Document number P0529R0

Date 2016-11-23

Author Richard Smith <richard@metafoo.co.uk>

Audience Evolution

Wording changes for proposed Modules TS extensions

Wording for portions of P0273R0

Introduction

This paper provides wording changes for three extensions to the Modules TS proposed in P0273R0 and approved by EWG in Jacksonville:

- The *module-declaration* specifying that a translation unit is a module must be the first declaration in the source file. Vote: **6 | 14 | 9 | 2 | 0**[1]
- Module partitions, allowing a single module (and its notion of module ownership) to be split across multiple interface files. Vote: **11 | 12 | 3 | 4 | 0**
- Module implementations begin with "`module implementation M`" stanza rather than "`module M`". Vote: **10 | 11 | 8 | 3 | 0**

Consolidated wording is provided specifying all three features. The baseline for this wording is N4610 plus the proposed resolutions for issues from the Jacksonville modules issues list.

Wording

In 2.10 [lex.name] Table 2 ("Identifiers with special meaning"), add:

`implementation`
`partition`

Change grammar changes to 3.5 [basic.link] as follows (unchanged productions are not quoted):

translation-unit:
module-head$_{opt}$ *toplevel-declaration-seq*$_{opt}$

[1] Per WG21 convention for 5-way polls, these results are:
Strongly in favor | in favor | neutral | against | strongly against

module-head:
`module` *module-key*$_{opt}$ *module-name attribute-specifier-seq*$_{opt}$ `;`

module-key: one of
`implementation` `partition`

toplevel-declaration:
~~*module-declaration*~~
module-export-declaration
module-import-declaration
exported-fragment-group
global-module-declaration
proclaimed-ownership-declaration
declaration

~~*module-declaration:*~~
~~`module` *module-name attribute-specifier-seq*$_{opt}$ `;`~~

module-import-declaration:
`import` *module-name attribute-specifier-seq*$_{opt}$ `;`
`import partition` *string-literal* `;`

fragment:
module-import-declaration
global-module-declaration
declaration

global-module-declaration:
`module {` *toplevel-declaration-seq*$_{opt}$ `}`

module-name:
module-name-qualifier~~*-seq*~~$_{opt}$ *identifier*

~~*module-name-qualifier-seq:*~~
~~*module-name-qualifier .*~~
~~*module-name-qualifier-seq identifier .*~~

module-name-qualifier:
module-name-qualifier$_{opt}$ *identifier* `.`

Drafting note: the changes to module-name are just a simplification, with no normative impact.

Add a new paragraph before 3.5 [basic.link] paragraph 2:

A translation unit that has a *module-head* is a *module unit*, and is part of the module named by its *module-name*. A module unit with the *module-key* `implementation` is a *module implementation unit*; any other module unit is a *module interface unit*. For each *module-name* that is used by a program, there shall be exactly one module interface unit for that module with no *module-key*, which is the *principal module interface unit* for the module. The principal module interface unit shall have a transitive interface dependency (7.2.2) on each other module interface unit for the module.

Change in new bullet in 3.5 [basic.link] paragraph 2

— When a name has module linkage, the entity it denotes is owned by a module M and can be referred to by names from ~~other~~ scopes of module units of the same module ~~unit~~ (7.7) or from other scopes of the same translation unit ~~other module units part of M~~.

Drafting note: the changes here address the inconsistency in wording between this bullet and bullet 1, and the awkward "other module units part of M" phrasing.

Delete 7.7 [basic.modules] paragraph 1

~~A *translation-unit* shall contain at most one *module-declaration* [...]~~

Change in 7.7 [basic.modules] paragraph 2

A *module* is a collection of module units designating the same *module-name* in their *module-heads*~~, at most one of which contains~~. A module implementation unit (3.5) shall not contain *export-declaration*s or *exported-fragment-group*s or *module-export-declaration*s. ~~Such a distinguished module unit is called the module interface unit. Any other module unit is called a module implementation unit.~~

Change in 7.7 [basic.modules] paragraph 3

~~A module unit purview starts at the module-declaration and extends to the end of the translation unit.~~ The purview of a module unit comprises all declarations within that module unit that are not within a *global-module-declaration.* The purview of a module M is the set of module unit purviews of M's module units.

Change in 7.7 [basic.modules] paragraph 5

The *global module* is the collection of all declarations not in the purview of any module~~-declaration~~, including those declared within a *global-module-declaration* in a module unit. By extension, such declarations are said to be in the purview of the global module. [Note: The global module has no name and is not introduced by any module-declaration. A *global-module-declaration* does not establish a scope. — end note]

Change in 7.7.1 [dcl.module.interface] paragraph 1

[...] All entities with linkage other than internal linkage declared in a module interface unit of a module M are visible to all module implementation units of M. [...]

Change in 7.7.2 [dcl.module.import] paragraph 1

An *import-declaration* without the `partition` specifier makes exported declarations from the interface of the nominated module visible to name lookup in the current translation unit, in the same namespaces and contexts as in the nominated module. [Note: The entities are not redeclared in the translation unit containing the import-declaration. — end note] [Example]

Add a new paragraph after in 7.7.2 [dcl.module.import] paragraph 1

An *import-declaration* with the `partition` specifier makes declarations with external linkage from the nominated translation unit visible to name lookup in the current translation unit, in the same namespaces and contexts as in the nominated translation unit. The mapping from *string-literal*s to translation units is implementation-defined. The translation unit containing the import-declaration and the nominated translation unit shall be module interface units of the same module. [Example:

```
  // translation unit "a"
  module partition M;
  struct A {};

  // translation unit "b"
  module partition M;
  import partition "a";
  struct B : A {};

  // principal module interface unit
  module M;
  import partition "b";
  B *b; // OK
  A *a; // error: not visible, partition b does not re-export

  // module implementation unit
  module M;
  A make_a(); // OK, interface is implicitly visible
```

— end example]

Change in 7.7.2 [dcl.module.import] paragraph 2

A module M1 *has a dependency* on a module M2 if any module unit of M1 contains an *import-declaration* without the `partition` specifier nominating M2. A module shall not have a dependency on itself.

Change in 7.7.2 [dcl.module.import] paragraph 3

A module interface unit M1 has an interface dependency on a module interface unit M2 if the module interface unit of M1 contains an *import-declaration* nominating M2. A module interface unit shall not have a transitive interface dependency on itself.

End