ISO/IEC JTC1/SC22/WG5 N1542

WORKING DRAFT

ISO IEC TECHNICAL REPORT 19767

ISO/IEC JTC1 WG5 PROJECT 1.22.02.01.01.01

Enhanced Module Facilities

in

Fortran

An extension to IS 1539-1

April 2003

THIS PAGE TO BE REPLACED BY ISO-CS

Contents

Foreword

[General part to be provided by ISO CS]

This technical report specifies an extension to the module program unit facilities of the programming language Fortran. Fortran is specified by the international standard ISO/IEC 1539-1. This document has been prepared by ISO/IEC JTC1/SC22/WG5, the technical working group for the Fortran language. It is the intention of ISO/IEC JTC1/SC22/WG5 that the semantics and syntax specified by this technical report be included in the next revision of the Fortran standard (ISO/IEC 1539-1) without change unless experience in the implementation and use of this feature identifies errors that need to be corrected, or changes are needed to achieve proper integration, in which case every reasonable effort will be made to minimize the impact of such changes on existing implementations.

This extension is upward compatible with the version of Fortran specified by the international standard ISO/IEC 1539-1 : 1996, informally known as Fortran 95. It would be possible for the facilities defined in this technical report to be implemented as an extension to this version of Fortran.

A revision of the 1996 standard is in an advanced stage of processing and is expected to be published in 2004. This version whose technical content was finalized in 2003 will be known as Fortran 2003. The extension defined by this TR is also upward compatible with this 2003 revision of Fortran.

Introduction

The module system of Fortran, as standardized by ISO/IEC 1539-1 : 1996 and unchanged in ISO/IEC 1539-1 : 2004, while adequate for programs of modest size, has shortcomings that become evident when used for large programs or programs requiring large modules. The primary cause of these shortcomings is that modules are monolithic. The declaration of usable data, datatypes, interfaces to procedures manipulating such data, and the implementation details defining such procedures must be included within the same module.

A change to the procedure implementation definition within a module without any change to the interface will usually trigger unnecessary recompilation of any dependent modules or programs. For a module that has a large number of dependent programs, this can be a very significant overhead.

The lack of separation between interface declaration and procedure implementation makes it difficult to manage the program development process, particularly if multiple programmers are needed to work on one large module. Because of the monolithic nature of the module facility each programmer needs to work on the same source code with the obvious danger of incompatible code changes.

This technical report extends the module facility of Fortran by introducing a new program unit called a submodule. A submodule is separate from but dependent on a module. Program developers may include the implementation details of module procedures in submodules while declaring their interfaces in the module on which the submodules depend.

Since a submodule may not be accessed directly by a USE statement but only indirectly via its parent module, a change to an implementation detail for a module procedure defined separately in a submodule does not change the interface on which program units that use the module depend. Such changes therefore do not trigger chains of dependent recompilations.

Separate submodules may be implemented by different programmers but they must implement separate module procedures that are consistent with the original interface designs expressed in the parent module. Thus the management of large development processes is significantly facilitated.

Provided a module declares only the publicly visible aspects of the interface to a facility and all the implementation details are provided in submodules, the source code of the module may be published as definitive documentation of the interface without exposing possibly propriety implementation techniques.

Information technology – Programming Languages – Fortran

Technical Report: Enhanced Module Facilities

1 General

1.1 Scope

This technical report specifies an extension to the module facilities of the programming language Fortran. The current Fortran language is specified by the international standard ISO/IEC 1539-1 : 1996, informally known as Fortran 95. This standard is currently under revision and a new version is scheduled for publication in 2004. This technical report is also a compatible extension to this new language version, informally known as Fortran 2003.

The extension specified in this technical report allows program authors to develop the implementation details of concepts in new program units, called submodules that cannot be accessed directly by use association but are dependent on a "parent" module that declares the interfaces.

Clause 2 of this technical report contains a general and informal but precise description of the extended functionalities. Clause 3 contains detailed editorial changes that would implement the revised language specification if they were applied to the international standard ISO/IEC 1539-1 : 1996. Clause 4 contains detailed editorial changes that would implement the revised language specification if they were applied to the proposed international standard expected to be designated ISO/IEC 1539-1 : 2004.

1.2 Normative References

The following standards contain provisions that, through reference in this text, constitute provisions of this technical report. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. Parties to agreements based on this technical report are, however, encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referenced applies. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 1539-1 : 1996 Information technology - Programming Languages - Fortran

ISO/IEC 1539-1 : 2004 Information technology - Programming Languages - Fortran

2 Requirements

The following subclauses contain a general description of the extensions to the syntax and semantics of the current Fortran programming language to provide facilities for submodules, and to allow the separation of module procedure definitions into interface and implementation parts.

