NAME

cmake-cxxmodules − CMake C++ Modules Support Reference

Added in version 3.28.

C++ 20 introduced the concept of "modules" to the language. The design requires build systems to order compilations among each other to satisfy **import** statements reliably. CMake's implementation asks the compiler to scan source files for module dependencies during the build, collates scanning results to infer ordering constraints, and tells the build tool how to dynamically update the build graph.

COMPILATION STRATEGY

With C++ modules, compiling a set of C++ sources is no longer embarrassingly parallel. That is, any given source may first require the compilation of another source file first in order to provide a "CMI" (compiled module interface) or "BMI" (binary module interface) that C++ compilers use to satisfy **import** statements in other sources. With headers, sources could share their declarations so that any consumers could compile independently. With modules, declarations are now generated into these BMI files by the compiler during compilation based on the contents of the source file and its **export** statements.

The order necessary for compilation requires build−time resolution of the ordering because the order is controlled by the contents of the sources. This means that the ordering needs extracted from the source during the build to avoid regenerating the build graph via a configure and generate phase for every source change to get a correct build.

The general strategy is to use a "scanner" to extract the ordering dependency information and update the build graph with new edges between existing edges by taking the per−source scan results (represented by *-P1689R5* files) and "collating" the dependencies within a target and to modules produced by targets visible to the target. The primary task is to generate "module map" files to pass to each compile rule with the paths to the BMIs needed to satisfy **import** statements. The collator also has tasks to use the build−time information to fill out information including **install** rules for the module interface units, their BMIs, and properties for any exported targets with C++ modules.

NOTE:

CMake is focusing on correct builds before looking at performance improvements. There are known tactics within the chosen strategy which may offer build performance improvements. However, they are being deferred until we have a working model against which to compare them. It is also important to note that a tactic useful in one situation (e.g., clean builds) may not be performant in a different situation (e.g., incremental builds). Finding a balance and offering controls to select the tactics is future work.

SCANNING CONTROL

Whether or not sources get scanned for C++ module usage is dependent on the following queries. The first query that provides a yes/no answer is used.

- If the source file belongs to a file set of type **CXX_MODULES**, it will be scanned.
- If the target does not use at least C++ 20, it will not be scanned.
- If the source file is not the language **CXX**, it will not be scanned.
- If the *CXX_SCAN_FOR_MODULES* source file property is set, its value will be used.
- If the *CXX_SCAN_FOR_MODULES* target property is set, its value will be used. Set the *CMAKE_CXX_SCAN_FOR_MODULES* variable to initialize this property on all targets as they are created.
- Otherwise, the source file will be scanned if the compiler and generator support scanning. See policy *CMP0155*.

Note that any scanned source will be excluded from any unity build (see *UNITY_BUILD*) because module−related statements can only happen at one place within a C++ translation unit.

COMPILER SUPPORT

Compilers which CMake natively supports module dependency scanning include:

- MSVC toolset 14.34 and newer (provided with Visual Studio 17.4 and newer)
- LLVM/Clang 16.0 and newer
- GCC 14 (for the in−development branch, after 2023−09−20) and newer

IMPORT STD SUPPORT

Support for **import std** is limited to the following toolchain and standard library combinations:

- Clang 18.1.2 and newer with **−stdlib=libc++**
- MSVC toolset 14.36 and newer (provided with Visual Studio 17.6 Preview 2 and newer)

The *CMAKE_CXX_COMPILER_IMPORT_STD* variable may be used to detect support for a standard level with the active C++ toolchain.

NOTE:

This support is provided only when experimental support for **import std;** has been enabled by the **CMAKE_EXPERIMENTAL_CXX_IMPORT_STD** gate.

GENERATOR SUPPORT

The list of generators which support scanning sources for C++ modules include:

- *Ninja*
- *Ninja Multi−Config*
- *Visual Studio 17 2022*

Limitations

There are a number of known limitations of the current C++ module support in CMake. This does not document known limitations or bugs in compilers as these can change over time.

For all generators:

- Header units are not supported.
- No builtin support for **import std;** or other compiler−provided modules.

For the Ninja Generators:

- **ninja** 1.11 or newer is required.

For the *Visual Studio Generators*:

- Only Visual Studio 2022 and MSVC toolsets 14.34 (Visual Studio 17.4) and newer.
- No support for exporting or installing BMI or module information.
- No support for compiling BMIs from **IMPORTED** targets with C++ modules (including **import std**).
- No diagnosis of using modules provided by **PRIVATE** sources from **PUBLIC** module sources.

COPYRIGHT