

# A Grammar for Whitespace Characters

Resolution for NB comment US 5-108

Document #: P3657R2  
Date: 2025-11-03  
Project: Programming Language C++  
Audience: Core  
Reply-to: Alisdair Meredith  
<[ameredith1@bloomberg.net](mailto:ameredith1@bloomberg.net)>

## Contents

<b>Abstract</b>	<b>2</b>
<b>Revision history</b>	<b>2</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Purpose . . . . .	3
1.2 Audience . . . . .	3
1.3 Details Resolved by Papers Adopted for C++26 CD . . . . .	3
1.4 Previous work . . . . .	3
<b>2 Proposed Solution</b>	<b>4</b>
<b>3 Review History</b>	<b>5</b>
3.1 SG16 Teleconference 2025-05-14 . . . . .	5
3.2 SG16 Teleconference 2025-10-21 . . . . .	5
<b>4 Future Treatment of Vertical Whitespace</b>	<b>5</b>
<b>5 Wording</b>	<b>7</b>
<b>6 Acknowledgements</b>	<b>10</b>
<b>7 References</b>	<b>10</b>

# Abstract

This proposal addresses NB comment US 5-108 by providing a rigorous treatment of the C++ grammar for whitespace characters. It clarifies the Standard regarding whitespace in general, but makes no functional changes.

## Revision history

### R2 October 2025 (Kona meeting)

- Record feedback from SG16 telecon, 2025-10-21
  - note we no longer resolve [CWG2002]
  - forward to CWG
- Moved C++26 Resolved Issues into Introduction, following Previous Work
- Clarified why the last *preprocessing-token* production is not impacted
- Wording updates
  - Rebased wording onto the current working draft
  - Fixed typos in proposed wording
  - Resolve editorial issue #5502 following (limited) feedback from SG16 reflector.

### R1 October 2025 (SG16 Telecon)

- Rebased wording onto [N5014]
- Explicitly address NB comment US 5-108
- Significantly restructured the wording
- Dropped all suggestions that might change behavior
- Struck all references to whitespace separating tokens (rather than preprocessing tokens)
- Simplified comparison with paper [P2348R3]
- Recorded interactions with papers that landed for the C++26 CD
- Added future direction to handle vertical whitespace

### R0 March 2024 (post-Hagenberg mailing)

Initial draft of this paper.

# 1 Introduction

## 1.1 Purpose

This paper is concerned with the underspecification of whitespace characters, an existing term of art that the Standard uses in important and specific ways yet is not formally specified. Providing that specification leads to the question of whether or not new-line characters are whitespace characters and resolving that question leads to formally specifying *whitespace* as a term of power in a single place.

The intent of this paper is make no functional change to the language while bringing clarity and rigor to the Standard specification for whitespace.

The initial version of this proposal made some small normative changes to simplify the treatment of whitespace, but all such suggestions have been removed. The most useful of those suggestions have already been applied to the C++26 CD via other papers.

## 1.2 Audience

This paper is targeted directly at the Core Working Group rather than Evolution because it aims to deliver no functional change — i.e., the content should be purely editorial in nature.

This paper is addressed to the Core Working Group rather than the project editors because the changes are not minimal and risk accidentally changing meaning if not reviewed by Core experts.

## 1.3 Details Resolved by Papers Adopted for C++26 CD

- [P2996R13] merged phases 7 and 8 of translation, and addressed our one minor concern in the process.
- [P2843R3] changed the restrictions on which whitespace characters are allowed in comments.

## 1.4 Previous work

Corentin Jabot has been making progress with a larger treatment of whitespace in general through paper [P2348R3], last updated in September 2022. It needs a major revision to rebase onto the current C++ working paper as there have been significant changes to the clauses it amends.

That paper provides a full treatment bringing *all* whitespace into the C++ grammar, including comments, retaining the full content of comments rather than replacing them with a single space in phase 3 of translation. It further provides an extensive treatment of line-breaks, supporting both U+000D CARRIAGE RETURN and U+000A LINE FEED as new-line markers, which makes up the majority of the wording changes. Finally, it resolves [CWG1655] by revising the grammar associated with new-line characters

This paper would serve as a foundation should work resuming on that larger proposal. Both this paper and [P2348R3] treat new-line as *whitespace* that is not a whitespace character. The grammar for *whitespace-character* in this paper maps directly to the grammar for *horizontal-whitespace* in [P2348R3].

