Create namespaces for modules without wrapping files in namespace blocks

Document #:	PXXXXRX
Date:	2022-09-15
Project:	Programming Language C++
Audience:	Core Evolution Working Group

Reply-to:	Maximilian Downey Twiss <creatorsmithmdt@gmail.com>

1 Introduction

This paper proposes to allow the creation of namespaces for modules without wrapping files in namespace-blocks through specifying a namespace-name in a module-declaration.

2 Motivation and Scope

In brief, this paper hinges on the idea that wrapping implementation files in namespace blocks is bad, as they add unnecessary (albeit optional) indentation, decrease clarity when not indented and add unnecessarily lines of code.

While the effects of this can partially be mitigated by not indenting the namespace blocks and using an ending block comment, it does not solve the root issue, which gets worse the larger the file is and the more interface partitions are used, as they also have to be wrapped in namespace blocks.

While headers allow one to relatively easily use namespaces without wrapping implementation files in namespace blocks, modules have no such equivalent.

Using headers, one can wrapping the header files in namespace blocks and prepending the namespace onto the functions in the implementation file, although this is against good style.

For example:

old.cpp

```
#include "old.h"

void old::bar()
{
  int bar;
}

void old::foo()
{
  int foo;
}
```

old.h

```
namespace old
{
public:
  void bar();
```

```
private:
  void foo();
};
```

However, when using modules, there is currently no way, regardless of style, to avoid wrapping implementation files in namespace blocks.

See example: current.cpp

```
export module current;

namespace current
{
export void bar()
{
  int bar;
}

void foo()
{
  int foo;
}
} //namespace current
```

Through this change, this paper creates a simple, compact way to use namespaces with modules, as opposed to wrapping implementation files inside namespace blocks.

Like so: new.cpp

```
export module new namespace new;

export void bar()
{
  int bar;
}

void foo()
{
  int foo;
}
```

For clarification, current.cpp and new.cpp have identical behaviour.

While the current examples may seem to be relatively similar, the differences between the current method and new method become significantly more apparent in larger files, and with multiple interface partitions. Examples reflecting this were not included for brevity.

3 Wording

Relative to N4910.

3.1 #[module]

Change the grammar before paragraph 1:

```
  module-declaration:
    export-keyword(opt) module-keyword module-name module-partition(opt)
```

```
-    attribute-specifier-seq(opt);
+    namespace-name(opt) attribute-specifier-seq(opt);
  module-name:
    module-name-qualifier(opt) identifier
```

(The re-adjusting of numbers after the introduction of a new paragraph was not included for diff conciseness, but is implied) Insert new paragraph after 3:

```
+ A module-declaration with a namespace-name implies a namespace-definition where the
+ identifier is the namespace-name and there is a namespace-body enclosing all names
+ in the translation unit.
```