2.1 Summary

This technical report defines a new program unit, "a submodule", a new form of name association, "submodule association", an additional variety of interface body, "a forward interface body", and a new variety of procedure, "a separate module procedure". A submodule is a program unit that is dependent on and subsidiary to a module or another submodule. By putting a forward interface body in a module and its corresponding separate module procedure in a submodule, compilation of program units that access the interface body by use association does not depend on the details of the corresponding procedure body implementation. The procedure body implementation cannot change the interface body on which it depends.

2.2 Submodules

A module or submodule may have several subsidiary submodules. If it has subsidiary submodules, it is the parent of those subsidiary submodules, and each of those submodules is a child of its parent. A submodule accesses its parent by submodule association. An ancestor of a submodule is that submodule, or an ancestor of its parent. A descendant of a module or submodule is that program unit, or a descendant of a child of that program unit.
A submodule is introduced by a statement of the form,

```
SUBMODULE ( parent-name ) submodule-name
```

and terminated by a statement of the form,

```
END SUBMODULE submodule-name
```

The *parent-name* is the name of the parent module or submodule.
It is not possible to access entities declared in the specification part of a submodule by use association because a `USE` statement is required to specify a module, not a submodule. `PRIVATE` and `PUBLIC` attributes may not be specified for entities declared in a submodule. All entities declared in a submodule are accessible in its descendants, within procedures contained in the submodule by host association and within any child submodule by submodule association.

2.3 Submodule association

The SUBMODULE statement identifies the parent of the submodule. Within the scoping unit of the submodule all named entities visible in the parent are accessed by **submodule association**. The accessed entities have the attributes, characteristics, definition status, and values, if any, specified in the parent. The entities made accessible in the submodule are identified by the names or generic identifiers used to identify them in the parent.
Within a submodule any entity accessed by submodule association may be redeclared in part or in full. However, such redeclaration shall confirm the attributes and characteristics of the submodule associated entity and all references to submodule associated entities, whether redeclared or not, are references to the parent entity.
If a data entity is initialized in any descendant submodule this must also confirm the definition status of the associated data entity and shall initialize it to the same value.

{ To provide checkable interface details at the point of implementation development, it is crucial that it be possible to redeclare forward interface characteristics. Since such forward interface characteristics may well depend on data-objects with initialized values accessible within an ancestor, it is desirable that redeclaration of all submodule associated entities be permitted but that it is a requirement that redclaration does not alter the status or value of the inherited entity. }

2.4 Forward interface body

A **forward interface body** is declared with the `FORWARD` prefix on the `FUNCTION` or `SUBROUTINE` statement that introduces the interface body specification in an interface block. Such an interface body specifies the characteristics and dummy argument names of the procedure, and that its corresponding procedure body shall be defined by a separate module procedure in a descendant of the module or submodule in which it appears. A forward interface body accesses the module or submodule in which it is declared by host association.

2.5 Separate module procedure

A **separate module procedure** is declared with a `SEPARATE` prefix on the `FUNCTION` or `SUBROUTINE` statement that introduces its implementation definition. The name of the procedure

so declared shall be that of a function or subroutine whose interface was declared with a FORWARD prefix in an ancestor module or submodule.

{Strictly speaking the SEPARATE prefix is not necessary. The name of the procedure is sufficient to identify an already accessible forward interface body and hence to indicate that the particular procedure implementation definition is that of a separate module procedure. However, the SEPARATE prefix may be considered to provide additional documentary alert that the operative interface is declared elsewhere.
A possible restricted form of redeclaration for separate module procedures would be to use the syntax
SEPARATE,PROCEDURE(*forward-interface-name*)
As the header if no redeclaration was to be done and to require full redeclaration of the interface if any was wanted. I have not written the remainder of the TR to do this as I do not think it necessary or particularly desirable but mention it here as a possibility.}

The characteristics and dummy argument names are those declared by the referenced interface body, but they may optionally be confirmed by redeclaration in the separate procedure body. A separate procedure may be redeclared PURE even if its interface body does not specify that it is PURE.
If the procedure is a function, the result variable name is determined by the declaration of the separate module procedure, not by the forward interface body. If the forward interface body declares a result variable name different from the function name, that declaration is ignored, except for its use in specifying the result variable characteristics.
A separate procedure is accessible if and only if its interface body is accessible.
At most one separate module procedure shall define an implementation for each forward interface in the tree of related modules and submodules that are linked to form an executable program.

