C++ Module TS Issues List
Gabriel Dos Reis
Microsoft

1. export import M; [Richard Smith, 9/7/2016]
Remove from grammar. Or ban it through semantics prose. Same thing with export { import M; }
Proposed 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.
Proposed 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.
Proposed 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:

```c++
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.

```c++
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?

**Proposed Resolution**

See [P0500R0](#) adopted at the Fall 2016 Issaquah meeting.

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 to apply also for templated functions.

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?

**Proposed 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.

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

**Question**

Consider

```cpp
// 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.
8. Point of instantiation across name lookup and name lookup [John Spicer, 11/8/2016]
See paper P0582R0 titled "Modules: Contexts of template instantiations and name lookup".

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 and proposal P0584R0 titled “Module Interface and Preamble”.

11. Redefinition 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 redefinition an 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.