P1838R0: Modules User-Facing Lexicon and File Extensions

ISO/IEC JTC1 SC22/WG21 - Programming Languages - C++

Authors:

Bryce Adelstein Lelbach <brycelelbach@gmail.com>
Boris Kolpackov <boris@codesynthesis.com>

Audience:

Tooling (SG15)

Motivation

C++20 modules introduces a new compilation model for C++; as with any new large feature, we need a number of new words to discuss it. This paper seeks to define and bikeshed a user-facing lexicon for modules.

This paper does **NOT** seek to bikeshed terminology in the C++ International Standard. It is solely concerned with modules terminology that will be used by the broader C++ community (C++ programmers, C++ library implementors, build system implementors, tooling implementors, compiler implementors, etc).

In addition to terminology, modules introduces new types of C++ source files and new types of C++ build artifacts. This paper additionally seeks to recommend conventions for the file extensions that should be used for these new file types.

The content of this paper is intended for the C++ Modules Ecosystem Technical Report.

For each item, we need to answer two questions:

- Do we want to define this term/file extension in the Technical Report?
- If we do want to define it, what should the name and definition be?

Terminology

Term	**Definition**
Built Module Interface (BMI)	The artifact created by a compiler to represent a module unit or header unit. The format for this representation is implementation specific and holds C++ entities, which can be represented in the form of compiler specific data structures (e.g. ASTs, metadata, etc), machine code (object files) or any intermediate representation chosen by the implementer.

	PRIOR ART: ● Compiled Module Interface (CMI) ○ Modules Are Coming, Core C++ 2019, Bryce Adelstein Lelbach ● Binary Module Interface (BMI) ○ The vast majority of pre 2019 modules content
BMI Configuration	The set of characteristics at translation time that identify compatible BMI/CMIs. These traits are implementation defined and it's recommended that for BMI/CMI to have the same configuration (and be reusable) they should at least maintain ABI and ODR compatibility.
Precompilation *or* Module Compilation	The act of creating a BMI/CMI from a module unit or header unit, possibly as a distinct step from translation of said unit. PRIOR ART: ● Precompiled headers ● Clang/LLVM's `--precompile` flag ● Modules Are Coming, Core C++ 2019, Bryce Adelstein Lelbach
Dependency Scanner	A tool which parses C++ source files and outputs their dependency metadata. Such a tool might be a standalone utility or an option that can be enabled in a C++ compiler driver. PRIOR ART: ● `clang-scan-deps` ○ `clang-scan-deps`, LLVM Euro 2019, Alex Lorenz, Michael Spencer ● P1689: Format for Describing Dependencies of Source Files, Ben Boeckel
Dependency Metadata *or* Dependency Information	Information about the translation units and textual includes of a C++ source file. This includes the name of the module the source file exports, the modules that it imports, the files it textually includes, and other metadata. PRIOR ART: ● P1689: Format for Describing Dependencies of Source Files, Ben Boeckel
Dependency Metadata Format *or* Dependency Information Format	A structured file format that describes dependency metadata.

