 Edits

Edits refer to 04-007. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by associated text, while a page and line number followed by + (-) indicates that associated text is to be inserted after (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

[After the third right-hand-side of syntax rule R202 insert:] 9:12+

or *submodule*

[After syntax rule R1104 add the following syntax rule. This is a quotation of the "real" syntax rule in subclause 11.2.2.] 9:34+

R1115a *submodule* **is** *submodule-stmt*
[*specification-part*]
[*module-subprogram-part*]
end-submodule-stmt

[Add another alternative to R1108:] 10:32+

or *separate-module-subprogram*

[In the second line of the first paragraph of subclause 2.2 insert ", a submodule" after "module".] 11:41

[In the fourth line of the first paragraph of subclause 2.2 insert a new sentence:] 11:43

A submodule is an extension of a module; it may contain the definitions of procedures declared in a module or another submodule.

[In the sixth line of the first paragraph of subclause 2.2 insert ", a submodule" after "module".] 11:45

[In the penultimate line of the first paragraph of subclause 2.2 insert "or submodule" after "module".] 11:47

[In the second sentence of 2.2.3.2, insert "or submodule" between "module" and "containing".] 12:28

[Insert a new subclause:] 13:17+

2.2.5 Submodule

A **submodule** is a program unit that extends a module or another submodule. It may provide definitions (12.5) for procedures whose interfaces are declared (12.3.2.1) in an ancestor module or submodule. It may also contain declarations and definitions of other entities, which are accessible in descendant submodules. An entity declared in a submodule is not accessible by use association unless it is a module procedure whose interface is declared in the ancestor module. Submodules are further described in Section 11.

NOTE 2.2$\frac{1}{2}$

The scoping unit of a submodule accesses the scoping unit of its parent module or submodule by host association.

[In the second line of the first row of Table 2.1 insert ", SUBMODULE" after "MODULE".] 14

[Change the heading of the third column of Table 2.2 from "Module" to "Module or Submodule".] 14

[In the second footnote to Table 2.2 insert "or submodule" after "module" and change "the module" to "it".] 14

[In the first line of 2.3.3, insert ", *end-sep-subprogram-stmt*" after "*end-subroutine-stmt*", and insert ", *end-submodule-stmt*," after "*end-module-stmt*". In the third line of subclause 2.3.3, replace "and *end-subroutine-stmt*" by "*end-subroutine-stmt*, and *end-sep-subprogram-stmt*". In the fifth line of subclause 2.3.3, replace "or *end-subroutine-stmt*" by ", *end-subroutine-stmt*, or *end-sep-subprogram-stmt*".] 14:2,4,6

[In the last line of 2.3.3 insert ", *end-submodule-stmt*," after "*end-module-stmt*".] 15:2

[In the first line of the second paragraph of 2.4.3.1.1 insert ", submodule" after "module".] 17:4

[At the end of 3.3.1, immediately before 3.3.1.1, add "END PROCEDURE" and "END SUBMODULE" into the list of adjacent keywords where blanks are optional, in alphabetical order.] 28

[In the second line of the third paragraph of 4.5.1.1 after "definition" insert ", and within its descendant submodules".] 46:10

[In the last line of Note 4.18, after "defined" add ", and within its descendant submodules".] 46

[In the last line of the fourth paragraph of 4.5.3.6, after "definition", add ", and within its descendant submodules".] 55:10

[In the last line of Note 4.40, after "module" add ", and within its descendant submodules".] 55

[In the last line of Note 4.41, after "definition" add ", and within its descendant submodules".] 56

[In the last line of the paragraph before Note 4.44, after "definition" insert ", and within its descendant submodules".] 58:8

[In the third line of the second paragraph of 4.5.5.2 insert ", or submodule" after "module".] 59:23

[In the fourth line of the second paragraph of 4.5.5.2 insert ", or accessing the submodule" after "module".] 59:24

[In the second paragraph of Note 4.48, insert "or submodule" after the first "module" and insert "or accessing the submodule" after the second "module"]. 60

[In the first line of the second paragraph of 5.1.2.12 after "attribute" insert ", or within any of its descendant submodules".] 84:3

[In the first and third lines of the second paragraph of 5.1.2.13 insert "or submodule" after "module" twice.] 84:14,16