{This avoids the problem of having to specify a search order if a forward interface has more than one submodule implementing its procedure body. Of course it would be possible to allow redefinition down any leg of the tree but to define that say the first separate procedure body encounted was used and later definitions ignored. This however does not deal with two or more definitions at the same level, where any search order would have to be specified by means not yet defined within the language.}

2.6 Examples of modules with submodules

The example module POINTS below declares a type POINT and a forward interface body for a separate module function POINT_DIST. Because the interface body includes the FORWARD prefix, the interface body accesses the scoping unit of the module by host association.

```
MODULE POINTS

  TYPE :: POINT
    REAL :: X, Y
  END TYPE POINT

  INTERFACE
    FORWARD FUNCTION POINT_DIST( A, B )
      TYPE(POINT), INTENT(IN) :: A, B ! type accessed by host association
      REAL :: POINT_DIST
    END FUNCTION POINT_DIST
  END INTERFACE

END MODULE POINTS
```

The example submodule POINTS_A below is a submodule of the POINTS module. The scope of the type name POINT and the name and characteristics of the function POINT_DIST extend into the submodule by submodule association. The characteristics of the function POINT_DIST can be fully confirmed by redeclaration in the separate module function body, or taken from the forward interface body in the POINTS module.

```
SUBMODULE ( POINTS ) POINTS_A
CONTAINS

  SEPARATE FUNCTION POINT_DIST( A, B ) ! complete redeclaration
    TYPE(POINT), INTENT(IN) :: A, B    ! of the interface confirming
    REAL :: POINT_DIST                 ! the characteristics from the parent
    POINT_DIST = SQRT( (A%X-B%X)**2 + (A%Y-B%Y)**2 )
  END FUNCTION POINT_DIST

END SUBMODULE POINTS_A
```

An alternative example of a coding of the submodule POINTS_A showing minimal redeclaration of the separate module function is

```
SUBMODULE ( POINTS ) POINTS_A
CONTAINS

  SEPARATE FUNCTION POINT_DIST() ! mimimal redeclaration
                                 ! of the interface
    POINT_DIST = SQRT( (A%X-B%X)**2 + (A%Y-B%Y)**2 )
  END FUNCTION POINT_DIST

END SUBMODULE POINTS_A
```

The minimum required to introduce the definition of a separate module procedure is the specification of whether it is a function or subroutine and the name of the associated forward interface. However, any combination of some or all of the additional characteristics of the procedure interface may be redeclated as part of the definition as long as they merely confirm the characteristics of the associated forward interface.
Program units may access the facilities defined in this module and submodule set only by way of a USE statement that names the module; program units are prohibited from accessing submodules by use association since a submodule name may not appear in a USE statement. As a result a change made to the code in the submodule cannot affect the attributes or characteristics of the entities accessed from the module by use association. Therefore, a submodule change need not trigger a recompilation of any of the use associated program units.

3 Required editorial changes to, ISO/IEC 1539-1 : 1996

The following editorial changes to ISO/IEC 1539-1 : 1996, if implemented, would provide the facilities described in foregoing section of this report as an extension to Fortran 95. The specific changes are grouped by the sections of the above document to which they apply.

Section 2 changes

In 2.1
After rule R1104 add rule

R1111a *submodule* **is** *submodule-stmt*
[*specification-part*]
[*module-procedure-part*]
end-submodule-stmt

In 2.2

First paragraph, second sentence, after “module,” add “ submodule,”
Before sentence beginning “A block data program” add sentence
A submodule contains implementation definitions for separate module procedures whose interfaces are declared with a `FORWARD` prefix in an ancestor program unit, and definitions that are to be made accessible to descendant submodules.
In the pre-penultimate sentence, after “module,” add “ submodule,”
In the ultimate sentence, before “but” add “ or submodule “
In 2.2.3.2
Replace the second sentence by
A module procedure may be invoked from within any scoping unit that accesses its declaration (12.3.2.1) or definition (12.5).
Insert the following note at the end of 2.2.3.2.
NOTE 2.2a
The scoping unit of a submodule accesses the scoping unit of its parent module or submodule by submodule association (11.3.4).
Insert a new subclause:
2.2.5 Submodule
A submodule is a program unit that extends a module or submodule. It contains definitions (12.5) for separate module procedures whose forward interfaces are declared (12.3.2.1) in its parent module or submodule. It may also contain declarations and definitions of entities that are accessible to descendant submodules. An entity declared in a submodule is not accessible by use association, but a procedure that is declared in a module and defined in one of that module’s descendant submodules is accessible by use association.
In 2.3.1
In the second line of the first row of Table 2.1 insert “,SUBMODULE ”after “MODULE ”.
In 2.3.2
Change the heading of the third column of Table 2.2 from “Module ”to
“Module or Submodule ”
In the third footnote to Table 2.2 insert “or submodule ”after “module ”and change “the module ”to “it ”
In the last line of 2.3.3 insert “,*end-submodule-stmt*” after “*end-module-stmt* ”
In 2.5.6
Last sentence, before “statement” add “ or submodule “

Section 3 changes
At the end of 3.3.1,immediately before 3.3.1.1,add “END SUBMODULE ”to the list of adjacent keywords where blanks are optional, in alphabetical order.

