Wording for Modules

Gabriel Dos Reis

Abstract

This document provides formal wording for a module system for C++. This document is to be read in conjunction with document N4465 “A Module C++ for C++”.

1 New Keywords

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

module import

2 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-name composed of the same character sequence.

Append the following phrase to paragraph 3.1/2:

...or a module-declaration, or an import-declaration, or a module-exportation.
2.1 ODR: Owning Module is Part of an Entity’s Identity

Add a seventh bullet to 3.2/6 as follows:

– each definition of D shall appear in the purview of the same 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.

2.2 Program and Linkage

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

translation-unit:
toplevel-declaration-seqopt
toplevel-declaration:
module-exportation
module-importation
export-declaration
exported-fragment-group
fragment

module-declaration:
module module-name ;

module-exportation:
export module-declaration

module-importation:
import module-name ;

export-declaration:
export declaration

exported-fragment-group:
export { fragment-seq }

fragment:
module-declaration

declaration

top-level-declaration

module-name: identifier
module-name . identifier

3 Exported Functions

3.1 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.

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.

4 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 top-level-declaration. A module unit is a translation-unit that contains 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 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 a module unit, and after the module-declaration designating \( M \). The module \( M \) is said to be the owning module of \( D \).

4 A module-declaration establishes the ownership of the module designated by the module-name over all namespace-scope declarations that follow the module-declaration.

5 The global module is the collection of all declarations not in the purview of any named module.
Add a new subsection 7.7.1 titled “Export declaration”:

1 The interface of a module $\mathcal{M}$ is the set of all export-declarations under the purview of $\mathcal{M}$. 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.

2 The name introduced by an export-declaration shall have an external linkage. If that declaration introduces an entity with a type, then that type shall have an external linkage. If the export-declaration introduces a function template or a variable template then the type of the corresponding current instantiation shall contain only types with external linkage. If the export-declaration introduces a template alias then the aliased type shall have external linkage. If the export-declaration defines a class template, then all non-internal members of the corresponding current instantiation shall contain only types with 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 makes visible the names of all entities in the interface of the nominated module. The semantics of those entities are as if the module interface containing their declaration has been processed from translation phase 1 through 7. [Note: The entities are not redeclared in the translation unit containing the import-declaration. Only their names are made visible. –end note.]

Add a new subsection 7.7.3 titled “Module exportation”:

1 Normally, a module interface unit (for a module $\mathcal{M}$) containing an import-declaration does not make the imported names transitively visible to translation units importing the module $\mathcal{M}$. A module-exportation nominating a module $\mathcal{M}'$ in the purview of a module $\mathcal{M}$ makes all exported names of $\mathcal{M}'$ visible to any translation unit importing $\mathcal{M}$.

5 Templates

TBD.