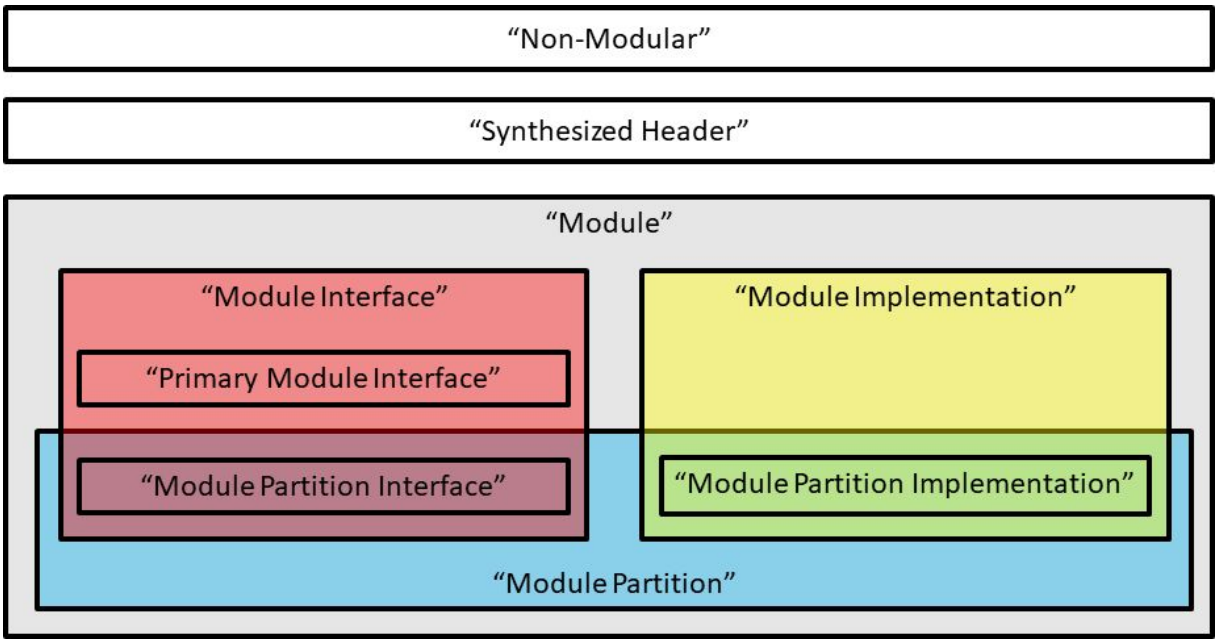
	PRIOR ART: • P1689: Format for Describing Dependencies of Source Files, Ben Boeckel
Implicit Module Builds	Modules build mode where the compiler knows how to build all components for a module and its dependencies without requiring the user to explicitly invoke the compiler for building specific parts or said dependencies. In this mode the compiler incorporates some of the roles of a build system.
Explicit Module Builds	Modules build mode where components of a module and its dependencies are built separately, with explicit compiler invocations being used to build each piece. In this mode, some external system is responsible for computing dependencies and schedule build order for the components.
Strong Module Ownership	An entity with external linkage which belongs to a module will be a different entity from, and will not conflict with, an entity with the same name in the global module.
Weak Module Ownership	An entity with external linkage which belongs to a module will conflict with (or simply be the same entity as) an entity with the same name in the global module.
Importable Headers	[module.unit] p5 s3
Include Translation	[cpp.include] p7

Translation Units

Kind of Translation Unit	**Definition**
Non-Modular Translation Unit	Translation units that are not module units or header units. Prior to C++20, we just called these translation units. PRIOR ART: • Modules Are Coming, Core C++ 2019, Bryce Adelstein Lelbach
Synthesized Header Translation Unit	Translation unit formed by a header. [module.import] p5

Primary Module Interface Translation Unit	[module.unit] p2
Module Partition Interface Translation Unit	[module.unit] p3
Module Implementation Translation Unit	[module.unit] p2
Module Partition Implementation Translation Unit	[module.unit] p3

The following diagram describes the less specific terms that can be used to refer to types of translation units. "Translation Unit" or "Unit" can be appended to the end of any of these terms.



File Extensions

Extension	**File Type**

`.cpp`
`.cppm`
`.ixx`
`.mpp`
`.mxx`
`.cmi`
	Module interface unit.
	
	`export module ...;`
	`...`
	
	PRIOR ART:
	● Modules Are Coming, Core C++ 2019, Bryce Adelstein Lelbach
	● build2 (`.mpp`/`.mxx`)
	● MSVC (`.ixx`)
	● Clang/LLVM (`.cppm`)
	● GCC (.cpp)

`.pcm`
`.gcm / .gcmu / .gcms`
`.ifc`
`.bmi`
	BMI.
	
	PRIOR ART:
	● `.bmi`:
	○ Modules Are Coming, Core C++ 2019, Bryce Adelstein Lelbach
	● `.pcm`:
	○ Clang/LLVM
	● `.gcm`:
	○ GCC
	● `.ifc`:
	○ MSVC

Prior Art

Conference Talks

- `clang-scan-deps`, LLVM Euro 2019, Alex Lorenz, Michael Spencer
- Modules Are Coming, Core C++ 2019, Bryce Adelstein Lelbach

ISO C++ Committee Papers

- P1689: Format for Describing Dependencies of Source Files, Ben Boeckel

Blog Posts

- C++ Modules Might Be Dead-on-Arrival, January 2019, vector-of-bool