Section 4 changes
In 4.4
In the third line of the paragraph following note 4.16 after “itself ” add “and its descendant submodules ”.
In 4.4.1
In the last line of the second paragraph following note 4.19, after “definition ”add “and its descendant submodules ”.]
In the first sentence of the third paragraph following Note 4.19, after “definition ”,add “and its descendant submodules ”
In the last line of the same paragraph, after “defining module ”add “and its descendant submodules ”
In the Note 4.22, in the sentence following the first block of code, after “module ”add “and its descendant submodules ”
In the last line of Note 4.23, after “defined” add “, and its descendant submodules ”

Section 5 changes

In 5.1.2.2

[In the first line of the paragraph following the constraint, replace “outside the module.” by “by use association.”

Add at the end of the paragraph,

All entities, declared in the module, regardless of their accessibility attribute, are accessible in the module’s descendant submodules by submodule association.

In 5.1.2.5

In the first sentence of the second paragraph after, “module” add “ or one of its decsendent submodules”

Section 6 changes

In 6.3.3.1

In item (2) in the first list, after “module” add “ or submodule”

In item (1) of the second list, after “use association” add “ or submodule association”

In 6.3.3.2

In item (4) of the list, after “module” add “ or submodule” twice

Sections 7 – 10 no changes

Section 11 changes

First paragraph, after “a module,” add “ a submodule,”

Add new subsection 11.3.3 and renumber as necessary

11.3.3 Submodules

A **submodule** is a program unit that is subsidiary to and dependent on a module or another submodule, termed its **parent**. This parent is identified by the *parent-name* in the submodule statement that introduces the submodule. A submodule is a **child** of its parent. An **ancestor** of a submodule is itself and any ancestor of its parent. A **descendant** of a module or a submodule is itself and any descendant of its child submodules.

R1111a *submodule* **is** *submodule-stmt*
[*specification-part*]
[*module-procedure-part*]
end-submodule-stmt

R1111b *submodule-stmt* **is** SUBMODULE (*parent-name*) *submodule-name*

R1111c *end-submodule-stmt* **is** END [SUBMODULE [*submodule-name*]]

Constraint: The *parent-name* shall be the name of a submodule or a nonintrinsic module.

Constraint: If a *submodule-name* is specified in the *end-submodule-stmt* it shall be identical to the *submodule-name* specified in the *submodule-stmt*.

Constraint: A submodule *specification-part* shall not contain a *stmt-function-stmt*, an *entry-stmt*, or a *format-stmt*.

Constraint: If an object of a type for which *component-initialization* is specified appears in the specification-part of a submodule and does not have the ALLOCATABLE or POINTER attribute, the object shall have the SAVE attribute.

A submodule name must be different from that of every other module or submodule. A submodule may contain USE statements by which it may access modules other than its ancestor module.

A submodule accesses named entities from its parent by submodule association (11.3.4).

A submodule may provide implementations for separate module procedures that are declared by forward interface bodies in ancestor program units. A submodule may also declare and define other entities that are accessible in descendant submodules. One and only one separate module procedure shall provide a definition for a procedure declared by a forward interface.

11.3.4 Submodule association

The scoping unit of a submodule accesses all named data objects, derived types, interface blocks, procedures, generic identifiers, and namelist groups accessible in its parent by submodule association, regardless of the accessibility attributes.

Any reference to the name of a submodule associated entity within the scoping unit of the submodule is a reference to the entity accessed from the parent program unit.

A submodule associated entity may be redeclared, in part or completely, in the submodule but such redeclaration shall confirm the attributes, characteristics, and values (if any) of the associated parent entity. Any attributes, characteristics or values not confirmed by redeclaration remain as specified in the ancestor program unit.

Section 12 changes

In 12.1.2.2, third paragraph before "subprogram" add "or submodule ", add the sentence "A is a procedure that defines an implementation for a procedure declared by a forward interface body."

In 12.3.2.1,

In rule R1205 before "function-stmt" and "subroutine-stmt" add "[FORWARD] "

Before rule R1206 add

Constraint: The FORWARD prefix shall appear only in an *interface-body* within a module or submodule.

In paragraph following constraints, third sentence, after "external procedure" add ", separate module procedure,"

In the penultimate line of this paragraph, before ";otherwise, add ". If the interface body has the `FORWARD` prefix it declares the interface for a module procedure that shall be defined by a separate module procedure body within a descendant module or submodule"

In 12.5.2.2,

At the end of rule R1219 add line

or SEPARATE

Add further constraints before rule R1220

Constaint: SEPARATE shall appear if and only if the *function-name* is that of a forward interface body declared in an ancestor module or submodule.

Constraint: SEPARATE shall not appear except for a module procedure.

In 12.5.2.3,

Add constraint following rule R1222

Constaint: SEPARATE shall appears if an only if the *subroutine-name* is that of a forward interface body declared in an ancestor module or submodule.

