Enable Modules on Windows

Vaibhav Garg
Google Summer of Code' 20

Mentors:
Vassil Vassilev & Bertrand Bellenot

ROOT

- ROOT is a modular scientific software toolkit. It provides the functionalities needed to deal with big data processing, statistical analysis, visualization, and storage.
- It is a **core software used in HEP** not only for data analysis but also as a backend for LHC experiment's software such as CMSSW.

Modules

What are they?

- Software is built using libraries (system + third-party).
- In C, we access the libraries using `#include <SomeLib.h>`
- Modules provide an alternate, simple way to access libraries.

What do they provide?

- Better compile-time scalability.
- Elements problems inherent to using the C preprocessor to access the API of a library.

Why switch?

Problems with #include

- Compile-time scalability.
  - Every time a header is included, it's contents need to be parsed.
  - For example, say, you have a project with M translation units each having N header files in each unit, the compiler needs to perform M x N level of work.
  - This is worse in C++, as support for templates forces a large amount of code in each header.

Why switch?

Problems with #include

- Compile-time scalability.
- Fragility
  - #include directive is treated as textual inclusion.
  - They are, therefore, subject to any active macro definitions at the time of inclusion.
  - If any of the active macro definitions happens to collide with a name in the library, it causes failures.
  - Workarounds possible, ex: Include guards.

Why switch?

Problems with #include

- Compile-time scalability.
- Fragility
- Tool Confusion
  - In a C-based language, it is hard to develop tools, because the boundaries of the libraries are not clear.
  - Which headers belong to a particular library, and in what order should those headers be included?
  - Are the headers C, C++ or Objective-C++?

How do Modules solve these issues?

Semantic Import

```
import std.io ;                    // pseudo-code
```

Modules improve access to the API of software libraries by replacing the textual preprocessor inclusion model with a more robust, more efficient semantic model.

There is only a minor change in the user's perspective, but the import declarations behave quite differently.

How do Modules solve these issues?

Semantic Import

```
import std.io ;                    // pseudo-code
```

When the compiler sees the module import above, it loads a binary representation of the std.io module and makes its API available to the application directly.

So, essentially, we create a binary representation of each header file once, and just load it whenever required.

How do Modules solve these issues?

Benefits

- Each module is compiled only once, so the earlier problem of O(M x N) problem of compilation is reduced to O(M + N).
- Each module is parsed as a standalone entity, so it has a consistent preprocessor environment.
- Modules describe the API of software libraries, therefore, tools can rely on the module definition to ensure that they get the complete API for the library.

C++ Modules in ROOT

- The ROOT v6.16 release came with a preview of the module technology.
- C++ Modules are default in ROOT, starting from v6.20 in UNIX and v6.22 in OS X.
- This project aimed to extend the support of C++ Modules of ROOT to Windows.

Why is Windows different?

- Windows uses MSVC (Microsoft Visual C++), instead of commonly used GCC in Unix based systems.
- MSVC has an absence of C99 support.
- Also, several GCC specific headers are not present in MSVC. Ex: `bits/allocator.h` and likewise.
- MSVC allows many invalid constructs in class templates that Clang has historically rejected.

Major Changes

New Modulemap files for Standard Library of Windows

- Each module needs to have a corresponding modulemap, which describes how a collection of existing headers maps on to the (logical) structure of a module.
- Since the file structure of libraries present in MSVC is different from Unix, there was a need to have different modulemap files for Windows.
- The new modulemap files are now closely related to the actual structure of libraries present in MSVC.

Major Changes

Merge Two Decl's successfully when Inheritable attributes are present. (Clang)

- In Clang, during the formation of AST, there are certain attributes of a Declaration that are needed to be present in the redeclaration chain.
- This was usually done during ASTReading, but was somehow not done in the case of Inheritable Attributes.
- A patch was submitted to Clang (Backported to ROOT) in order to fix this issue.

Major Changes

Correctly parse LateParsedTemplates (Clang)

- In Clang, when template instantiations from dependent modules were read, it was assumed that the Local DeclID, and Global DeclID is consistent.
- However, in practice, it was not the case, which led to DeclID confusion while parsing them.
- A patch was submitted to Clang to fix this issue. (Under review).

Major Changes

Teach DLM to recognise symbols in COFF Object Files

- All the symbol lookup of ROOT is handled by the DynamicLibraryManager.
- On Windows, all the symbols are present inside the COFF Object File Format, which the DLM did not understand previously.

Other Changes

- Several other changes were made in ROOT, which were related to small bugs present in the codebase that were discovered from time to time.
- Also, there were some issues which were related to non-compatibility of Windows and Unix and were appropriately handled.

Results

- We are now able to successfully build ROOT on Windows with C++ modules enabled.
- 65% of the tests are currently passing.
- Some more minor issues are needed to be fixed in order to make Modules default on Windows.

Next steps

Fix issues with the Jenkins build system.

2 build nodes are currently unable to build on Windows with C++ Modules enabled.

Diagnose the failing tests

Diagnose and resolve the remaining 35% tests.

Make C++ Modules default on Windows.

Specific List of Future Work.

Issues with Builds:

- Debug the buildbot failure on lcgapp-win10-65.cern.ch and lcgapp-win10-64.cern.ch.
- Debug the SourceLocation overflow offset issue.

We will then have a working modules build of ROOT on Windows. After this, we need to fix the tests, the major failures are associated with:

- 'definition with same mangled name as another definition' issue.
- 'got exit code Access violation but expected 0' issue.
- ‘No curly at claimed position of opening curly!’ issue.

Questions?