Subject: Extensible modules
From: Van Snyder
References: 98-105, 04-279

1 Number

TBD

2 Title

Extensible modules.

3 Submitted By

J3

4 Status

For consideration.

5 Basic Functionality

Provide for modules to be extended, in a way analogous to type extension. An extension module would access its parent by host association, and be accessible by use association. Private components and bindings of extensible types do not — cannot — become public in an extension type. Similarly, private entities of an extensible module would not — could not — become public entities of an extension module.

6 Rationale

In 1998, the reason given to reject the proposal to include extensible modules in the Enhanced Modules Technical Report was that it compromised the integrity of the PRIVATE attribute. After deciding at meeting 168 that the PRIVATE attribute referred only to the accessibility of names by USE association, the new reason given to reject the proposal, which was outlined in paper 04-279, was a desire to think about it more. Paper 04-279 provided a superficial rationale. This paper provides a deeper rationale.

One of the great strengths of Fortran is the module. Technical Report 19767 notices that although Fortran's module system is powerful, it has defects for large programs, or programs with large modules. These defects are remedied by that technical report. It allows the interface of a module to be separated from its implementation, which in turn enables the details of the implementation and the module's clients to be compiled separately without interference provided the specification remains stable.

Although this improves the situation for moderately large programs, it is still cumbersome for very large or complex programs, or programs that are in use for a long time, thereby undergoing continuing development. There are two problems: The coarse control of visibility of private entities, and the inability to extend without recompilation.

There are occasions when one wishes to write two logically distinct packages that nonetheless share a private entity. One can not do this with Fortran 95 or Fortran 2003, even with Technical Report 19767. One either needs to make the type public so that both modules can see it, with the unfortunate consequence that all of the client modules can see it; this breaks the abstraction. Or, if one wishes to keep the abstraction, one needs to merge the two modules together into a single module — or module and system of submodules if the facilities described in Technial Report 19767 are available. This results in a large monolithic package with increased compilation and maintenance costs.

The other aspect of the problem arises when one wishes to extend an existing module by adding more facilities to it. If one adds to a module specification one has to recompile it, and moreover recompile all of the module's clients even if the changes to the module have no effect on them. In addition to all

of the undesirable results laid out here and in Technical Report 19767, modifying a module in order to extend its functionality or to extend one of its types is impossible if one doesn't have the source text for it.

One goal of providing extensible types is to be able to extend the functionality of a type and its associated operations without modifying the type definition. If the type has any private components or bindings, or any of its type-bound procedures access private entities, in Fortran 2003 — even with the facilities described in Technical Report 19767 — it is almost certainly necessary to modify the module containing the type's definition and related private entities in order to provide the extension.

These problems could be solved by allowing modules to be extended in a way analogous to how types can be extended. This proposal would allow a hierarchy of program units accessible by USE association. As is the case with submodules, extension modules would be able to access their parent, including its private entities, by host association. Private entities accessed by USE association would not thereby become public entities, even if they are accessed by host extension in an extension module, which is itself accessible by USE association. This extension mechanism fits neatly with type extension.

An interesting point is that clients that access the parent module by USE association have access only to the names of entities declared within the parent module, but by way of type extension and polymorphic objects they might nonetheless have indirect access to entities of child modules. Clients that access a child module by use association would have access both to public entities of that module, and to public entities of its ancestor modules, without needing to access the ancestors explicitly by USE association. In particular, access to extension types declared in child modules gives access to public components of objects for which those types are the declared types.

7 Estimated Impact

This is a modest project, of substantially lesser scope than submodules. Technical Report 19767 includes over five pages of normative edits, and another five pages of nonnormative edits. It is unlikely that a final version of this proposal will require more than $1\frac{1}{2}$ pages of normative edits. The most difficult problem for implementors will be to provide for an extension module to access its parent by host association, but that problem needs to be solved anyway to implement Technical Report 19767.

8 Detailed Specification

Provide that a module is extensible by default, and for specifications that either confirm that or specify that a module is not extensible.

Provide for specification that a module is an extension of another module. Provide that an extension is by default extensible, and for specifications that either confirm that or specify that an extension module is not extensible.

Similarly to what we did with types, we may ultimately decide that all modules are extensible, and that these specifications are therefore not necessary. The detailed edits proposed in Section 8.1 below would thereby be somewhat simplified.

Examples:

```
module, extensible :: MyModule

module, nonextensible :: MyModule

module, extends(myModule) :: MyExtension

module, extends(myModule), nonextensible :: MyDeadEnd
```

Unlike type extension, provide that an extension module accesses its parent by host association.

Provide that an extension module is accessible by use association.

There is a detailed example in 98-105, and an illustration in Section 10 below.

8.1 Edits w.r.t. 04-007

A few more edits than these, but not many more, may be necessary. The primary purpose for providing these is to give an idea of the scope of the project. Final wording is, of course, TBD.

[After "definition" insert "and that module's extensions".]	46:10
[After "defined" insert "and that module's extensions".]	46:10+7
[After "definition" insert "and that module's extensions".]	55:10
[After "module" insert "and its extensions".]	55:10+17
[After "definition" insert "and its extensions".]	56:0+4
[After "definition" insert "and its extensions".]	58:8
[After "module" insert "or any of its extensions".]	59:24
[After "module" insert "or any of its extensions".]	60:4+5
[After "attribute" insert "or its extensions".]	84:3
[After "module" insert "or any of its extensions".]	115:10
C548$\frac{1}{2}$ (R518) The PUBLIC attribute shall not be specified for an *access-id* with the PRIVATE attribute that is accessed by host association.	86:8+
Default public accessibility does not apply to private entities accessed by host association.	86:19+ New ¶

R1105	*module-stmt*	**is**	MODULE [[, *module-attrib-list*] ::] *module-name*	250:11
R1105a	*module-attrib*	**is**	EXTENSIBLE	
		or	NONEXTENSIBLE	
		or	EXTENDS (*module-name*)	

C1103a (R1105a) If EXTENSIBLE appears, NONEXTENSIBLE shall not appear.

C1103b (R1105a) The *module-name* shall be the name of an extensible module.

A module that is introduced by a MODULE statement in which the NONEXTENSIBLE attribute does not appear is an **extensible module**. A module that is introduced by a MODULE statement in which the EXTENDS attribute appears is an **extension module**. It extends another module. The module that it extends is its **parent**; its parent is specified by the *module-name* in the EXTENDS attribute. An **ancestor** of an extension module is its parent, or an ancestor of its parent. An extension module accesses is parent by host association.	251:4+
[After "itself" insert "or any of its ancestors by use association".]	251:8
C1110$\frac{1}{2}$ (R1109) If the USE statement appears within an extension module, *module-name* shall not be the name of that extension module or any of its ancestor modules.	251:34+
[After "body." insert a new sentence "An extension module has access to its parent by host association."]	411:4
[After "module" insert "or any of its extensions".]	415:14

If we keep the glossary, definitions for "extension module" and "extensible module" will be needed, and the definition of "parent" given in the Modules TR will need embellishment.

An example in Annex C may be desirable. One can be plagiarized from 98-105.

In the several places in the Modules TR where it says "the ancestor module," the wording will need to be changed to "an ancestor module". Some small amount of work may be necessary concerning separate procedures.

9 History

10 Illustration of relation of 007, 19767 and 04-341