This paper has deferred any proposed resolution to [CWG2002]. Its preferred long term resolution is described in [Future Treatment of Vertical Whitespace](#), where [P2348R3] would remove the conflicted wording in 15.1 [cpp.pre]p5 allowing vertical whitespace to separate preprocessing token in a preprocessing directive.

## 2 Proposed Solution

The approach of this paper is to introduce a new subsection that defines *whitespace* as a term of power, where it also provides a grammar definition for *whitespace-characters* taken from the clearest definition of the term “whitespace character”.

There are three complementary definitions of “whitespace character” under 5 [lex] but none of them are a term of power. Similarly, the term “whitespace” is defined parenthetically but not as a term of power.

According to 5.10 [lex.token]p.1 (citing from [N5008])

Blanks, horizontal and vertical tabs, newlines, form-feeds, and comments (collectively, “whitespace”), as described below, are ignored except as they serve to separate tokens.

This definition of “whitespace” informally specifies the set of characters that are considered whitespace characters — blanks, horizontal and vertical tabs, newlines, and form-feeds. Note that “blanks” is not a defined term, but assumed to mean the space character U+0020 SPACE. The primary definition of whitespace characters is given in 5.5 [lex.pptoken]p1

... Preprocessing tokens can be separated by whitespace; this consists of comments 5.4 [lex.comment], or whitespace characters (U+0020 SPACE, U+0009 CHARACTER TABULATION, new-line, U+000B LINE TABULATION, and U+000C FORM FEED), or both. ...

A similar specification for the set of whitespace characters can be extracted from the grammar for *d-char* in the specification of raw string literals.

A new grammar production, *whitespace-character*, will define whitespace characters more clearly. For practical purposes this paper excludes new-line from the set of whitespace characters. Most uses of the term “whitespace character” in the current Standard immediately follow up with “excluding new-line”. After applying the proposed change there were no occurrences of *whitespace-character* that would have to add an extra reference to new-line compared to the current Standard wording.

Note that there may be a subtle change to how the *preprocessing-token* grammar is interpreted, still achieving the same effect as the current Standard. Observe the last part of the grammar for *preprocessing-token*:

each non-whitespace character that cannot be one of the above

Non-whitespace characters are technically irrelevant, as whitespace is what separates preprocessing tokens so does not participate in this part of the grammar. As new-line continues to be whitespace, although not a *whitespace-character*, it continues to serve as a separator of *preprocessing-tokens*. Similarly comments are not listed here, as they are transformed into spaces in phase 3 of translation. Both of these observations might be picked up by a future paper, such as a revision of [P2348R3] that incorporates both new-line and comments into the formal grammar.

Following an editorial trend that was confirmed in [P2843R3], all uses of “space” to mean “the space character” are replaced with the Unicode code point U+0020 SPACE, and similarly all “horizontal tab”, “vertical tab”, and “form feed” characters are replaced by the corresponding precise forms U+0009 CHARACTER TABULATION, U+000B LINE TABULATION, and U+000C FORM FEED. Then adjust surrounding text to similarly specify characters as their Unicode code point only where that would improve consistency.

Do **not** adjust any uses of new-line to U+000A LINE FEED as we wish to leave that space clear for [P2348R3] and its broader treatment of line-breaks.

## 3 Review History

### 3.1 SG16 Teleconference 2025-05-14

- concern expressed that we should follow Unicode treatment of vertical whitespace
- concern that the paper reaches too far into semantic changes
  - recommendation to leave fixing UB to [P2843R3]
- the introduction of *whitespace-character* into the grammar seems “useful”
- clarification that whitespace matters only for separating preprocessing tokens and not regular tokens

### 3.2 SG16 Teleconference 2025-10-21

Concern was raised about explicit use of Unicode code points, but agreement that this is the direction that the specification has moved, and this paper is consistent with following that direction, hopefully completing the remaining outstanding changes.

It was noted that this revision no longer resolves [CWG2002], which is a consequence of stepping back to make no normative changes. Updated the expository material introducing this paper to make that clearer.

