C++ Club Meeting Notes

Gleb Dolgich

2017-11-16

Meeting C++ Trip Report

- Simon Brand

- Gordon Brown

2017-10 Albuquerque ISO C++ Commitee Meeting

- Pre-Albuquerque Mailing
- Reddit Trip Report by Bryce Lelbach
- Trip report by Herb Sutter

Feature macros

P0819R0

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

P0795R0

The `module` keyword is already being used in many code bases as identifier.

Business Requirements for Modules

P0678R0 by John Lakos, Bloomberg

Some are looking, primarily, to reduce protracted build times for template-ladened header files. Others want to use modules as a vehicle to clean up impure vestiges of the language, such as macros, that leak out into client code. Still others are looking to “modernize” the way we view C++ rendering completely – even if it means forking the language. These are all very different motivations, and they may or may not be entirely compatible, but if the agreed-upon implementation of modules does not take into account established code bases, such as Bloomberg’s, they will surely fall far short of wide-spread adoption by industry.

Module TS Wording Does Not Support Intended Use Case

P0832R0 by Bloomberg

Exposing existing code as a module should involve providing a new module interface unit that can reuse existing source files without modification such that current consumers are not broken.

Modules at scale

P0841R0 by Apple

This paper discusses Apple's experience with modules on its software ecosystem and how the current state of the Modules TS lacks the necessary expressiveness to reflect Apple's requirements.

Problems of Modules TS:

- The `module` keyword is not context-sensitive
- A module can only have one module interface unit file (showstopper)
- Macros are used to manage OS feature availability (showstopper, transition needed)

```
1 export module M; // declare module M
2 export module M.__macros; // M export all macros
3 export module M.__macros.INFINITY; // M exports macro INFINITY
```

Clang 6 Documentation: Modules (not Modules TS!)

Clang 6 website

- Problems with header files: compile-time scalability, fragility, conventional workarounds, tool confusion

Problems modules don't solve:

- Rewrite the world's code
- Versioning
- Namespaces
- Binary distribution of modules

Proposed modules changes from implementation and deployment experience, by Google

P0273R0

Does C++ need a universal package manager?

- Article by Paul Fultz II
- Reddit thread
- Hacker News thread
- Many package managers but no standard build system for C++
- A platform-independent package manager for C++ must effectively degrade into a build system that rebuilds all dependencies from source (Rust & Cargo)

  What C++ needs is a common format to communicate a package's requirements among different package manager tools.

CppCon 2017: Boris Kolpackov “Building C++ Modules”

YouTube

- Isolation from macros and symbols
- A physical design mechanism
- A step towards not needing the preprocessor
- Reliable distributed compilation
- Faster builds
- Interesting questions from John Lakos (how to support modules given a huge legacy codebase)

CppCon 2017: Boris Kolpackov “C++ Modules and Packages: Making Dreams Come True”

YouTube (5m)

A live demo of using *build2* package manager and build system with C++ Modules.

Common C++ Modules TS Misconceptions

Post by Boris Kolpackov

- “I cannot have everything in a single file” – you can
- “I cannot export macros from modules” – good
- “I cannot modularize existing code without touching it” – some duplication required
- “No build system will be able to support modules” – *build2*

Reddit

- It would be good if people interested in modules would read the whole proposal.
- Nothing in the current proposal mandates a binary module interface (BMI) or that it has to be a file. Is that a good thing?

CppCon 2017: Isabella Muerte “There Will Be Build Systems: I Configure Your Milkshake”

YouTube

- The C++ ecosystem is really bad (compared to other languages)
- Discusses complexities of build systems
- Mentions meta-build systems (CMake, Meson, Premake)

CppCast

- A great confusion exists on what exactly are C++ Modules, even in the Committee
- Build systems need to understand modules (Gabriel Dos Reyes)
- *build2* is currently the only build system supporting all existing implementations of Modules TS

Why the C++ modules feature is very important for the C++ future?

CppDepend Blog

C++ Modules, what are they for?

Reddit post with a “Modules Lite” proposal

Goals of Modules TS

1. componentization;
2. isolation from macros;
3. scalable build;
4. support for modern semantics-aware developer tools.

Furthermore, the proposal reduces opportunities for violations of the One Definition Rule (ODR), and increases practical type-safe linking.

What would the proposed C++ Module TS solve or help regarding modern C++ coding?

Reddit

CppCon 2017: Nathan Sidwell “Adding C++ modules-ts to the GNU Compiler”

YouTube

s/RAII/DRES/

DRES: Destructors Run on Exit Scope

Proposed by John D. Woolverton (his CppCon 2017 talk).

Quote

Stan Kelly-Bootle:

> *Should array indices start at 0 or 1? My compromise of 0.5 was rejected without, I thought, proper consideration.*

Quote

@jamesshore:

> *Do; or do not. There is no //TODO*