[In the third line of the penultimate paragraph of 6.3.1.1 replace "or a subobject thereof" by "or submodule, or a subobject thereof,".] 113:18

[In the first two lines of the first paragraph after Note 6.23 insert "or submodule" after "module" twice.] 115:9-10

[In the second line of the first paragraph of Section 11 insert ", a submodule" after "module".] 249:3

[In the first line of the second paragraph of Section 11 insert ", submodules" after "modules".] 249:4

[Add another alternative to R1108] 250:17+

or *separate-module-subprogram*

[Within the first paragraph of 11.2.1, at its end, insert the following sentence:] 251:8

A submodule shall not reference its ancestor module by use association, either directly or indirectly.

[Then insert the following note:]

NOTE 11.6$\frac{1}{3}$

It is possible for submodules with different ancestor modules to access each others' ancestor modules by use association.

[After constraint C1110 insert an additional constraint:] 251:34+

C1110a (R1109) If the USE statement appears within a submodule, *module-name* shall not be the name of the ancestor module of that submodule (11.2.2).

[Insert a new subclause immediately before 11.3:] 253:2-

11.2.2 Submodules

A **submodule** is a program unit that extends a module or another submodule. The program unit that it extends is its **parent**; its parent is specified by the *parent-identifier* in the *submodule-stmt*. A submodule is a **child** of its parent. An **ancestor** of a submodule is its parent or an ancestor of its parent. A **descendant** of a module or submodule is one of its children or a descendant of one of its children. The **submodule identifier** consists of the ancestor module name together with the submodule name.

NOTE 11.6$\frac{2}{3}$

A module and its submodules stand in a tree-like relationship one to another, with the module at the root. Therefore, a submodule has exactly one ancestor module and may optionally have one or more ancestor submodules.

A submodule accesses the scoping unit of its parent by host association.

A submodule may provide implementations for module procedures, each of which is declared by a module procedure interface body (12.3.2.1) within that submodule or one of its ancestors, and declarations and definitions of other entities that are accessible by host association in descendant submodules.

R1115a *submodule* **is** *submodule-stmt*
[*specification-part*]
[*module-subprogram-part*]
end-submodule-stmt

R1115b *submodule-stmt* **is** SUBMODULE (*parent-identifier*) *submodule-name*

R1115c *parent-identifier* **is** *ancestor-module-name* [: *parent-submodule-name*]

R1115d *end-submodule-stmt* **is** END [SUBMODULE [*submodule-name*]]

C1114a (R1115a) An automatic object shall not appear in the *specification-part* of a submodule.

C1114b (R1115a) A submodule *specification-part* shall not contain a *format-stmt* or a *stmt-function-stmt*.

C1114c (R1115a) If an object of a type for which *component-initialization* is specified (R444) is declared in the *specification-part* of a submodule and does not have the ALLOCATABLE or POINTER attribute, the object shall have the SAVE attribute.

C1114d (R1115c) The *ancestor-module-name* shall be the name of a nonintrinsic module; the *parent-submodule-name* shall be the name of a descendant of that module.

C1114e (R1115d) If a *submodule-name* is specified in the *end-submodule-stmt*, it shall be identical to the *submodule-name* specified in the *submodule-stmt*.

[In the last line of the first paragraph of 12.3 after "units" add ", except that for a separate module procedure body (12.5.2.4), the dummy argument names, binding label, and whether it is recursive shall be the same as in its corresponding module procedure interface body (12.3.2.1)".] 257:13

[In C1210 insert "that is not a module procedure interface body" after "*interface-body*".] 259:20

[After the third paragraph after constraint C1211 insert the following paragraphs and constraints.] 259:30+

A **module procedure interface body** is an interface body in which the *prefix* of the initial *function-stmt* or *subroutine-stmt* includes MODULE. It declares the **module procedure interface** for a separate

module procedure (12.5.2.4). A separate module procedure is accessible by use association if and only if its interface body is declared in the specification part of a module and its name has the PUBLIC attribute. If a corresponding (12.5.2.4) separate module procedure is not defined, the interface may be used to specify an explicit specific interface but the procedure shall not be used in any way.

C1211a (R1205) A scoping unit in which a module procedure interface body is declared shall be a module or submodule.

