TECHNICAL REPORT

ISO/IEC TR 19767

First edition
2005-02-15

Information technology — Programming languages — Fortran — Enhanced Module Facilities

Technologies de l'information — Langages de programmation — Fortran — Facilités améliorées de module

Reference number
ISO/IEC TR 19767:2005(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 19767:2004, which is a Technical Report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces.*

0 Introduction

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 1539-1:2004.

The module system of Fortran, as standardized by ISO/IEC 1539-1:2004, while adequate for programs of modest size, has shortcomings that become evident when used for large programs, or programs having large modules. The primary cause of these shortcomings is that modules are monolithic.

This technical report extends the module facility of Fortran so that program developers can optionally encapsulate the implementation details of module procedures in **submodules** that are separate from but dependent on the module in which the interfaces of their procedures are defined. If a module or submodule has submodules, it is the **parent** of those submodules.

The facility specified by this technical report is compatible to the module facility of Fortran as standardized by ISO/IEC 1539-1:2004.

It is the intention of ISO/IEC JTC1/SC22 that the semantics and syntax specified by this technical report be included in the next revision of the Fortran International Standard 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.

0.1 Shortcomings of Fortran's module system

The shortcomings of the module system of Fortran, as specified by ISO/IEC 1539-1:2004, and solutions offered by this technical report, are as follows.

0.1.1 Decomposing large and interconnected facilities

If an intellectual concept is large and internally interconnected, it requires a large module to implement it. Decomposing such a concept into components of tractable size using modules as specified by ISO/IEC 1539-1:2004 may require one to convert private data to public data. The drawback of this is not primarily that an "unauthorized" procedure or module might access or change these entities, or develop a dependence on their internal details. Rather, during maintenance, one must then answer the question "where is this entity used?"

Using facilities specified in this technical report, such a concept can be decomposed into modules and submodules of tractable size, without exposing private entities to uncontrolled use.

Decomposing a complicated intellectual concept may furthermore require circularly dependent modules, but this is prohibited by ISO/IEC 1539-1:2004. It is frequently the case, however, that the implementations of some parts of the concept depend upon the interfaces of other parts. Because the module facility defined by ISO/IEC 1539-1:2004 does not distinguish between the implementation and interface, this distinction cannot be exploited to break the circular dependence. Therefore, modules that implement large intellectual concepts tend to become large, and thus expensive to maintain reliably.

Using facilities specified in this technical report, complicated concepts can be implemented in submodules that access modules, rather than modules that access modules, thus reducing the possibility for circular dependence between modules.

0.1.2 Avoiding recompilation cascades

Once the design of a program is stable, few changes to a module occur in its **interface**, that is, in its public data, public types, the interfaces of its public procedures, and private entities that affect their definitions. We refer to the rest of a module, that is, private entities that do not affect the definitions of public entities, and the bodies of

its public procedures, as its **implementation**. Changes in the implementation have no effect on the translation of other program units that access the module. The existing module facility, however, draws no structural distinction between the interface and the implementation. Therefore, if one changes any part of a module, most language translation systems have no alternative but to conclude that a change might have occurred that could affect the translation of other modules that access the changed module. This effect cascades into modules that access modules that access the changed module, and so on. This can cause a substantial expense to retranslate and recertify a large program. Recertification can be several orders of magnitude more costly than retranslation.

Using facilities specified in this technical report, implementation details of a module can be encapsulated in submodules. Submodules are not accessible by use association, and they depend on their parent module, not vice-versa. Therefore, submodules can be changed without implying that a program unit accessing the parent module (directly or indirectly) must be retranslated.

It may also be appropriate to replace a set of modules by a set of submodules each of which has access to others of the set through the parent/child relationship instead of USE association. A change in one such submodule requires the retranslation only of its descendant submodules. Thus, compilation and certification cascades caused by changes can be shortened.

0.1.3 Packaging proprietary software

If a module as specified by International Standard ISO/IEC 1539-1:2004 is used to package proprietary software, the source text of the module cannot be published as authoritative documentation of the interface of the module, without either exposing trade secrets, or requiring the expense of separating the implementation from the interface every time a revision is published.

Using facilities specified in this technical report, one can easily publish the source text of the module as authoritative documentation of its interface, while witholding publication of the source text of the submodules that contain the implementation details, and the trade secrets embodied within them.

0.1.4 Easier library creation

Most Fortran translator systems produce a single file of computer instructions and data, frequently called an *object file*, for each module. This is easier than producing an object file for the specification part and one for each module procedure. It is also convenient, and conserves space and time, when a program uses all or most of the procedures in each module. It is inconvenient, and results in a larger program, when only a few of the procedures in a general purpose module are needed in a particular program.

Modules can be decomposed using facilities specified in this technical report so that it is easier for each program unit's author to control how module procedures are allocated among object files. One can then collect sets of object files that correspond to a module and its submodules into a library.

0.2 Disadvantage of using this facility

Translator systems will find it more difficult to carry out global inter-procedural optimizations if the program uses the facility specified in this technical report. Interprocedural optimizations involving procedures in the same module or submodule will not be affected. When translator systems become able to do global inter-procedural optimization in the presence of this facility, it is possible that requesting inter-procedural optimization will cause compilation cascades in the first situation mentioned in subclause 0.1.2, even if this facility is used. Although one advantage of this facility could perhaps be reduced in the case when users request inter-procedural optimization, it would remain if users do not request inter-procedural optimization, and the other advantages remain in any case.

