Document Number: P0774r0
Date: 2017-10-02
To: SC22/WG21 CWG/EWG
Reply to: Nathan Sidwell
nathan@acm.org / nathans@fb.com
Re: Working Draft, Extensions to C ++ for Modules, n4681

Module Declaration Location

Nathan Sidwell

The current wording of n4681 does not place an ordering restriction on the location of the module declaration, it may appear at any point as a top-level declaration. This document presents an alternative, restricting it to be the first top-level declaration. Syntaxes are suggested that keep the existing contiguous nature of the global module portion of a module unit.

1 Background

The current modules draft, n4681, permits the module-declaration to appear as any top-level declaration. Declarations that precede it are part of the global module. The global module cannot be re-opened after the module-declaration.

At the Toronto ‘17 a desire was put forwards that the module declaration, if present, should be the first top-level declaration of a translation unit. This is not a new desire, p0529r1 (2016-11-23, R.Smith) presented such a scheme.

However, such schemes have required the ability to be more flexible than the current wording in regards to the global module. In particular allowing arbitrary reopening of the global module. This is undesirable.

A different Toronto suggestion was the ability to state at the beginning of a file that there would be a module declaration later. While such a scheme might satisfy some needs, it does seem a suboptimal solution. Once one has gone to the effort of stating there will be a module-declaration, the next ask might be the ability to say up front whether it is an interface or implementation unit. And finally the obvious question is ‘why can I not simply say what the module is?’

2 Proposal

I suggest amending the grammar so that the *module-declaration* must be first (after preprocessing). But further amend it so that a single optional global module block must immediately succeed it, if present. Two alternatives to achieve this are presented.

The semantics of the global module are unchanged.

2.1 Direct Combination

This approach amends the *module-declaration* itself so that it includes an optional global module block. Examples are:

```
// module interface with no global module fragment
export module foo;
// module interface declarations here
```

and

```
// module interface with global module fragment
export module foo using {
  // global module declarations here
}
// module interface declarations here
```

The existing '`using`' keyword is pressed into service joining the module declaration to the global module fragment. Naturally module implementation units follow the same pattern, but lack the initial '`export`' keyword in the module-declaration.

2.2 Explicit Global Module

While the first example satisfies the need of placing the module declaration first, and having a single global module fragment, it is a little awkward. It will be hard to explain why the two separate pieces are being glued together with a `using` keyword. Further, I suspect source code arrangements will be awkward, with the following being a common desired layout:

```
export module foo // first line of the file
// Expository text block
// Copyright & licensing
// Authorship
using {
  // global module fragment
}
// module contents continue
```

The desire for a standard comment block immediately after the module announcement separates it from the global module fragment. Where should the `using` keyword be placed? There appears to be no good answer.

Thus, rather than have a combined module & global fragment declaration, use a separate mechanism for declaring the global module fragment, but grammatically restrict it to immediately follow the *module-declaration*. With such a scheme, the above example becomes:

```
export module foo;
// Expository text block
// Copyright & licensing
// Authorship
module {
 // global module fragment
}
// module contents continue
```

2.3 Commonalities

The purview of the module starts after the global module fragment, which means the global module fragment continues to precede the purview. The current draft species the purview starts at the *module-declaration*, leaving it ambiguous as to whether (parts of) the *module-declaration* itself is within its own purview.

In both alternatives, the global module fragment is contained within a brace pair, without trailing semi-colon. This matches existing syntax for a *namespace-definition* and *namespace-alias*, examples of which are:

```
namespace foo { /*…*/ }
namespace bar = foo ;
```

2.3.1 Context Sensitivity

With the *module-declaration* moved to the start of source, it is worth considering whether the ‘`module`’ could be a context-sensitive keyword. Introducing new keywords is always a danger to existing code bases, and it is wise to minimize them.

Unfortunately ‘`module`’ is also used in a *proclaiming-ownership-declaration*, which presents difficulties with context sensitivity. I do not explore the issue further in this paper, refer to p0788 for *proclaiming-ownership-declaration* discussion.