Concern that as a non-functional change this might not be needed for C++26, notably a concern that it would take CWG time away from more important concerns. However, it does directly address an editorial NB comment, and the CWG Chair is well qualified to prioritise CWG work.

Strong consensus to forward P3657R1 to CWG as a recommended resolution for US 5-018,

## 4 Future Treatment of Vertical Whitespace

To make progress on the important matters of specification, this paper now drops all edits that would change C++26 behavior with regard to vertical whitespace, notably vertical tabs and form feeds.

However, reviews of this paper and [P2843R3] showed interest in taking a principled approach to adopt a consistent approach to vertical whitespace throughout the language. There is currently an between the treatment of vertical whitespace in preprocessing directives vs. vertical whitespace in “regular” code. There is also an inconsistency between C++ and the recommended practice coming for Unicode, which recommends treating all vertical whitespace, including code-points that C++ does not yet recognise, as line-breaks with the semantic meaning that entails.

To address these concerns, we provide the tentative plans to address all vertical whitespace concerns in C++29, so that the editorial changes in this paper can be seen to support such a direction.

An overriding concern of the author is that vertical whitespace has become quite esoteric, largely divorced from its meaning in the era of hard copy and printouts, awkward to enter as no current keyboards have keys for those characters, and frequently badly rendered in user interfaces in ways that might lead to misleading presentation of code once vertical whitespace takes on a normative meaning, such as the Unicode recommendation to consistently treat all such whitespace as line-breaks.

Hence, the planned future direction will be to remove the inherent complexity and unfamiliarity of vertical whitespace from the language. A preliminary (but substantial) code search suggests there are no active uses of U+000B LINE TABULATION in the wild, but U+000C FORM FEED has a special place as a page separator in the Gnu coding conventions. It should be sufficient to retain support for only text lines having just horizontal whitespace and form feeds — in fact, it is not clear that support for horizontal whitespace on those lines is needed, but the C++ grammar would be easier that way.

We will propose further simplifying whitespace by:

A complete set of wording changes would:

- Add U+000B LINE TABULATION to `[tab:lex.charset.literal]`
- Strike U+000B LINE TABULATION from `[tab:lex.charset.basic]`
- Strike U+000B LINE TABULATION and U+000C FORM FEED from *whitespace-character*
- Strike [Note 1] from 5.4.2 `[lex.whitespace.char]p1`
- Strike 15.1 [\[cpp.pre\]p5](#)
- Add a second null directive production that allows a sequence of U+000C FORM FEED followed by a new-line
- Resolves Core issue [\[CWG2002\]](#)

U+000C FORM FEED is retained in the basic character set so that we do not make a program ill-formed during phase 3 where we would fail to form a valid preprocessing token before it could be discarded as a null directive at the end of phase 4.

## 5 Wording

Make the following changes to the C++ Working Draft. All wording is relative to [N5014], the latest draft at the time of writing.

### 5.2 [lex.phases] Phases of translation

- <sup>2</sup> If the first translation character is U+FEFF BYTE ORDER MARK, it is deleted. Each sequence of a ~~backslash character~~U+005C REVERSE SOLIDUS (\) immediately followed by zero or more ~~whitespace characters other than new-line~~*whitespace characters* followed by a new-line character is deleted, splicing physical source lines to form *logical source lines*. Only the last backslash on any physical source line shall be eligible for being part of such a splice.

[Note 2: Line splicing can form a *universal-character-name* (5.3.1 [lex.charset]). —end note]

A source file that is not empty and that (after splicing) does not end in a new-line character shall be processed as if an additional new-line character were appended to the file.

- <sup>3</sup> The source file is decomposed into preprocessing tokens (5.5 [lex.pptoken]) and ~~sequences of whitespace characters (including comments)~~ *whitespace* (5.4 [lex.whitespace]). A source file shall not end in a partial preprocessing token or in a partial comment.<sup>1</sup> Each comment (5.4.1 [lex.comment]) is replaced by one U+0020 SPACE character. New-line characters are retained. Whether each nonempty sequence of ~~whitespace characters other than new-line~~*whitespace characters* is retained or replaced by one U+0020 SPACE character is unspecified. As characters from the source file are consumed to form the next preprocessing token (i.e., not being consumed as part of a comment or other forms of whitespace), except when matching a *c-char-sequence*, *s-char-sequence*, *r-char-sequence*, *h-char-sequence*, or *q-char-sequence*, *universal-character-names* are recognized (5.3.2 [lex.universal.char]) and replaced by the designated element of the translation character set (5.3.1 [lex.charset]). The process of dividing a source file's characters into preprocessing tokens is context-dependent.