Information technology — Programming languages — Fortran — Enhanced Module Facilities

1 General

1.1 Scope

This technical report specifies an extension to the module facilities of the programming language Fortran. The Fortran language is specified by International Standard ISO/IEC 1539-1:2004 : Fortran. The extension allows program authors to develop the implementation details of concepts in new program units, called **submodules**, that cannot be accessed directly by use association. In order to support submodules, the module facility of International Standard ISO/IEC 1539-1:2004 is changed by this technical report in such a way as to be upwardly compatible with the module facility specified by International Standard ISO/IEC 1539-1:2004.

Clause 2 of this technical report contains a general and informal but precise description of the extended functionalities. Clause 3 contains detailed instructions for editorial changes to ISO/IEC 1539-1:2004.

1.2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 1539-1:2004 : *Information technology – Programming languages – Fortran – Part 1: Base language*

2 Requirements

The following subclauses contain a general description of the extensions to the syntax and semantics of the Fortran programming language to provide facilities for submodules, and to separate subprograms into interface and implementation parts.

2.1 Summary

This technical report defines a new entity and modifications of two existing entities.

The new entity is a program unit, the *submodule*. As its name implies, a submodule is logically part of a module, and it depends on that module. A new variety of interface body, a *module procedure interface body*, and a new variety of procedure, a *separate module procedure*, are introduced.

By putting a module procedure interface body in a module and its corresponding separate module procedure in a submodule, program units that access the interface body by use association do not depend on the procedure's body. Rather, the procedure's body depends on its interface body.

2.2 Submodules

A **submodule** is a program unit that is dependent on and subsidiary to a module or another submodule. 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 host association.

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.

A submodule is introduced by a statement of the form SUBMODULE (*parent-identifier*) *submodule-name*, and terminated by a statement of the form END SUBMODULE *submodule-name*. The *parent-identifier* is either the name of the parent module or is of the form *ancestor-module-name* : *parent-submodule-name*, where *parent-submodule-name* is the name of a submodule that is a descendant of the module named *ancestor-module-name*.

Identifiers declared in a submodule are effectively PRIVATE, except for the names of separate module procedures that correspond to public module procedure interface bodies (2.3) in the ancestor module. 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. ISO/IEC 1539-1:2004 permits PRIVATE and PUBLIC declarations only in a module, and this technical report does not propose to change this.

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 all other respects, a submodule is identical to a module.

2.3 Separate module procedure and its corresponding interface body

A **module procedure interface body** specifies the interface for a separate module procedure. It is different from an interface body defined by ISO/IEC 1539-1:2004 in three respects. First, it is introduced by a *function-stmt* or *subroutine-stmt* that includes MODULE in its *prefix*. Second, it specifies that its corresponding procedure body is in the module or submodule in which it appears, or in one of its descendant submodules. Third, it accesses the module or submodule in which it is declared by host association.

A **separate module procedure** is a module procedure whose interface is declared in the same module or submodule, or is declared in one of its ancestors and is accessible from that ancestor by host association. The module

subprogram that defines it may redeclare its characteristics, whether it is recursive, and its binding label. If any of these are redeclared, the characteristics, corresponding dummy argument names, whether it is recursive, and its binding label if any, shall be the same as in its module procedure interface body. The procedure is accessible by use association if and only if its interface body is accessible by use association. It is accessible by host association if and only if its interface body is accessible by host association.

If the procedure is a function and its characteristics are not redeclared, 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.

2.4 Examples of modules with submodules

The example module **POINTS** below declares a type **POINT** and a module procedure interface body for a module function **POINT_DIST**. Because the interface body includes the MODULE prefix, it accesses the scoping unit of the module by host association, without needing an IMPORT statement; indeed, an IMPORT statement is prohibited.

```
MODULE POINTS
  TYPE :: POINT
    REAL :: X, Y
  END TYPE POINT

  INTERFACE
    REAL MODULE FUNCTION POINT_DIST ( A, B ) RESULT ( DISTANCE )
      TYPE(POINT), INTENT(IN) :: A, B ! POINT is accessed by host association
      REAL :: DISTANCE
    END FUNCTION POINT_DIST
  END INTERFACE
END MODULE POINTS
```


The example submodule **POINTS_A** below is a submodule of the POINTS module. The type **POINT** and the interface **POINT_DIST** are accessible in the submodule by host association. The characteristics of the function POINT_DIST are redeclared in the module function body, and the dummy arguments have the same names. The function **POINT_DIST** is accessible by use association because its module procedure interface body is in the ancestor module and has the PUBLIC attribute.

```
SUBMODULE ( POINTS ) POINTS_A
  CONTAINS
    REAL MODULE FUNCTION POINT_DIST ( A, B ) RESULT ( DISTANCE )
      TYPE(POINT), INTENT(IN) :: A, B
      DISTANCE = SQRT( (A%X-B%X)**2 + (A%Y-B%Y)**2 )
    END FUNCTION POINT_DIST
END SUBMODULE POINTS_A
```

An alternative declaration of the example submodule POINTS_A shows that it is not necessary to redeclare the properties of the module procedure POINT_DIST.

```
SUBMODULE ( POINTS ) POINTS_A
  CONTAINS
    MODULE PROCEDURE POINT_DIST
      DISTANCE = SQRT( (A%X-B%X)**2 + (A%Y-B%Y)**2 )
    END PROCEDURE POINT_DIST
END SUBMODULE POINTS_A
```

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

The following editorial changes, if implemented, would provide the facilities described in foregoing clauses of this report. Descriptions of how and where to place the new material are enclosed between square brackets.

[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

[Add another alternative to R1108:]

or *separate-module-subprogram*

[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".]

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

[Insert a new subclause:]

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".]

[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 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*".]

[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".]

[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.]

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

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

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

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

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

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

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

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

[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".]

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

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

[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 two lines of the first paragraph after Note 6.23 insert "or submodule" after "module" twice.]

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

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

[Add another alternative to R1108]

or *separate-module-subprogram*

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

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