3 Changes to Modules-TS Draft

The two options lead to two similar sets of changes to the TS Draft. For avoidance of doubt, I recommend the ‘explicit global module’ approach of 3.2.

Note that p0788 also modifies the grammar in [basic.link,6.5]/1 and covers making `module` context-sensitive. Refer to that paper for the combined changes.

3.1 Direct Combination

To accept the '`export module foo using {…}`' syntax, change the grammar added to [basic.link,6.5]/1 as follows:

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

toplevel-declaration-seq:
~~*toplevel-declaration*~~
toplevel-declaration-seq$_{opt}$ ~~*toplevel-*~~*declaration*
toplevel-declaration-seq$_{opt}$ ***proclaimed-ownership-declaration***

~~*toplevel-declaration*~~
~~*module-declaration*~~
~~*proclaimed-ownership-declaration*~~
~~*declaration*~~

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

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

proclaimed-ownership-declaration:
`extern` `module` *module-name* `:` *declaration*

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

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

~~*module-name-qualifier*~~
~~*identifier*~~

Modify [dcl.module,10.7] as follows:

1 A *module unit* is a translation unit that contains a *module-declaration*. A *named module* is the collection of module units with the same *module-name*. ~~A translation unit may not contain more than one *module-declaration*.~~ A *module-name* has external linkage but cannot be found by name lookup.

3 A *module unit purview* starts ~~a~~**immediately after** the *module-declaration* and extends to the end of the translation unit. The *purview* of a named module `M` is the set of module unit purviews of `M`'s module units.

4 A *namespace-scope* declaration `D` of an entity ~~(other than a module)~~[1] in the purview of a module `M` is said to be owned by `M`. Equivalently, the module `M` is the owning module of `D`.

5 The *global module* is the collection of all declarations **in *global-module* fragments** ~~not in the purview of any *module-declaration*~~. By extension, such declarations are said to be in the purview of the global module. [*Note:* The global module has no name~~,~~ **and** no module interface unit~~, and is not introduced by any *module-declaration*~~. — *end note*]

3.2 Explicit Global Module

To accept the '`export module foo; module {…}`' syntax, change the grammar added to [basic.link,6.5]/1 as follows:

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

toplevel-declaration-seq:
~~*toplevel-declaration*~~
toplevel-declaration-seq$_{opt}$ ~~*toplevel-declaration*~~
toplevel-declaration-seq$_{opt}$ ***proclaimed-ownership-declaration***

~~*toplevel-declaration*~~
~~*module-declaration*~~
~~*proclaimed-ownership-declaration*~~
~~*declaration*~~

***module-preamble*:**
module-declaration global-module-declaration$_{opt}$

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

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

proclaimed-ownership-declaration:
`extern` *module module-name* `:` *declaration*

1 Also noted in p0775r0, this appears to be superfluous wording, regardless of new restrictions on the *module-declaration* location.

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

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

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

Modify [dcl.module,10.7] as follows:

1 A *module unit* is a translation unit that contains a *module-declaration*. A *named module* is the collection of module units with the same *module-name*. ~~A translation unit may not contain more than one *module-declaration*.~~ A *module-name* has external linkage but cannot be found by name lookup.

3 A *module unit purview* starts ~~a~~**immediately after** the *module-*~~*declaration*~~***preamble*** and extends to the end of the translation unit. The *purview* of a named module `M` is the set of module unit purviews of `M`'s module units.

5 The *global module* is the collection of all declarations **in *global-module-declarations*** ~~not in the purview of any *module-declaration*~~. By extension, such declarations are said to be in the purview of the global module. [*Note:* The global module has no name~~,~~ **and** no module interface unit~~, and is not introduced by any *module-declaration*~~. — *end note*]

3.3 Common Changes

Amend the note in [lex.key,5.11]:

[Note: The ~~export and~~ register keyword~~s are~~**is** unused but ~~are~~**is** reserved for future use. — end note]