Section 13 no changes

Section 14 changes

In 14.1.2.4

In the list add an item (1)(e)

If that scoping unit is a submodule and that name is made accessible by submodule association from an ancestor where the name is established to generic.

Add an item (2)(g)

If that scoping unit is a submodule and that name is made accessible by submodule association from an ancestor where the name is established to be specific.

In 14.1.2.4.1

In numbered paragraph (1), second line, after "appears" add ", is made accessible by submodule association,"

In numbered paragraph (2) third line, after "appears" add ", is made accessible by submodule association,"

In numbered paragraph (3), after the first “If” add “the name is accessible in the scoping unit by submodule association, “
In 14.1.2.4.2
In both paragraphs (1) and (2) first lines, before “interface” add “non-forward”
Add paragraph (6), and renumber
If the scoping unit is a submodule that accesses the procedure by submodule association by that name.
In 14.6.1
In first line replace “three” by “four”, before “and” add “submodule association,”
In second line, before “and” add “submodule,”
Add a new unit 14.6.1.3 and renumber

14.6.1.3 Submodule association

Submodule association is the association of names in the scoping unit of a submodule with those in its parent. The rules of submodule association are given in 11.3.4. Submodule association allows a child submodule to access entities defined in its ancestors. Such an association remains in effect throughout the execution of the program.

Annex A changes

Insert the following definitions into the glossary in alphabetical order:
ancestor (11.2.3):A module, a submodule, or an ancestor of the parent of that submodule.
child (11.2.3):A submodule, when considered in its relation to the module or submodule upon which it depends.
descendant (11.2.3):A module, a submodule, or a descendant of a child of that module or submodule.
forward interface (12.3.2.1):An interface defined by an interface body with a FORWARD prefix. It declares the interface for a module procedure that has a separately-defined implementation body in a descendant module or submodule.
parent (11.2.3):A module or submodule, when considered in its relation to the submodules that depend upon it. The module or submodule named as the parent in the submodule statement that introduces the particular submodule.
separate module procedure (12.5) A module procedure that defines an implementation body for a procedure that has a forward interface.
submodule (2.2.5,11.2.3):A program unit that depends on a module or another submodule; it extends the program unit on which it depends.
submodule association (14.6.1.3) The association of names accessed in a submodule that are specified in an ancestor.

Annex B no changes

Annex C changes

Add subsection

C.8.3.8 Submodule use

The example module `POINTS` below declares a type `POINT` and a forward interface body for a separate module function `POINT_DIST`. Because the interface body includes the `FORWARD` prefix, the interface body accesses the scoping unit of the module by host association.

```
MODULE POINTS

  TYPE :: POINT
    REAL :: X, Y
  END TYPE POINT

  INTERFACE
    FORWARD FUNCTION POINT_DIST( A, B )
```

```
      TYPE(POINT), INTENT(IN) :: A, B ! Accessed by host association
      REAL :: POINT_DIST
    END FUNCTION POINT_DIST
  END INTERFACE

END MODULE POINTS
```

The example submodule `POINTS_A` below is a submodule of the `POINTS` module. The scope of the type name `POINT` and the name and characteristics of the function `POINT_DIST` extend into the submodule by submodule association. The characteristics of the function `POINT_DIST` can be fully confirmed by redeclaration in the separate module function body, or taken from the forward interface body in the `POINTS` module.

```
SUBMODULE ( POINTS ) POINTS_A
CONTAINS

  SEPARATE FUNCTION POINT_DIST( A, B ) ! complete redeclaration
    TYPE(POINT), INTENT(IN) :: A, B    ! of the interface confirming
    REAL :: POINT_DIST                 ! the characteristics from the parent
    POINT_DIST = SQRT( (A%X-B%X)**2 + (A%Y-B%Y)**2 )
  END FUNCTION POINT_DIST

END SUBMODULE POINTS_A
```

An alternative example of a coding of the submodule `POINTS_A` showing minimal redeclaration of the separate module function is

```
SUBMODULE ( POINTS ) POINTS_A
CONTAINS

  SEPARATE FUNCTION POINT_DIST() ! mimimal redeclaration of the interface
    POINT_DIST = SQRT( (A%X-B%X)**2 + (A%Y-B%Y)**2 )
  END FUNCTION POINT_DIST

END SUBMODULE POINTS_A
```

Programs may access the facilities defined in this module and submodule only by way of a USE statement that names the module; programs are prohibited from accessing submodules by use association since a submodule name may not appear in a USE statement. As a result a change made to the code in the submodule cannot affect the attributes or characteristics of the entities accessed from the module by use association. Therefore, a submodule change need not trigger a recompilation of any of the use associated program units.

4 Required editorial changes to, ISO/IEC 1539-1 : 2004

The following editorial changes to ISO/IEC 1539-1 : 2004, if implemented, would provide the facilities described in foregoing section of this report as an extension to Fortran 2003. The changes are grouped by the major sections to which they apply.

