P0501R2 2017-06-19 Reply-To: gdr@microsoft.com

C++ Module TS Issues List

Gabriel Dos Reis
Microsoft

Active Issues

[5] Static local variables [John Spicer, 11/8/2016]

Question

Should there be a restriction on local statics in exported function template?

Proposed Resolution

The current design purposefully does **not** place any restriction. It even has an explicit note to that effect for inline functions. Maybe that note should be clarified that it also applies to function templates.

[7] Default arguments for exported declarations [John Spicer, 11/8/2016]

Question

Consider

```
      // interface unit of M
      module M;
      export namespace N {
            int f(int);
      }
      namespace N {
            int f(int = 5);
      }

      // 1.cxx, not part of M
      import M;
      int main() { return N::f(); } // OK?
```

Proposed Resolution

Only default arguments in exported declarations are effectively exported, i.e. visible to importing translation units. This issue is resolved by the new wording for issue #4. Add a note to the specification.

[9] Point of definition of implicitly defined special member functions [Roger Orr, 11/7/2016]

Question

What is the point of definition of delayed implicitly defined special member functions?

Proposed Resolution

When the definition of an implicitly defined special member function is needed, the context of the definition shall be right after the last exported declaration from the owning module's interface unit.

[10] Annotation of module declaration in module interface unit [Nathan Sidwell, 2/2/2017]

See discussion on the 'modules' reflector with title 'modules'. Request:

> *It would be nice if the module declaration was decorated in some unique way to denote the interface unit.*

Proposed Resolution

From specification perspective, there is no ambiguity about which translation unit is a module interface unit. It might however be convenient for some to see a redundant annotation in the source code indicating that a source file is indeed a module interface unit. See proposal P0273R1, proposal P0584R0 titled "*Module Interface and Preamble*", and proposal P0629.

[11] Redeclaration within the purview of a module [Nathan Sidwell 1/9/2017]

See discussion titled 'modules' on the 'modules' reflector.

Question

In the purview of a module, can a redeclaration export an entity that wasn't previously declared as exported?

Proposed Resolution

No. This was expressly forbidden, and that restriction is encoded via the forbidden change of linkage (from module linkage to external linkage).
Add a note in the specification to illustrate this.

[12] Exported partial specialization [Nathan Sidwell, 1/30/2017]

See the discussion 'export and templates' on the 'modules' reflector.

Question

Given an exported class template declaration X, is it possible to declare two non-exported partial specialization of X in two modules?

Proposed Resolution

Yes; however, the partial specialization shall depend on non-exported types or templates.

[13] Module Partitions [EWG, 3/4/2017]

This issue results from a EWG straw poll at the Spring 2017 Kona meeting.

Question

Provide a notation for expressing "module partitions", see proposal P0273R1 for background. See proposal P0584R0.

[14] Uniqueness of namespace names across modules [Nathan Sidwell, 4/14/2017]

Report

> *From the TS I see that namespaces are always exported. But can an unrelated module bind the same name to something other than a namespace? For instance*

/// file A
module A [[interface]];
namespace N { export void Foo (); };

// file B
module B [[interface]];
export void N ();

// file C
module C [[interface]];
import A;
export inline void Frob () {N::Foo ();}

// file D
import C;
import B;

Is this well formed? If it isn't at what point is a diagnostic required? This case looks like it might be an ODR violation because both 'namespace N' and 'void N ()' are exported. But what if B didn't export 'void N ()'?.

More philosophically, I'd conceptualized namespaces as a program-wide object, of which a particular module might have an empty partition. But, I'm not sure that's justifiable from the TS. An empty partition is not necessarily the same as a null partition. Does the latter permits a module to bind the identifier to something else?

Discussion

[15] Semantics of exported using-directives [Nathan Sidwell, 6/19/2017]

Question

What is the semantics of exported using-directive?

[16] Semantics of exported using-declarations [Nathan Sidwell, 6/19/2017]

Question

What is the semantics of exported using-declaration?

Closed Issues

[1] export import M; [Richard Smith, 9/7/2016]

Remove from grammar. Or ban it through semantics prose. Same thing with export { import M; }

Resolution
See P0500R0 adopted at the Fall 2016 Issaquah meeting.

[2] import M; at interface level [Richard Smith, 9/7/2016]
Ban it from interface units.

Resolution
See P0500R0 adopted at the Fall 2016 Issaquah meeting.

[3] export const int n = 5; [Richard Smith, 9/8/2016]
Clarify that this is allowed.

Resolution
See P0500R0 adopted at the Fall 2016 Issaquah meeting.

[4] Import declaration and namespace partitions; [Lukasz Mendakiewicz, 11/3/2016]
Problem:
I was reading N4610 and have a question:

```
module M;
export namespace N
{
  struct A {};
}
namespace N
{
  struct B {};
}
```

7.7.1/4 says that all members of **namespace-body** are exported, meaning N::A.

```
import M;
```

7.7.2/1 says that import declaration adds the **namespace partitions** with external linkage from M to the current TU.
Namespace partition N from M contains both N::A and N::B.

So is N::B visible and can be used in the second TU or not?

Resolution
See P0500R0 adopted at the Fall 2016 Issaquah meeting.

[6] Entities referenced from exported templates [John Spicer, 11/8/2016]

Question

Are there any restrictions on the linkage of the entities that can be referenced from an exported template definition?

Resolution

For function templates, by design, **there is no restriction on the linkage of entities that can be referenced from a definition**. Note, this is the same restriction as for inline functions.

[8] Point of instantiation across translation units and name lookup [John Spicer, 11/8/2016]

See paper P0582R0 titled "*Modules: Contexts of Template Instantiations and Name Lookup*", adopted at the Spring 2017 Kona meeting.