C1212b (R1205) A module procedure interface body shall not appear in an abstract interface block.

[Add another alternative to R1228:] 280:3+

or MODULE

[Add constraints after C1242:] 280:7+

C1242a (R1227) MODULE shall appear only within the *function-stmt* or *subroutine-stmt* of a module subprogram or of an interface body that is declared in the scoping unit of a module or submodule.

C1242b (R1227) If MODULE appears within the *prefix* in a module subprogram, a module procedure interface having the same name as the subprogram shall be declared in the module or submodule in which the subprogram is defined, or shall be declared in an ancestor of that program unit and be accessible by host association from that ancestor.

C1242c (R1227) If MODULE appears within the *prefix* in a module subprogram, the subprogram shall specify the same characteristics and dummy argument names as its corresponding (12.5.2.4) module procedure interface body.

C1242d (R1227) If MODULE appears within the *prefix* in a module subprogram and a binding label is specified, it shall be the same as the binding label specified in the corresponding module procedure interface body.

C1242e (R1227) If MODULE appears within the *prefix* in a module subprogram, RECURSIVE shall appear if and only if RECURSIVE appears in the *prefix* in the corresponding module procedure interface body.

[Insert the following new subclause before the existing subclause 12.5.2.4 and renumber succeeding subclauses appropriately:] 283:1-

12.5.2.4 Separate module procedures

A **separate module procedure** is a module procedure defined by a *separate-module-subprogram*, by a *function-subprogram* in which the *prefix* of the initial *function-stmt* includes MODULE, or by a *subroutine-subprogram* in which the prefix of the initial *subroutine-stmt* includes MODULE. Its interface is declared by a module procedure interface body (12.3.2.1) in the *specification-part* of the module or submodule in which the procedure is defined, or in an ancestor module or submodule.

R1234a *separate-module-subprogram* **is** MODULE PROCEDURE *procedure-name*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-sep-subprogram-stmt

R1234b *end-sep-subprogram-stmt* **is** END [PROCEDURE [*procedure-name*]]

C1251a (R1234a) The *procedure-name* shall be the same as the name of a module procedure interface that is declared in the module or submodule in which the *separate-module-subprogram* is defined, or is declared in an ancestor of that program unit and is accessible by host association from that ancestor.

C1251b (R1234b) If a *procedure-name* appears in the *end-sep-subprogram-stmt*, it shall be identical to the *procedure-name* in the MODULE PROCEDURE statement.

A module procedure interface body and a subprogram that defines a separate module procedure **correspond** if they have the same name, and the module procedure interface is declared in the same program unit as the subprogram or is declared in an ancestor of the program unit in which the procedure is defined and is accessible by host association from that ancestor. A module procedure interface body

shall not correspond to more than one subprogram that defines a separate module procedure.

NOTE 12.40$\frac{1}{2}$

A separate module procedure can be accessed by use association if and only if its interface body is declared in the specification part of a module and its name has the PUBLIC attribute. A separate module procedure that is not accessible by use association might still be accessible by way of a procedure pointer, a dummy procedure, a type-bound procedure, a binding label, or means other than Fortran.

If a procedure is defined by a *separate-module-subprogram*, its characteristics are specified by the corresponding module procedure interface body.

If a separate module procedure is a function defined by a *separate-module-subprogram*, the result variable name is determined by the FUNCTION statement in the module procedure interface body. Otherwise, the result variable name is determined by the FUNCTION statement in the module subprogram.

[In constraint C1253 replace "*module-subprogram*" by "a *module-subprogram* that does not define a separate module procedure".] 283:7

[In the first line of the first paragraph after syntax rule R1237 in 12.5.2.6 insert ", submodule" after "module",] 284:37

[After the second paragraph of subclause 15.4.1 insert the following constraint]: 403:38+

C1506 A procedure defined in a submodule shall not have a binding label unless its interface is declared in the ancestor module.