Section 2 changes

In 2.1

After the third right-hand-side of syntax rule R202 insert:

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.

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

end-submodule-stmt

In the second line of the first paragraph of subclause 2.2 insert “,a submodule ”after “module ”
In the fourth line of the first paragraph of subclause 2.2 insert a new sentence:
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 ”
In the penultimate line of the first paragraph of subclause 2.2 insert “or submodule ”after “module ”
Replace the second sentence of 2.2.3.2 by the following sentence
A module procedure may be invoked from within any scoping unit that accesses its declaration (12.3.2.1) or definition (12.5).
Insert the following note at the end of 2.2.3.2.
NOTE 2.2
The scoping unit of a submodule accesses the scoping unit of its parent module or submodule by submodule association.
Insert a new subclause:

2.2.5 Submodule

A submodule is a program unit that extends a module or submodule. It contains definitions (12.5) for separate module procedures whose forward interfaces are declared (12.3.2.1) in its parent module or submodule. It may also contain declarations and definitions of entities that are accessible to descendant submodules. An entity declared in a submodule is not accessible by use association, but a procedure that is declared in a module and defined in one of that module ’s descendants is accessible by use association.
In the second line of the first row of Table 2.1 insert “,SUBMODULE ”after “MODULE ”
Change the heading of the third column of Table 2.2 from “Module ”to “Module or Submodule ”
In the second footnote to Table 2.2 insert “or submodule ”after “module ”and change “the module ”to “it ”
In the last line of 2.3.3 insert “,*end-submodule-stmt*” after “*end-module-stmt* ”
In the first line of the second paragraph of 2.4.3.1.1 insert “,submodule ”after “module ”
In 2.5.3, first paragraph, after “use association” insert “, submodule association”

Section 3 changes

At the end of 3.3.1,immediately before 3.3.1.1,add “END SUBMODULE ”to the list of adjacent keywords where blanks are optional, alphabetize the list.

Section 4 changes

In the third paragraph of 4.5.1.1, after “definition ”add “and its descendant submodules ”
In the last line of Note 4.19, after “definition ”add “and its descendant submodules ”
In the third line of the second paragraph of 4.5.5.2 insert “or submodule ”after “module ”. In the third and fourth line, replace “referencing the module ”by “that has access to that program unit ”
In the first line of the second paragraph of Note 4.49,insert “or submodule ”after “module ”

Section 5 changes

In constraint C532 insert “or submodule ”after “module ”
In the first line of the second paragraph of 5.1.2.12 insert “,or any of its descendant submodules ”after “attribute ”
In the first line of the second paragraph of 5.1.2.13 insert “or any of its descendant submodules ”after “module ”.
In the last line of the same paragraph, after “module” add “ or submodule”
In constraint C559 insert “or submodule ”after “module ”

Section 6 changes

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,”

In the first line of the first paragraph after Note 6.23 insert “or submodule ”after “module ”

Sections 7-10 no changes

Section 11 changes

First paragraph, after “a module,” add “ a submodule,”

In the first line of the second paragraph insert “,submodules ”after “modules ”

After constraint C1109 insert an additional constraint:

C1109a (R1109) If the USE statement appears within a submodule, *module-name* shall not be the name of the ancestor module of the submodule.

Add new subsection 11.2.2 and renumber as necessary

11.2.2 Submodules

A **submodule** is a program unit that is subsidiary to and dependent on a module or another submodule, termed its **parent**. This parent is identified by the *parent-name* in the submodule statement that introduces the submodule. A submodule is a **child** of its parent. An **ancestor** of a submodule is itself and any ancestors of its parent. A descendant of a module or a submodule is itself and any descendants of its child submodules.

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

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

R1115c *end-submodule-stmt* **is** END [SUBMODULE [*submodule-name*]]

C1114a R(1115b) The *parent-name* shall be the name of a submodule or a nonintrinsic module.

C1114b R(1115b) The *submodule-name* shall not be the same as *parent-name*.

C1114c R(1115c) If a *submodule-name* is specified in the *end-submodule-stmt* it shall be identical to the *submodule-name* specified in the *submodule-stmt*.

C1114d R(1115a) submodule *specification-part* shall not contain a *stmt-function-stmt*, an *entry-stmt*, or a *format-stmt*.

C1114e R(1115a) An automatic object shall not appear in the *specification-part* of a submodule.

C1114f R(1115a) If an object of a type for which *component-initialization* is specified appears in the specification-part of a submodule and does not have the ALLOCATABLE or POINTER attribute, the object shall have the SAVE attribute.

C1114g R(1115b) A *submodule-name* shall not appear as *module-name* in a USE statement.