[Example 1: See the handling of < within a #include preprocessing directive (15.3 [cpp.include]). —end example]

- <sup>4</sup> The source file is analyzed as a *preprocessing-file* (15.1 [cpp.pre]). Preprocessing directives (Clause 15 [cpp]) are executed, macro invocations are expanded (15.7 [cpp.replace]), and \_Pragma unary operator expressions are executed (15.13 [cpp.pragma.op]). A #include preprocessing directive (15.3 [cpp.include]) causes the named header or source file to be processed from phase 1 through phase 4, recursively. All preprocessing directives are then deleted. Whitespace ~~characters~~ separating preprocessing tokens ~~are~~*is* no longer significant.

### 5.4 Whitespace [lex.whitespace]

#### 5.4.1 Comments [lex.comment]

- <sup>1</sup> The characters /\* start a comment, which terminates with the characters \*/. These comments do not nest. The characters // start a comment, which terminates immediately before the next new-line character.

[Note 1: The comment characters //, /\*, and \*/ have no special meaning within a // comment and are treated just like other characters. Similarly, the comment characters // and /\* have no special meaning within a /\* comment. —end note]

[Note 2: Comments are turned into U+0020 SPACE characters in phase 3 of translation as part of decomposing a source file into preprocessing tokens and whitespace. —end note]

#### 5.4.2 Whitespace Characters [lex.whitespace.char]

*whitespace-character:*

U+0009 CHARACTER TABULATION  
U+000B LINE TABULATION  
U+000C FORM FEED  
U+0020 SPACE

---

<sup>1</sup>A partial preprocessing token would arise from ...

- <sup>1</sup> Sequences of *whitespace-characters*, new-line characters, and comments (5.4 [lex.comment]) form *whitespace*, which carries no semantic significance other than to separate preprocessing tokens (5.5 [lex.pptoken]).

[*Note 1*: Implementations are permitted but not required to coalesce non-empty sequences of whitespace into a single U+0020 SPACE while retaining new-lines 5.2 [lex.phases]. —*end note*]

[*Note 2*: As described in Clause 15 [cpp], in certain circumstances during translation phase 4, whitespace (or the absence thereof) serves as more than preprocessing token separation. U+000B LINE TABULATION and U+000C FORM FEED are not used to separate preprocessing tokens within a preprocessing directive (15.1 [cpp.pre]). Whitespace can appear within a preprocessing token only as part of a header name or between the U+0027 APOSTROPHE characters in a character literal or between the U+0022 QUOTATION MARK characters in a string literal. —*end note*]

## 5.5 [lex.pptoken] Preprocessing tokens

*preprocessing-token*:

*header-name*

*import-keyword*

*module-keyword*

*export-keyword*

*identifier*

*pp-number*

*character-literal*

*user-defined-character-literal*

*string-literal*

*user-defined-string-literal*

*preprocessing-op-or-punc*

each non-~~whitespace-character~~whitespace-character that cannot be one of the above

- <sup>1</sup> A preprocessing token is the minimal lexical element of the language in translation phases 3 through 6. In this document, glyphs are used to identify elements of the basic character set (5.3.1 [lex.charset]). The categories of preprocessing token are: header names, placeholder tokens produced by preprocessing `import` and `module` directives (*import-keyword*, *module-keyword*, and *export-keyword*), identifiers, preprocessing numbers, character literals (including user-defined character literals), string literals (including user-defined string literals), preprocessing operators and punctuators, and single non-~~whitespace-character~~whitespace-characters that do not lexically match the other preprocessing token categories. If a U+0027 APOSTROPHE, ~~or~~ a U+0022 QUOTATION MARK, ~~or~~ ~~character matches the last category, the program is ill-formed.~~ If any character not in the basic character set matches the last category, the program is ill-formed. ~~Preprocessing tokens can be separated by whitespace; this consists of comments 5.4 [lex.comment], or whitespace characters (U+0020 SPACE, U+0009 CHARACTER TABULATION, new-line, U+000B LINE TABULATION, and U+000C FORM FEED), or both. As described in Clause 15 [cpp], in certain circumstances during translation phase 4, whitespace (or the absence thereof) serves as more than preprocessing token separation. Whitespace can appear within a preprocessing token only as part of a header name or between the quotation characters in a character literal or string literal.~~
- <sup>2</sup> Each preprocessing token that is converted to a token (5.10 [lex.token]) ...



