Document number:P0795R0
Date: 20171016 (pre-Albuquerque)
Project: Programming Language C++, WG21, SG14, EWG
Authors: Simon Brand, Neil Henning, Michael Wong, Christopher Di Bella, Kenneth Benzie
Email: simon@codeplay.com, neil@codeplay.com, michael@codeplay.com
Reply to: michael@codeplay.com

From Vulkan with love: a plea to reconsider the Module Keyword to be contextual

The problem

This paper is a late comment to the Module PDTS ballot.

N4681 proposes adding `module` as a non-contextual keyword. This will create incompatibilities with C++ code bases which already use `module` as an identifier. A simple text replacement may suffice for many projects, but for widely-used standards or libraries, this could have a big knock-on effect and may cause those projects to not adopt a new version of the standard.

This is because the Module keyword is too common and already in common use in many codebases through a simple search of Github, Google codebases. Any hundreds of use of Module has already been in existence as can be seen in the Alternatives Considered section, where if the C++ keyword is not made at least contextual, then all these codebases will potentially have to change.

Indeed one of the biggest concern is from a friendly Consortium that shepherds the graphics/games/low-latency community of SG14, the Khronos group.

From Khronos, which hosts many of the graphics, and heterogeneous programming specifications including OpenCL and SYCL, it includes one of the latest graphics specifications known as Vulkan, the replacement to OpenGL which is one of the hottest graphics interface for Games (in addition to Directx). This defect report [https://github.com/KhronosGroup/Vulkan-Docs/issues/568] identifies a name collision with Modules keyword when used with VS 2015.

Michael Wong chairs the SYCL Heterogeneous C++ language group within Khronos, and was asked by representatives from Nvidia, Qualcomm, Imagination, Adobe(indeed a continuous stream of people in the Khronos meeting connected with him) as well as a number of graphics company to represent this view point. At the time of the Khronos meeting, he had just also concurrently submitted the Canadian comments and ballot based on his own internal caucus (which disapproved Module for its own reason), but before the Khronos Vulkan concerned was raised, hence this late comment paper. However, he did make the editor of Module aware of both the Canadian position, and the Vulkan concern. For the record, all the authors here like the Modules TS and would like it to proceed forward to TS and consider this a friendly amendment.

Vulkan provides a class `VkPipelineShaderStageCreateInfo` which has a member variable named `module`:

```
#include <vulkan/vulkan.h>

typedef struct VkPipelineShaderStageCreateInfo {
    VkStructureType                     sType;
    const void*                         pNext;
    VkPipelineShaderStageCreateFlags    flags;
    VkShaderStageFlagBits               stage;
    VkShaderModule                      module; ← fails here
    const char*                         pName;
    const VkSpecializationInfo*         pSpecializationInfo;
} VkPipelineShaderStageCreateInfo;

VkPipelineShaderStageCreateInfo pipelineShaderStage;

{
  pipelineShaderStage.module = myModule;  // error
}
```

MSVC 2015 update 3 error message with `/experimental:module`

```
vulkan.h(1691): error C2059: syntax error: 'module'
vulkan.h(1691): error C2238: unexpected token(s) preceding ';'
test.cpp(6): error C2059: syntax error: 'module'
test.cpp(6): error C2039: 'shaderStageCreateInfo': is not a member of
'VkPipelineShaderStageCreateInfo'
vulkan.h(1686): note: see declaration of
'VkPipelineShaderStageCreateInfo'
```

Clang 5.0.0 error message with `-fconcepts-ts`:

```
vulkan.h:2011:41: error: expected member name or ';' after declaration
specifiers
    VkShaderModule                      module;
    ~~~~~~~~~~~~~~                      ^
test.cpp:6:25: error: expected unqualified-id
    pipelineShaderStage.module = myModule;
```

If `module` is kept as a non-contextual keyword then the Vulkan standard would need to be changed if compatibility with C++ with modules is to be supported. Many other codebases would also need to change to remove its use to align with the new C++.

Proposed Solution

`module` could be moved from from Table 5 in [lex.key] to Table 4 in [lex.name]. This would allow it to be used as an identifier and would fix the issue presented above. The use of `module` should be disambiguated as a *module-declaration* when possible, so the following code would declare a module rather than a global variable of type `module`:

```
struct module{};
module i_am_a_module;
```

An alternative would be to make the above construct ill-formed.

Using `module` as a module name would need to be explicitly disallowed:

```
module module; //bad
```

Alternatives Considered

If we do not change it then any code which uses `module` as an identifier will need to be changed for it to compile under the rules in N4681. A short list of popular projects which would require changes follows. It was not ascertained whether or not these changes would affect client code (some are used internally to functions or in tests, for example), but all of these libraries would need to make code changes to compile under the current rules:

- Vulkan
- Tensorflow
- Swift
- Electron
- OpenCV
- V8
- Osquery

- Sqlitebrowser
- Xmbc
- OpenFrameworks
- Paddle
- Cap’n Proto

Acknowledgement