Document Number: P0721
Date: 2017-06-20
To: SC22/WG21 CWG/EWG
Reply to: Nathan Sidwell
nathan@acm.org
Re: Working Draft, Extensions to C ++ for Modules, n4647

Exporting Using Declarations

Nathan Sidwell

This discusses how export, import and using declarations interact.

1 Background

The Modules proposal is not entirely clear what the semantics of exported using declarations are. Module imports are non-transitive, but how does that interact with using declarations?

2 Example

Consider:

```
// Foo interface TU
export module Foo;
namespace bill {
  export void X (int);
};
```

```
// Bar interface TU
export module Bar;
import Foo; // non-rexport
export using bill::X;
```

```
// User 1 TU
import Bar;
… X … // #1 What is found?
import Foo;
… X … // #2 What is found?
```

```
// User 2 TU
import Foo;
import Bar;
… X … // #3 What is found?
```

3 Discussion

Currently, the set of declarations that a namespace-scope using declaration imports into the current namespace is determined at the time the using declaration is parsed. If functions are later added to the overload set of the nominated namespace, these do not automatically become visible in the using declaration's context. How does importing and exporting affect that?

The above example shows a `Foo` module exporting `bill::X` and a `Bar` module exporting a using declaration for `bill::X`. At parse time, that lookup will find the first module's declaration `bill::X`. However, because `Bar` does not export `Foo`, it is not clear what a user of `Bar` will observe. Is it dependent on whether the user has already imported `Foo` or not? If the user later imports `Foo`, does the answer change?

It seems undesirable for the behaviour of a using declaration in one module to depend on the local context of an importing module. Particularly as the importing module might or might not have (indirectly) imported the modules exporting the ultimately nominated declaration(s). This suggests that using declarations see through the imports of the module exporting the using declarations. Declarations introduced by such a using declaration would therefore break the non-transitivity of imports.

Alternatively, perhaps the using declaration is ill-formed, because it does not nominate any declarations that are (re)exported from the current module? If that were the case, what if it nominates some reexported declaration, and some other declarations? Are the non-reexported ones culled from the set of declarations that importers of the module see?

4 Conclusion

The modules proposal should clarify that the intended semantics for exported using declarations.