## 5.10 [lex.token] Tokens

*token:*

*identifier*

*keyword*

*literal*

*operator-or-punctuator*

- <sup>1</sup> There are five kinds of tokens: identifiers, keywords, literals, operators, and other separators. ~~Comments and the characters U+0020 SPACE, U+0009 CHARACTER TABULATION, U+000B LINE TABULATION, U+000C FORM FEED, and new-line (collectively, “whitespace”), as described below, are ignored except as they serve to separate tokens. [Note 1: Whitespace can separate otherwise adjacent identifiers, keywords, numeric literals, and alternative tokens containing alphabetic characters.—end note]~~

### 5.13.5 [lex.string] String literals

*string-literal:*

*encoding-prefix*<sub>opt</sub> " *s-char-sequence*<sub>opt</sub> "

*encoding-prefix*<sub>opt</sub> R *raw-string*

*s-char-sequence:*

*s-char* *s-char-sequence*<sub>opt</sub>

...

*d-char-sequence:*

*d-char* *d-char-sequence*<sub>opt</sub>

*d-char:*

any member of the basic character set except:

~~*whitespace-character*, U+0020 SPACE, U+0028 LEFT PARENTHESIS, U+0029 RIGHT PARENTHESIS, U+005C REVERSE SOLIDUS, U+0009 CHARACTER TABULATION, U+000B LINE TABULATION, U+000C FORM FEED, and new-line~~

## 15 [cpp] Preprocessing directives

### 15.1 [cpp.pre] Preamble

- <sup>7</sup> The only ~~whitespace-character~~*whitespace-characters* that shall appear between preprocessing tokens within a preprocessing directive (from just after the directive-introducing token through just before the terminating new-line character) are ~~space and horizontal-tab~~ *U+0020 SPACE and U+0009 CHARACTER TABULATION* (including spaces that have replaced comments or possibly other ~~whitespace-character~~*whitespace-characters* in translation phase 3).

### 15.7 [cpp.replace] Macro replacement

#### 15.7.1 [cpp.replace.general] General

- <sup>8</sup> The identifier immediately following the **define** is called the *macro name*. There is one name space for macro names. Any ~~whitespace-character~~*whitespace-characters* preceding or following the replacement list of preprocessing tokens are not considered part of the replacement list for either form of macro.
- <sup>13</sup> A preprocessing directive of the form ...
- ... Within the sequence of preprocessing tokens making up an invocation of a function-like macro, new-line is considered a ~~normal-whitespace-character~~*whitespace-character*.
- <sup>14</sup> The sequence of preprocessing tokens bounded by ...

## 6 Acknowledgements

Thanks to Michael Park for the pandoc-based framework used to transform this document's source from Markdown.

## 7 References

[CWG1655] Mike Miller. 2013-04-26. Line endings in raw string literals.  
<https://wg21.link/cwg1655>

[CWG2002] Richard Smith. 2014-09-10. White space within preprocessing directives.  
<https://wg21.link/cwg2002>

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

[N5014] Thomas Köppe. 2025-08-05. Working Draft, Standard for Programming Language C++.  
<https://wg21.link/n5014>

[P2348R3] Corentin Jabot. 2022-09-11. Whitespaces Wording Revamp.  
<https://wg21.link/p2348r3>

[P2843R3] Alisdair Meredith. 2025-07-18. Preprocessing is never undefined.  
<https://wg21.link/p2843r3>

[P2996R13] Barry Revzin, Wyatt Childers, Peter Dimov, Andrew Sutton, Faisal Vali, Daveed Vandevoorde, Dan Katz. 2025-06-20. Reflection for C++26.  
<https://wg21.link/p2996r13>