Document number:	P0143R1
Date:	2016-02-15
Working group:	CWG
Reply to:	`gdr@microsoft.com`
Revises:	P0143R0

Wording for Modules

Gabriel Dos Reis

Abstract

This documents provides formal wording for a module system for C++. This document is to be read in conjunction with document P0142R0 “A Module System for C++”. The proposed wording are with respect to WG21 Committee Draft (N4567).

1 Changes from previous

1.1 Delta from P0143R0

- Incorporated CWG review feedback from the December 07, 2015 Teleconference Review

1.2 Delta from N4466

- Allow attributes on module declarations and import directives
- Add “proclaimed ownership declaration” of entities not owned by the current module
- Incorporate reviews from Core Working Group at the Fall 2015 meeting in Kona, HI:
  - formal definition of *module unit purview*, *module purview*, *module ownership*.
  - clarify that an export declaration shall always appear in a module purview.
  - new linkage: module linkage
  - allow block-scope extern declarations to refer to previous declarations.
  - add the notion of namespace partition to clarify how existing name lookup rules carry over unchanged to modules.
  - clarify the scopes searched during instantiation of exported templates.

2 New Keywords

Add these two keywords to Table 3 in paragraph 2.11/1:

```
module import
```

3 Modules as Entities

Modify paragraph 3/3 as follows:

> An *entity* is a value, object, reference, function, enumerator, type, class member, bit-field, template, template specialization, namespace, module, parameter pack, or `this`.

Modify paragraph 3/4 as follows:

> A *name* is a use of an *identifier* (2.10), *operator-function-id* (13.5), *literal-operator-id* (13.5.8), *conversion-function-id* (12.3.2), ~~or~~ *template-id* (14.2), or *module-name* (7.7) that denotes an entity or *label* (6.6.4, 6.1).

Add a sixth bullet to paragraph 3/8 as follows:

> – they are *module-name*s composed of the same dotted sequence of *identifier*s.

Append the following phrase to paragraph 3.1/2:

> , or a *module-declaration*, or a *module-import-declaration*, or a *module-export-declaration*, or a *proclaimed-ownership-declaration*. *[Example:*

```
import std.io;               // make names from std.io available
module M;                    // declare module M
export module std.random;    // import and export names from std.random
export struct Point {        // define and export Point
  int x;
  int y;
};
```

> *–end example.]*

3.1 ODR: Owning Module is Part of an Entity’s Identity

Add a seventh bullet to 3.2/6 as follows:

> – if a declaration of `D` that is not a *proclaimed-ownership-declaration* appears in the purview of a module (7.7), all other such declarations shall appear in the purview of the same module and there can be at most one definition of `D` in the owning module.

The purpose of this requirement is to implement module ownership of declarations. Add a new paragraph 3.3.2/13 as follows:

> The point of declaration of a module is immediately after the *module-name* in a *module-declaration*.

3.2 Program and Linkage

Change the definition of *translation-unit* in paragraph 3.5/1 to:

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

toplevel-declaration-seq:
  toplevel-declaration
  toplevel-declaration-seq toplevel-declaration

toplevel-declaration:
  module-declaration
  module-export-declaration
  module-import-declaration
  proclaimed-ownership-declaration
  toplevel-export-group
  export-declaration

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

module-export-declaration:
  `export` *module-declaration*

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

export-declaration:
  `export`$_{opt}$ *declaration*

toplevel-export-group:
  `export` { *toplevel-export-seq* }

toplevel-export-seq:

toplevel-export
toplevel-export-seq toplevel-export

toplevel-export:
module-declaration
declaration

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

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

qmod-seq:
identifier `.`
qmod-seq identifier `.`

3.2.1 Module linkage

Insert a new bullet between first and second bullet of paragraph 3.5/2:

— 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 the same module unit (7.7) or from scopes of other module units part of `M`.

Insert a new paragraph before paragraph 3.5/8

A name declared at namespace scope, that does not have internal linkage by the previous rules, and that is introduced by an non-exported declaration has module linkage. The name of any class member where the enclosing class has name with module linkage also has module linkage.

3.2.2 Block-scope extern declarations

Modify 3.5/6 as follows:

6 The name of a function declared in block scope and the name of a variable declared by a block scope extern declaration have linkage. If there is a visible declaration of an entity with linkage having the same name and type, ignoring entities declared outside the innermost enclosing namespace scope, the block scope declaration declares that same entity and receives the linkage of the previous declaration. If

that entity was exported by an imported module, the program is ill-formed. If there is more than one such matching entity, the program is ill-formed. Otherwise, if no matching entity is found, the block scope entity receives external linkage and is owned by the global module.

4 Lookup Rules Adjusted

From end-user perspective, there are really no new lookup rules to learn. The "old" rules are the "new" rules, with appropriate adjustment in the definition of "namespace" which is now clarified as the collection of "namespace partitions".

Modify paragraph 3.3.6/1 as follows:

1 The declarative region of a *namespace-definition* is its *namespace-body*. Entities declared in a *namespace-body* are said to be members of the namespace, and names introduced by these declarations into the declarative region of the namespace are said to be *member names* of the namespace. A namespace member name has namespace scope. Its potential scope includes its namespace from the name's point of declaration (3.3.2) onwards, including the enclosing namespace's partition (7.3) in every translation unit that declares its owning module or that imports the owning module (7.7) if the name is exported; and for each *using-directive* (7.3.4) that nominates the member's namespace, the member's potential scope includes that portion of the potential scope of the *using-directive* that follows the members point of declaration ; and for each *module-import-declaration* that nominates a module that exports that name, the member's potential scope includes the portion of the potential scope of the *using-directive* that follows that *module-import-declaration*. *[Example:*

```
// m−1.ixx
module M;
export int sq(int i) { return i*i; }

// m−2.cxx
import M;
int main() { return sq(9); }  // OK: 'sq' from module M
```

–end example.]