A submodule name must be different from that of every other module or submodule. A submodule may contain USE statements by which it may access modules other than its ancestor module.

A submodule accesses named entities from its parent by submodule association (11.2.3).

A submodule may provide implementations for separate module procedures that are declared by forward interface bodies in ancestor program units. A submodule may also declare and define other entities that are accessible in descendant submodules. At most one separate module procedure shall provide a definition for any forward interface in any particular set of descendant program units.

11.2.4 Submodule association

The scoping unit of a submodule accesses all named data objects, derived types, interface blocks, procedures, generic identifiers, and namelist groups accessible in its parent by submodule association, regardless of the accessibility attributes.

Any reference to the name of a submodule associated entity within the submodule is a reference to the entity accessed from the parent program unit.
A submodule associated entity may be redeclared, in part or completely, in the submodule but such redeclaration shall confirm the attributes, characteristics, and values (if any) of the associated parent entity. Any attributes, characteristics or values not confirmed by redeclaration remain as specified in the ancestor program unit.

Section 12 changes
In the fourth paragraph before "host association" add "submodule association, "
In 12.1.2.2, third paragraph before "subprogram" add "or submodule "
At the end of the first paragraph of 12.3 add the sentence "However, iIf the dummy arguments are redeclared in a separate module procedure body (12.5.2.5) they shall have the same names as in the corresponding forward interface body (12.3.2.1)."
In 12.3.2.1,
In rule R1205 before "function-stmt" and "subroutine-stmt" add "[FORWARD] "
Before constraint C1207 add
C1206a (R1205) The FORWARD prefix shall appear only in an *interface-body* within a module or submodule.
Add additional constraint,
C1211a (R1209) An IMPORT statement shall not appear within an interface body that is declared with a FORWARD prefix.
Add the following as a new fourth paragraph after the constraint
If the interface body is introduced by a statement with the FORWARD prefix it is a **forward interface body**. A forward interface body declares the interface for a module procedure that shall be defined by a separate module procedure within a descendant program unit. If the definition of a separate module procedure body corresponding to a particular forward interface does not appear within a descendant of the of the program unit in which the forward interface body is declared (11.2.3), the interface may be referenced but the procedure shall not be invoked.
In the existing fourth paragraph first sentence after "body" add " without the FORWARD prefix"
At the end of rule R1228 add line
or SEPARATE
Add further constraints before rule 1229
C1243a (R1228) If SEPARATE is present the *function-name* shall be that of a forward interface body declared in an ancestor module or submodule.
C1243b R(1228) SEPARATE shall not appear for an internal procedure or an external procedure; it may appear only for a module procedure in a module or submodule.
Add paragraph before the end of subsection
If a *prefix-spec* of SEPARATE appears, the subprogram defines a separate module procedure whose interface shall have been declared in a forward interface body in an ancestor module or submodule. A separate module procedure is accessible by use association if and only if its forward interface body is accessible by use association. For a particular forward interface body at most one separate module procedure shall occur in any set of descendant program units.
In 12.5.2.2,
Add following rule R1231
C1248a R(1231) If SEPARATE is present the *subroutine-name* shall be that of a forward interface body declared in an ancestor module or submodule.
Insert a new subclause before 12.5.2.4 and renumber succeeding subclauses appropriately
12.5.2.4 Separate module procedure definition

A **separate module procedure** is a module procedure for which the interface is declared by a forward interface body (12.3.2.1) in the *specification-part* of a module or submodule. The procedure body that defines an implementation of a separate module procedure must be in a descendant of the program unit in which the forward interface body is declared.
NOTE 12.40a
A separate module procedure can be accessed by use association if and only if its interface body can be accessed by use association. A separate module procedure that is not accessible by use association might still be accessible by way of a procedure pointer, a dummy procedure, or a type-bound procedure.
A module subprogram that defines a separate module procedure may redeclare the characteristics declared in its interface body. If they are redeclared, they shall be identical to those specified in its interface body and hence merely confirm the characteristics referenced from the forward interface body, The one exception is that the module procedure may be specified to be pure even if the interface body does not so specify.
NOTE 12.40b
Specifications within an interface body that do not determine characteristics or dummy argument names have no effect. Therefore, if a separate module procedure is to be recursive, or it is to have a result name different from the function name, these properties must necessarily be specified within the separate module subprogram.
In the first line of the first paragraph after syntax rule R1236 in 12.5.2.6 insert ",submodule "after "module "

