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 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 that denotes an entity or label (6.6.4, 6.1).

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

– they are module-names composed of the same character sequence.

Append the following phrase to paragraph 3.1/2:

, or a module-declaration, or an module-importation, or a module-exportation, 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 {
    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 appeared in the purview of a module, 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 keyword module in a module-declaration.
3.2 Program and Linkage

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

\[
\text{translation-unit:} \\
\quad \text{toplevel-declaration-seq}_{\text{opt}}
\]

\[
\text{toplevel-declaration-seq:} \\
\quad \text{toplevel-declaration} \\
\quad \text{toplevel-declaration-seq} \ \text{toplevel-declaration}
\]

\[
\text{toplevel-declaration:} \\
\quad \text{module-declaration} \\
\quad \text{module-exportation} \\
\quad \text{module-importation} \\
\quad \text{export-declaration} \\
\quad \text{exported-fragment-group} \\
\quad \text{proclaimed-ownership-declaration}
\]

\[
\text{module-declaration:} \\
\quad \text{module} \ \text{module-name} \ \text{attribute-specifier-seq}_{\text{opt}} ;
\]

\[
\text{module-exportation:} \\
\quad \text{export} \ \text{module-declaration}
\]

\[
\text{module-importation:} \\
\quad \text{import} \ \text{module-name} \ \text{attribute-specifier-seq}_{\text{opt}} ;
\]

\[
\text{export-declaration:} \\
\quad \text{export}_{\text{opt}} \ \text{declaration}
\]

\[
\text{exported-fragment-group:} \\
\quad \text{export} \ \{ \ \text{fragment-seq} \ \}
\]

\[
\text{proclaimed-ownership-declaration:} \\
\quad \text{extern} \ \text{module} \ \text{module-name} : \ \text{declaration}
\]

\[
\text{module-name:} \\
\quad \text{identifier} \\
\quad \text{module-name} \ . \ \text{identifier}
\]
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 or from scopes of other module units parts of M.

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 is 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 names point of declaration (3.3.2) onwards; and for each using-directive (7.3.4) that nominates the members namespace, the members potential scope includes that portion of the potential scope of the using-directive that follows the members point of declaration; and for each module-importation that nominates a module that exports that name, the members potential scope includes the portion of the potential scope of the using-directive that follows that module-importation. [Example:

```
// m-l.ixx
module M;
export int sq(int i) { return i*i; }
```
\[ \text{m-2.cxx} \]
\begin{verbatim}
\import M;
int main() { return sq(9); } \quad \text{// OK: 'sq' from module M}
\end{verbatim}

\textit{--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 added by \textit{module-importations}.

\section{Exported Functions}

\subsection{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. [\textit{Note: There is no restriction on the linkage (or absence there-of) of entities that the function body of an exported inline function can reference. --end note.}]

Add a new paragraph 7.1.5/10 as follows:

An exported constexpr function shall be defined in the same translation unit containing its export declaration.

\section{Namespace-scope exported declarations}

Modify the grammar in paragraph 7.3.1/1 as follows

\begin{verbatim}
namespace-body:
  fragment-seq

fragment-seq:
  fragment
  fragment-seq fragment

fragment:
  exported-declaration
  export-fragment-group
\end{verbatim}
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. Therefore, 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 explicitly exported. [Note: There is no way to define a namespace with module linkage –end note.]

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. A module unit shall contain exactly one module-declaration. Such translation unit is said to be part of the module designated by the module-name.

2 A module is a collection of module units, at most one 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 The declarative region (in a module unit) starting from a module-declaration and extending to the end of that translation unit is called a module unit purview. The purview of a module \( M \) is the set of module unit purviews of \( M \)'s module units.

4 A declaration \( D \) of an entity (other than a module) is said to be in the purview of a module \( M \) if that declaration appears in the purview of \( M \). The module \( M \) is said to be the owning module of \( D \); or equivalently the declaration \( D \) is said to be owned by \( M \).

5 The global module is the collection of all declarations not in the purview of any named module. 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 named 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 $M$.

2 The name introduced by an export-declaration shall have an external linkage. If that declaration introduces an entity with a non-dependent type, then that type shall have an external linkage.

3 In a exported-fragment-group, each fragment is processed as if it was a declaration lexically preceded by the keyword export.

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. Only non-namespace members are owned by modules.

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 Normally, 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 $M$. A module-exportation nominating a module $M'$ in the purview of a module $M$ makes all exported names of $M'$ visible to any translation unit importing $M$.

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 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:
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 requesting the specialization.

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 global module. Such a specialization is said to be exported.

Modify second bullet of paragraph 14.6.4/1

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