Document Number: P0867r0
Date: 2017-11-21
To: SC22/WG21 EWG
Reply to: Nathan Sidwell
nathan@acm.org / nathans@fb.com
Re: Working Draft, Extensions to C ++ for Modules, n4681

'Module Interface' is Misleading

Nathan Sidwell

The modules TS defines ‘module interface unit’ and ‘module implementation unit’. Unfortunately ‘interface’ and ‘implementation’ come with existing baggage that can mislead new users.

1 Background

The modules TS separates module translation units into two categories:

- Module *interface* unit.
- Module *implementation* units.

The module interface unit is denoted by a module-declaration containing the ‘`export`’ keyword. The implementation units’ module-declarations lack that keyword. The interface unit may:

- Declare exported entities. These are entities that are visible to importers of the module.
- Declare module-linkage entities that are visible within module implementation units.
- Define exported and module-linkage entities.

This last bullet, when applied to non-inline non-template functions and variables is exactly the functionality of an implementation unit – the interface unit is itself an implementation unit.

While the modules-ts is careful to define these semantics, the words ‘interface’ & ‘implementation’ come loaded with presumed connotations. For instance, we are used to header files declaring an interface to a library and other source files containing implementation. Likewise, ‘interface class’ is a common concept of a class defining only abstract virtual member functions – defining no data members or functions. I have had conversations and talks explaining that a module interface may contain definitions and the other parties express surprise.

I suspect this confusion leads to simple module deployments consisting of two translation units – an interface containing only header-like entities and a separate implementation containing function definitions. A single translation unit may be more appropriate.

I believe this confusion is further compounded by compiler implementations that use a distinct source file suffix for module interface units. Such a scheme permits an interface unit and the single implementation unit to have the same basename. (Of course confusion then arises over naming the corresponding object files.)

2 Proposal

Using two unrelated names, whatever they may be, to distinguish the two cases could continue to mislead a user into thinking that the (now-called) implementation unit has features that the (now-called) interface unit does not.

Therefore I prefer names of the form ‘flanging module unit’ and ‘non-flanging module unit’, for some value of ‘flange’. I think this clarifies that one of these things has additional features the other lacks.

Given that the (now-called) interface unit is the translation unit that exports entities, ‘exporting’ seems a suitable distinguishing feature.

I propose using ‘exporting module unit’ and ‘non-exporting module unit’ to distinguish the two cases.

3 Changes to Modules-TS Draft

In brief, any mention of ‘interface unit’ would become ‘exporting unit’ and any mention of ‘implementation unit’ would become ‘non-exporting unit’. As non-module units also cannot export entities, it may be necessary to insert ‘module’, to clarify that non-exporting is a feature of a module unit.

Full details to be considered later.