Comment by Daniel Grimwood

To: J3
From: Craig Dedo
Date: November 11, 2002
Subject: Comment by Daniel Grimwood

ISO/IEC JTC1/SC22/WG5 N1499

Comment on Fortran 2000 CD from Australia

Draft Australian/New Zealand Standard Comment Form
Section 1 - Draft Details
To: ISO/IEC JTC 1/SC 22

Mr/Mrs/Ms: Mr

First Name: Daniel

Last Name: Grimwood

Position Title: Research Officer

Organization Details: Department of Chemistry
The University of Western Australia

Street Number and Name: 35 Stirling Highway

Suburb: Crawley

State: WA

Postcode: 6009

Contact Details:
Email : reaper@theochem.uwa.edu.au
Phone : +61 8 93803138
Fax : +61 8 93801005

Subject Title: Fortran 2000 (ISO/IEC 1539-1) draft

Date of Comment: 31/10/2002

Section 2 - Comments

C.8.2.1
Lines 18-22
It is difficult to write makefiles and scripts that work with module information files, since although vendors base their naming schemes on the name of the modules, some convert the names to uppercase/lowercase, and different vendors use different file extensions. Modern

filesystems distinguish between upper and lower case, so it is important that the mapping from Fortran to filesystem is standardized for such cases. It is recommended that this either be the name of the module converted to lower case or upper case, or be the name of the module exactly how it is declared. Most but not all vendors give the files a .mod suffix, so it would be trivial to standardize this aspect of the file name.

The proposed change is to append an extra sentence to the paragraph, such as:

> If permitted by the filesystem, the file should be named as the lowercase name of the module with a .mod suffix appended.

Under the proposed change, module "ThIsModULE" produces the file called "thismodule.mod".

C.8.2.1
Lines 23-28
In this paragraph a choice is given as to whether a module information file should contain all information about a module, or include pointers back to other files that contain some of the information. It appears that all vendors have chosen the route of putting all information into the file and to not use file pointers. Thus it would be minimal inconvenience to remove the choice and force vendors to not use pointers to other files. The benefit is that currently, for the method of using file pointers, it is near impossible to write automated dependency generators that use the "USE" statements of modules, since the file pointers mean that there are implicit dependencies that are not evident in the currently processed module. If there are no file pointers, then the dependencies become trivial.

Delete the paragraph and replace with:

> The result of translating a module must contain all public information specified in the module (including copies of public information originally specified in other modules).

C.8.2.2
Line 15
The last sentence acknowledges that if the public attributes to a module do not change, then other modules that use that module should not need to be recompiled. Unfortunately, some vendors insert the compilation date into translated modules, and this may be the only difference between two translations of the same module (even if no public or private attributes have changed). This unnecessarily reduces the ability to avoid recompiling other modules that use that module.

Add the following sentence to the end of the paragraph:

> If neither the public or private attributes differ between two versions of a module, then the results of translating the two

versions should be identical.

5.2.1
Page 84, Line 22
This comment is not backward-compatible with previous Fortran standards, but it is better to fix this in the Fortran 2000 standard than to leave it broken indefinitely.

Having a default accessibility of "public" for modules means that, by default, as more modules that USE each other are added to a chain of such modules, the name space of publicly available entities becomes larger and the time/space taken to compile the last modules of the chain becomes unacceptably large. This problem grows indefinitely, until the programmer overrides this default with a "private" statement. This is a simple solution, but most new users to Fortran would be unaware of it. Furthermore, it is difficult for newcomers to debug conflicting names of entities when no such entities are present in modules they have explicitly USEd, but rather are defined in implicitly USEd modules. It would be better to adopt the philosophy that if you want to propagate an entity between modules, then do so manually, and if you want to use an entity in another module, then USE that module explicitly.

Change the phrase "the default is public" to "the default is private".

References
02-007r3, Fortran 2000 Committee Draft

[End of J3 / 02-322]