[In the list in subclause 16.0, add an item after item (1):] 405:9+

(1$\frac{1}{2}$) A submodule identifier (11.2.2),

[In the second sentence of the first paragraph of 16.1, insert "non-submodule" before the first "program unit".] 405:19

[After the second sentence of the first paragraph of 16.1, insert a new sentence "A submodule identifier of a submodule is a global identifier and shall not be the same as the submodule identifier of any other submodule."] 405:22

[After Note 16.2 add:] 406:1-

NOTE 16.2$\frac{1}{2}$

Submodule identifiers are global identifiers, but since they consist of a module name and a descendant submodule name, the name of a submodule can be the same as the name of another submodule so long as they do not have the same ancestor module.

[In item (1) in the first numbered list in 16.2, after "abstract interfaces" insert ", module procedure interfaces".] 406:6

[In the paragraph immediately before Note 16.3, after "(4.5.9)" insert ", and a separate module procedure shall have the same name as its corresponding module procedure interface body".] 406:20

[In the first line of the first paragraph of 16.4.1.3 insert ", a module procedure interface body" after "module subprogram". In the second line, insert "that is not a module procedure interface body" after "interface body".] 411:2,3

[In the third line of the first paragraph of 16.4.1.3, after"interface body.", insert a new sentence: "A submodule has access to the named entities of its parent by host association."] 411:4

[In the fifth line of the first paragraph of subclause 16.4.1.3, insert ', module procedure interfaces' after 'abstract interfaces'.] 411:6

[In the third line after the sixteen-item list in 16.4.1.3 insert "that does not define a separate module procedure" after the first "subprogram".] 411:34

[In the first line of Note 16.9, after "interface body" insert "that is not a module procedure interface body".] 412:1+2

[Insert a new item after item (5)(d) in the list in 16.4.2.1.3:] 415:15+

(d$\frac{1}{2}$) Is in the scoping unit of a submodule if any scoping unit in that submodule or any of its descendant submodules is in execution.

[In item (3)(c) of 16.5.6 insert "or submodule" after the first instance of "module" and insert "or accessing the submodule" after the second instance of "module".] 422:14-15

[In item (3)(d) of 16.5.6 insert "or submodule" after the first instance of "module" and insert "or accessing the submodule" after the second instance of "module".] 422:16-17

[Insert the following definitions into the glossary in alphabetical order:]

ancestor (11.2.2) : Of a submodule, its parent or an ancestor of its parent. 425:15+

child (11.2.2) : A submodule is a child of its parent. 426:43+

correspond (12.5.2.4) : A module procedure interface body and a subprogram that defines a separate module procedure correspond if they have the same name, and the module procedure interface is declared in the same program unit as the subprogram or is declared in an ancestor of the program unit in which the procedure is defined and is accessible by host association from that ancestor. 427:31+

descendant (11.2.2) : Of a module or submodule, one of its children or a descendant of one of its children. 428:31+

module procedure interface (12.3.2.1) : An interface defined by an interface body in which MODULE appears in the *prefix* of the initial *function-stmt* or *subroutine-stmt*. It declares the interface for a separate module procedure. 432:11+

parent (11.2.2) : Of a submodule, the module or submodule specified by the *parent-identifier* in its *submodule-stmt*. 433:3+

separate module procedure (12.5.2.4): A module procedure defined by a *separate-module-subprogram* or a *function-subprogram* or *subroutine-subprogram* in which MODULE appears in the *prefix* of the initial *function-stmt* or *subroutine-stmt*. 434:30+

submodule (2.2.5, 11.2.2) : A program unit that depends on a module or another submodule; it extends the program unit on which it depends. 435:20+

submodule identifier (11.2.2) : Identifier of a submodule, consisting of the ancestor module name together with the submodule name.

[Insert a new subclause immediately before C.9:] 477:29+

C.8.3.9 Modules with submodules

Each submodule specifies that it is the child of exactly one parent module or submodule. Therefore, a module and all of its descendant submodules stand in a tree-like relationship one to another.

If a module procedure interface body that is specified in a module has public accessibility, and its corresponding separate module procedure is defined in a descendant of that module, the procedure can be accessed by use association. No other entity in a submodule can be accessed by use association. Each program unit that accesses a module by use association depends on it, and each submodule depends on its ancestor module. Therefore, if one changes a separate module procedure body in a submodule but does not change its corresponding module procedure interface, a tool for automatic program translation would not need to reprocess program units that access the module by use association. This is so even if the tool exploits the relative modification times of files as opposed to comparing the result of translating the module to the result of a previous translation.