Add to paragraph 3.4/2 a follows:

2 A name looked up in the context of an expression is looked up as an unqualified name in the scope where the expression is found. For a namespace scope, that includes any corresponding namespace partition (7.3) added by *module-import-declaration*s.

5 Exported Functions

5.1 Constexpr and inline functions

Add a new paragraph 7.1.2/7 as follows:

> An exported inline function shall be defined in the same translation unit containing its export declaration. An exported inline function has the same address in each translation unit importing its owning module. *[Note: There is no restriction on the linkage (or absence thereof) of entities that the function body of an exported inline function can reference. A constexpr function is implicitly inline –end note.]*

6 Namespace-scope exported declarations

Modify the grammar in paragraph 7.3.1/1 as follows

namespace-body:
  decl-fragment-seq

decl-fragment-seq:
  fragment
  decl-fragment-seq fragment

fragment:
  export-declaration
  `export` { *declaration-seq* }

7 Namespace partition

Modify paragraph 7.3/1 as follows:

1 A namespace is an optionally-named declarative region. The name of a namespace can be used to access entities declared in that namespace; that is, the members of the namespace. Unlike other declarative regions, the definition of a namespace can be split over several parts of one or more translation units. A *namespace partition* is the collection of all the *namespace-definitions* of the same namespace in a translation unit. A namespace consists of all its namespace partitions. A namespace with external linkage is always exported regardless of whether any of its *namespace-definition* is introduced by `export`. *[Note: There is no way to define a namespace with module linkage –end note.]* *[Example:*

```
module M;
namespace N { // N has external linkage and is exported
}
```

–end example.]

8 Module Declaration

Add a new section 7.7 titled "**Modules**" as follows:

1 A *translation-unit* shall contain at most one *module-declaration* as a *toplevel-declaration*. A *module unit* is a *translation-unit* that contains a *module-declaration*. Such a translation unit is said to be part of the module designated by the *module-name*. A *module-name* has external linkage.

2 A *module* is a collection of module units, at most one of which contains *export-declarations* or *exported-fragment-groups*. That distinguished module unit is called the *module interface unit*. Any other module unit is called a *module implementation unit*.

3 A *module unit purview* starts at the *module-declaration* and extends to the end of the translation unit. The *purview* of a 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) 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 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 is not introduced by any module-declaration. –end note.]*

Add a new subsection 7.7.1 titled "**Export declaration**":

1 An *export-declaration* shall appear in the purview of a module other than the global module. The *interface* of a module `M` is the set of all *export-declarations* in its purview. An *export-declaration* shall declare at least one entity. The names of all entities in the interface of a module are visible to any translation unit importing that module. All entities with linkage other than internal linkage declared in a module interface unit of a module `M` are visible to all module units of `M`. The entity and the declaration introduced by an *export-declaration* are said to be *exported*

2 The name introduced by the declaration of an *export-declaration* shall have external linkage. If that declaration introduces an entity with a non-dependent type, then that type shall have external linkage or shall involve only types with external linkage. *[Example:*

```
module M;
export static int n = 43; // error: n has internal linkage
namespace {
  struct S { };
}
export void f(S); // error: parameter type has internal linkage
```

–end example.]

3 In a *exported-fragment-group*, each *fragment* is processed as an exported declaration.

4 If an *export-declaration* introduces a *namespace-definition*, then each member of the corresponding *namespace-body* is implicitly exported and subject to the rules of export declarations.

Add a new subsection 7.7.2 titled "**Import declaration**":

1 An *import-declaration* adds the namespace partitions with external linkage from the interface of the nominated module to the list of namespace partitions of the current translation unit, thereby making visible to name lookup the declarations in the interface of the nominated module. *[Note: The entities are not redeclared in the translation unit containing the import-declaration. –end note.]*

Add a new subsection 7.7.3 titled "**Module exportation**":

1 A *module-export-declaration* nominating a module `M'` in the purview of a module `M` makes all exported names of `M'` visible to any translation unit importing `M`. *[Note: A module interface unit (for a module `M`) containing an import-declaration does not make the imported names transitively visible to translation units importing the module `M`. –end note.]*

Add a new section 7.7.4 titled "**Proclaimed ownership declaration**":

1 A *proclaimed-ownership-declaration* asserts that the entities introduced by the declaration are exported by the nominated module. It shall not be a defining declaration.

2 A program is ill-formed (no diagnoatic required) if the owning module in the *proclaimed-ownership-declaration* does not export the entities introduced by the declaration.

9 Templates

9.1 Ownership of specializations

Add a new paragraph to 14.7:

7 If the template argument list of the specialization of an exported template involves a non-exported entity, then the resulting specialization has module linkage and is owned by the module that contains the point of instantiation.

8 If all entities involved in the template-argument list of the specialization of an exported template are exported, then the resulting specialization has external linkage and is owned by the owning module of the template.

9 If the declaration of an explicit specialization or of a partial specialization is exported then the declaration of the primary template shall be exported.

Modify second bullet of paragraph 14.6.4/1

— Declarations from namespace partitions associated with the types of the function arguments both from the instantiation context (14.6.4.1) and from the definition context.