Restoring Private Module Fragments

Document #:	P3838R0
Date:	2025-08-13
Project:	Programming Language C++
Audience:	Core Working Group

Reply-to:	Alisdair Meredith <ameredith1@bloomberg.net>

Contents

1 Abstract

This paper restores the preprocessor grammar that permits the declaration of private module fragments during translation phase 7.

2 Revision history

R0 August 2025 (pre-Kona mailing)

Initial draft of this paper.

3 Analysis

Paper [P3034R1] amended the grammar for *pp-module* directives to forbid macro names in the identifiers that will become the module name and partition in translation phase 7. In amending the grammar, the *module-name* before the *module-partition* became non-optional. However, this change then makes a private module fragment ill-formed in translation phase 4:

```
module : private;
```

Note the keyword `private` is not a partition name in phase 7, even though that is the grammar position it occupies in phase 4. Most importantly though, observe the lack of a module name.

The fix seems simple, just make the *module-name* optional. However, once we make this change we can simplify the presentation considerably. In order to ensure that the *module-name* is a non-optional part of the optional *pp-tokens* following the identifier `module`, we moved the grammar decomposing the module name and partition into a separate specification that redundantly defines how to parse a sequence of preprocessor tokens. Now that all parts are optional we can integrate the grammar where the rules of the language solve those concerns more easily.

In addition, the first paragraph is entirely redundant after the application of [P2843R2] that makes all use of the specified identifiers as macro names ill-formed. We propose turning that paragraph into a note as part of the simplification of this clause.

4 Wording

Make the following changes to the C++ Working Draft. All wording is relative to [N5008], the latest draft at the time of writing.

15.5 [cpp.module] Module directive

pp-module:
`export`$_{\text{opt}}$ `module` *pp-module-name*$_{\text{opt}}$ *pp-module-partition*$_{\text{opt}}$ *pp-tokens*$_{\text{opt}}$ `;` new-line

1 A *pp-module* shall not appear in a context where `module` or (if it is the first preprocessing token of the *pp-module*) `export` is an identifier defined as an object-like macro.

2 The *pp-tokens*, if any, of a *pp-module* shall be of the form:

pp-module-name pp-module-partition$_{\text{opt}}$ *pp-tokens*$_{\text{opt}}$

where the *pp-tokens* (if any) shall not begin with a `(` preprocessing token and the grammar non-terminals are defined as:

pp-module-name:
pp-module-name-qualifier$_{\text{opt}}$ *identifier*

pp-module-partition:
`:` *pp-module-name-qualifier*$_{\text{opt}}$ *identifier*

pp-module-name-qualifier:
identifier `.`
pp-module-name-qualifier identifier `.`

No identifier in the *pp-module-name* or *pp-module-partition* shall currently be defined as an object-like macro.

3 Any preprocessing tokens after the `module` preprocessing token in the `module` directive are processed just as in normal text.

1 No identifier in the *pp-module-name* or *pp-module-partition* shall currently be defined as an object-like macro.

[*Note:* The identifiers `export` and `module` are not valid macro names. — *end note*]

2 The *pp-tokens*, if any, of a *pp-module* shall not begin with a `(` preprocessing token and are processed just as in normal text.

[*Note:* Each identifier currently defined as a macro name is replaced by its replacement list of preprocessing tokens. — *end note*]

43 The `module` and `export` (if it exists) preprocessing tokens are replaced by the *module-keyword* and *export-keyword* preprocessing tokens respectively.

[*Note:* This makes the line no longer a directive so it is not removed at the end of phase 4. — *end note*]

5 Acknowledgements

Thanks to Michael Park for the pandoc-based framework used to transform this document's source from Markdown. Thanks to Michael Spencer for the original work on this feature, and to David Herring for confirming my understanding of the issue.

6 References

[N5008] Thomas Köppe. 2025-03-15. Working Draft, Programming Languages — C++.
https://wg21.link/n5008

[P2843R2] Alisdair Meredith. 2025-03-17. Preprocessing is never undefined.
https://wg21.link/p2843r2

[P3034R1] Michael Spencer. 2024-03-21. Module Declarations Shouldn't be Macros.
https://wg21.link/p3034r1