Document Number: P0925r0
Date: 2018-02-08
To: SC22/WG21 EWG
Reply to: Nathan Sidwell
nathan@acm.org / nathans@fb.com
Re: Working Draft, Extensions to C ++ for Modules, n4720

Modules:Unqualified Using Declarations

Nathan Sidwell

Using declarations are grammatically restricted to employing a qualified name. The modules-ts presses them into service for exporting entities declared in the global module. Allowing an unqualified name may be clearer.

1 Background

A pre-existing use of a using declaration is to bring a set of entities into scope from a foreign namespace. As such using an unqualified name makes little sense, and a qualified name is required.

The modules-ts presents another use for a using declaration – exporting a set of entities from the global module:

```
namespace frobbing {
 void frob (); // external-linkage, not exported
}
export module frobber;
namespace frobbing {
 export using frobbing::frob; // export frobbing::frob
}
```

The requirement that a qualified name is required may be confusing in this case, and lead to inadvertent error. Pressing using-declarations into service as the mechanism by which global module external-linkage entities may be retroactively exported conflates two behaviours:

- The exporting behaviour we desire
- The making visible of an entity from a foreign namespace

1.1 Possible Errors

That a qualified name is required, permits the exporting declaration to change the namespace in which a global module entity is made reachable:

```
namespace frobbing {
 void frob ();
}
export module frobber;
// neglected to open namespace frobbing
export using frobbing::frob;
// ::frob now available in imports
```

This may well be an unintended change of namespace.

Modifying the example slightly to:

```
namespace frobbing {
 void frob (); // #1
}
export module frobber;
void frob (int); // #2, module linkage
export using frobbing::frob;
// ::frob now available in imports, sees #1
```

Creates a situation where the names found by the using declaration are now exported, but that is not the complete set of bindings for '`::frob`'. Indeed, it would now be an error to append the following:

```
export using ::frob;
```

in the above example, as that would find #2 in the overload set, which is not exportable (it has module linkage).

The programmer would need to examine each '`export using $namespace::$name;`' to determine whether `$namespace` is the current namespace or not.

2 Proposal

The proposal is that an unqualified name could be employed in a *using-declaration*, in at least some circumstances.

Allowing an unqualified name has an interesting corner case when the name is not already bound in the current namespace. The lookup rules are sufficiently different that using plain '`frob`', rather than '`current_namespace::frob`' could find different sets of entities.

Unqualified name lookup proceeds towards ‘::’, until a binding is discovered. Any using directives are followed when the deepest common ancestor namespace of the using directive’s source and target namespaces is encountered.

Qualified name lookup follows using directives in the nominated namespace. This search propagates in a wave-front manner, only stopping along a particular path if a binding is found, there is no using directive to follow, or a loop is discovered. This is a flood-fill algorithm.

An example showing the difference is:

```
// qualified name search
namespace bar {
  void frob ();
}
namespace outer {
  namespace frobbing {
    using namespace bar;
  }
  void frob ();
}
export module frobber;
namespace outer::frobbing {
  export using frobbing::frob; // finds bar::frob
}

// unqualified name search
namespace bar {
  void frob ();
}
namespace outer {
  namespace frobbing {
    using namespace bar;
  }
  void frob ();
}
export module frobber;
namespace outer::frobbing {
  export using frob; // would find outer::frob
}
```

That these behave differently is probably going to be confusing. We can avoid this by requiring that when an unqualified name is used, it must find a binding in the current namespace. (Such a binding might contain entities brought in by previous using declarations.)

A well-formed non-exported unqualified using declaration, not in the purview of a module interface unit, would have no effect. In the purview of a module interface, it might make declarations from the global module visible to implementation units.

Because a qualified using declaration can make names visible in new namespace, it may be it should be ill-formed to export such declarations. Thus exporting a name cannot make more names visible in the current namespace – it only changes the exportedness of names already visible.

The requirement that all names found by an exported using directive have no linkage or external linkage would be unaffected by the form of the name.

2.1 Summary

The changes to the modules-ts draft are:

- *Using-declarations* may take an unqualified name
- Such an unqualified *using-declaration* must find a binding in the current namespace
- Only unqualified *using-declarations* may be exported

3 Changes to Modules-TS Draft

Modify [namespace.udecl]:

- Allow *unqualified-id* as a *using-declarator*
- Restrict such uses to namespace-scope *using-declarations*.
- Require that lookup of the *unqualified-id* find a binding in the current namespace.
- Require that an exported *using-declaration* use an *unqualified-id*.

Precise wording to be suggested later.