By constructing taller trees, one can put entities at intermediate levels that are shared by submodules at lower levels; changing these entities cannot change the interpretation of anything that is accessible

from the module by use association. Developers of modules that embody large complicated concepts can exploit this possibility to organize components of the concept into submodules, while preserving the privacy of entities that are shared by the submodules and that ought not to be exposed to users of the module. Putting these shared entities at an intermediate level also prevents cascades of reprocessing and testing if some of them are changed.

The following example illustrates a module, `color_points`, with a submodule, `color_points_a`, that in turn has a submodule, `color_points_b`. Public entities declared within `color_points` can be accessed by use association. The submodules `color_points_a` and `color_points_b` can be changed without causing retranslation of program units that access the module `color_points`.

The module `color_points` does not have a *contains-part*, but a *contains-part* is not prohibited. The module could be published as definitive specification of the interface, without revealing trade secrets contained within `color_points_a` or `color_points_b`. Of course, a similar module without the `module` prefix in the interface bodies would serve equally well as documentation – but the procedures would be external procedures. It would make little difference to the consumer, but the developer would forfeit all of the advantages of modules.

```
module color_points

  type color_point
    private
    real :: x, y
    integer :: color
  end type color_point

  interface              ! Interfaces for procedures with separate
                         ! bodies in the submodule color_points_a
    module subroutine color_point_del ( p ) ! Destroy a color_point object
      type(color_point), allocatable :: p
    end subroutine color_point_del
    ! Distance between two color_point objects
    real module function color_point_dist ( a, b )
      type(color_point), intent(in) :: a, b
    end function color_point_dist
    module subroutine color_point_draw ( p ) ! Draw a color_point object
      type(color_point), intent(in) :: p
    end subroutine color_point_draw
    module subroutine color_point_new ( p ) ! Create a color_point object
      type(color_point), allocatable :: p
    end subroutine color_point_new
  end interface

end module color_points
```

The only entities within `color_points_a` that can be accessed by use association are separate module procedures for which corresponding module procedure interface bodies are provided in `color_points`. If the procedures are changed but their interfaces are not, the interface from program units that access them by use association is unchanged. If the module and submodule are in separate files, utilities that examine the time of modification of a file would notice that changes in the module could affect the translation of its submodules or of program units that access the module by use association, but that changes in submodules could not affect the translation of the parent module or program units that access it by use association.

The variable `instance_count` is not accessible by use association of `color_points`, but is accessible within `color_points_a`, and its submodules.

```
  submodule ( color_points ) color_points_a ! Submodule of color_points

    integer, save :: instance_count = 0

    interface                   ! Interface for a procedure with a separate
                                ! body in submodule color_points_b
      module subroutine inquire_palette ( pt, pal )
        use palette_stuff       ! palette_stuff, especially submodules
                                ! thereof, can access color_points by use
                                ! association without causing a circular
                                ! dependence during translation because this
                                ! use is not in the module.  Furthermore,
                                ! changes in the module palette_stuff do not
                                ! affect the translation of color_points.
        type(color_point), intent(in) :: pt
        type(palette), intent(out) :: pal
      end subroutine inquire_palette

    end interface

  contains ! Invisible bodies for public module procedure interfaces
           ! declared in the module

    module subroutine color_point_del ( p )
      type(color_point), allocatable :: p
      instance_count = instance_count - 1
      deallocate ( p )
    end subroutine color_point_del
    real module function color_point_dist ( a, b ) result ( dist )
      type(color_point), intent(in) :: a, b
      dist = sqrt( (b%x - a%x)**2 + (b%y - a%y)**2 )
    end function color_point_dist
    module subroutine color_point_new ( p )
      type(color_point), allocatable :: p
      instance_count = instance_count + 1
      allocate ( p )
    end subroutine color_point_new

  end submodule color_points_a
```