Section 13-15 no changes

Section 16 changes
In item (1)in the first list in 16.2,after "abstract interfaces "insert ",forward interfaces "
Before Note 16.3 add a new paragraph
A separate module procedure shall have the same name as its forward interface. A submodule accesses all available entities from its parent by submodule association. Such entities may have their attributes and characteristics confirmed by redeclaration within the submodule but all references to the names of such entities are to the associated entity.
In 16.4.1, first sentence, replace "five" by "six", after "use association," add " submodule association,"
Add new item 16.4.1.3 and renumber as necessary
16.4.1.3 Submodule association
Submodule association is the association of names in the scoping unit of a submodule with those in its parent. The rules of submodule association are given in (11.3.4). Submodule association allows a child submodule to access entities defined in its ancestors. Such an association remains in effect throughout the execution of the program. Submodule association does not demand but it permits redeclaration of associated entities but such redeclaration shall confirm the attributes, characteristics, definition status and value, if any, of the associated entity. Any reference to the named entity within the scoping unit of a submodule is a reference to the associated entity regardless of redeclaration and the entity has the attributes, definition status, value (if any), and characteristics of the parent entity.
In the first line of the existing first paragraph of 16.4.1.3 insert ",a forward interface body "after "module subprogram ".In the second line, insert "that is not a forward interface body "after "interface body "
In item 2 of 16.5.6 insert "or submodule "after "module "
In item 3c of 16.5.6 insert "or submodule "after the first "module "and replace the second "module "by "that scoping unit "
Replace Note 16.18 by the following
NOTE 16.18

A module subprogram inherently references the module or submodule that is its host. Therefore, for processors that keep track of when modules or submodules are in use, one is in use whenever any procedure in it or any of its descendant submodules is active, even if no other active scoping units reference its ancestor module; this situation can arise if a module procedure is invoked via a procedure pointer or by means other than Fortran.
In item 3d of 16.5.6 insert “or submodule ”after the first “module ”and replace the second “module ” by “that scoping unit ”

Annex A changes
Insert the following definitions into the glossary in alphabetical order:
ancestor (11.2.2):A module, a submodule, or an ancestor of the parent of that submodule.
child (11.2.2):A submodule, when considered in its relation to the module or submodule upon which it depends.
descendant (11.2.2):A module, a submodule, or a descendant of a child of that module or submodule.
forward interface (12.3.2.1):An interface defined by an interface body with a FORWARD prefix. It declares the interface for a module procedure that has a separately-defined implementation body in a descendant program unit.
parent (11.2.2):A module or submodule, when considered in its relation to the submodules that depend upon it. The module or submodule named as the parent in the submodule statement that introduces a particular submodule.
separate module procedure (12.5) A module procedure that defines an implementation for a procedure that has a forward interface.
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.
submodule association (14.6.1.3) The association of names accessed in a submodule that are specified in an ancestor.

Annex B no changes

Annex C changes
Insert a new subclause immediately before C.9
C.8.3.9 Modules with submodules
The example module `POINTS` below declares a type `POINT` and a forward interface body for a separate module function `POINT_DIST`. Because the interface body includes the `FORWARD` prefix, the interface body accesses the scoping unit of the module by host association.

```
MODULE POINTS

  TYPE :: POINT
    REAL :: X, Y
  END TYPE POINT

  INTERFACE
    FORWARD FUNCTION POINT_DIST( A, B )
      TYPE(POINT), INTENT(IN) :: A, B ! Accessed by host association
      REAL :: POINT_DIST
    END FUNCTION POINT_DIST
  END INTERFACE

END MODULE POINTS
```

The example submodule `POINTS_A` below is a submodule of the `POINTS` module. The scope of the type name `POINT` and the name and characteristics of the function `POINT_DIST` extend into the submodule by submodule association. The characteristics of the function `POINT_DIST` can

be fully confirmed by redeclaration in the separate module function body, or taken from the forward interface body in the `POINTS` module.

```
SUBMODULE ( POINTS ) POINTS_A
CONTAINS

  SEPARATE FUNCTION POINT_DIST( A, B ) ! complete redeclaration
    TYPE(POINT), INTENT(IN) :: A, B    ! of the interface confirming
    REAL :: POINT_DIST                 ! the characteristics from the parent
    POINT_DIST = SQRT( (A%X-B%X)**2 + (A%Y-B%Y)**2 )
  END FUNCTION POINT_DIST

END SUBMODULE POINTS_A
```

An alternative example of a coding of the submodule `POINTS_A` showing minimal redeclaration of the separate module function is

```
SUBMODULE ( POINTS ) POINTS_A
CONTAINS

  SEPARATE FUNCTION POINT_DIST() ! mimimal redeclaration of the interface
    POINT_DIST = SQRT( (A%X-B%X)**2 + (A%Y-B%Y)**2 )
  END FUNCTION POINT_DIST

END SUBMODULE POINTS_A
```

Programs may access the facilities defined in this module and submodule only by way of a USE statement that names the module; programs are prohibited from accessing submodules by use association since a submodule name may not appear in a USE statement. As a result a change made to the code in the submodule cannot affect the attributes or characteristics of the entities accessed from the module by use association. Therefore, a submodule change need not trigger a recompilation of any of the use associated program units.