The subroutine `inquire_palette` is accessible within `color_points_a` because its interface is declared therein. It is not, however, accessible by use association, because its interface is not declared in the module, `color_points`. Since the interface is not declared in the module, changes in the interface cannot affect the translation of program units that access the module by use association.

```
submodule ( color_points:color_points_a ) color_points_b ! Subsidiary**2 submodule

contains
  ! Invisible body for interface declared in the ancestor module
  module subroutine color_point_draw ( p )
    use palette_stuff, only: palette
    type(color_point), intent(in) :: p
    type(palette) :: MyPalette
    ...; call inquire_palette ( p, MyPalette ); ...
  end subroutine color_point_draw

  ! Invisible body for interface declared in the parent submodule
  module procedure inquire_palette
    ... implementation of inquire_palette
  end procedure inquire_palette

  subroutine private_stuff ! not accessible from color_points_a
    ...
  end subroutine private_stuff

end submodule color_points_b

module palette_stuff
  type :: palette ; ... ; end type palette
contains
  subroutine test_palette ( p )
  ! Draw a color wheel using procedures from the color_points module
    type(palette), intent(in) :: p
    use color_points ! This does not cause a circular dependency because
                     ! the "use palette_stuff" that is logically within
                     ! color_points is in the color_points_a submodule.
    ...
  end subroutine test_palette
end module palette_stuff
```

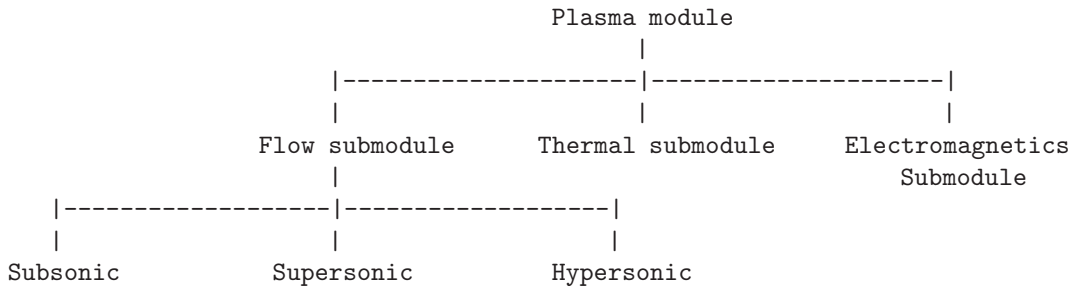
There is a `use palette_stuff` in `color_points_a`, and a `use color_points` in `palette_stuff`. The `use palette_stuff` would cause a circular reference if it appeared in `color_points`. In this case, it does not cause a circular dependence because it is in a submodule. Submodules are not accessible by use association, and therefore what would be a circular appearance of `use palette_stuff` is not accessed.

```
program main
  use color_points
  ! "instance_count" and "inquire_palette" are not accessible here
  ! because they are not declared in the "color_points" module.
  ! "color_points_a" and "color_points_b" cannot be accessed by
  ! use association.
  interface draw                    ! just to demonstrate it's possible
    module procedure color_point_draw
  end interface
  type(color_point) :: C_1, C_2
  real :: RC
  ...
  call color_point_new (c_1)        ! body in color_points_a, interface in color_points
  ...
  call draw (c_1)                   ! body in color_points_b, specific interface
```

```
                                    ! in color_points, generic interface here.
    ...
    rc = color_point_dist (c_1, c_2)  ! body in color_points_a, interface in color_points
    ...
    call color_point_del (c_1)        ! body in color_points_a, interface in color_points
    ...
  end program main
```

A multilevel submodule system can be used to package and organize a large and interconnected concept without exposing entities of one subsystem to other subsystems.

Consider a `Plasma` module from a Tokomak simulator. A plasma simulation requires attention at least to fluid flow, thermodynamics, and electromagnetism. Fluid flow simulation requires simulation of subsonic, supersonic, and hypersonic flow. This problem decomposition can be reflected in the submodule structure of the `Plasma` module:



Entities can be shared among the `Subsonic, Supersonic`, and `Hypersonic` submodules by putting them within the `Flow` submodule. One then need not worry about accidental use of these entities by use association or by the `Thermal` or `Electromagnetics` modules, or the development of a dependency of correct operation of those subsystems upon the representation of entities of the `Flow` subsystem as a consequence of maintenance. Since these these entities are not accessible by use association, if any of them are changed, the new values cannot be accessed in program units that access the `Plasma` module by use association; the answer to the question "where are these entities used" is therefore confined to the set of descendant submodules of the `Flow` submodule.