A Grammar for Whitespace Characters

Precursor to P2348

Document #:	P3657R0
Date:	2025-03-15
Project:	Programming Language C++
Audience:	Core

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

Contents

1 Abstract

This proposal provides a rigorous treatment of the C++ grammar for whitespace characters while leaving room for a future revision of [P2348R3] to more completely address the whole domain of whitespace, notably comments and line breaks.

2 Revision history

R0 March 2024 (post-Hagenberg mailing)

Initial draft of this paper.

3 Introduction

3.1 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, matching its existing usage, in a single place.

The intent of this paper is to be the minimal incremental change that can add clarity and rigor to the Standard specification, which can then serve as a foundation for more ambitious changes in follow-up papers. It should make no functional changes to the Standard, being purely editorial clean-up. However, to simplify the changes while observing the parallel progress [P2843], the EWG-approved change to valid whitespace characters in 5.4 [lex.comment] is adopted here too — making a normative change if [P2843] fails to pass as a whole.

Similarly, a resolution is provided for [CWG1655] as that is simpler than respecifying in a way that preserves the issue. However, wording is also available that will preserve this issue if that would ease concerns raised by this paper.

If CWG were to approve the contents of this paper as purely editorial in nature, the two normative changes would be replaced by strictly editorial updates that deliberately do not address those concerns. The first change will still be applied by [P2843] but would no longer be an wording conflict. The core issue would be resolved by the normal issues process.

3.2 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.

3.3 Previous work

Corentin Jabot has been making progress with a larger treatment of whitespace in general through paper [P2348R3], last updated in September 2002. That paper provides a full treatment bringing *all* whitespace into the C++ grammar, including comments, and retaining the spelling of comments rather than replacing them with a single space in phase 3 of translation. That paper also provides an extensive treatment of new-lines, supporting both `U+000D CARRIAGE RETURN` and `U+000A LINE FEED` as new-line markers, which makes up the majority of the wording changes.

In addition to the concerns addressed by this paper, [P2348R3] addresses more deep-rooted concerns with the treatment of whitespace with the following additional changes:

- provides grammar for comments
- preserves comments, rather than turning into spaces
- defines a *line-break* element that includes both carriage return and new-line
  - no longer necessary due to phase 1 clean-up in [P2295R6] adopted for C++26
  - reinforces preserving all whitespace without any character conversion
- defines horizontal whitespace without new-line or other *line-break* elements
- removes restrictions on specific whitespace characters in comments
- resolves [CWG1655] by revising the grammar associated with new-line characters

3.4 Comparing papers

Note that [P2348R3] currently needs a significant update to rebase onto the current C++ working paper as there have been significant changes to the clauses it amends. This paper intends so be a minimal but complete update to the Standard specification that can be easily reviewed an land in the C++26 project. It is forward-looking to serve as a foundation for future revisions of P2348.

3.4.1 Common foundation

Both papers 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].

Both papers implement the simplification the was approved in the 2025 Hagenberg meeting for what constitutes valid whitespace in a comment in 5.4 [lex.comment].

Both papers aim to resolve Core issue [CWG2002] by removing the conflicted wording in 15.1 [cpp.pre]p5.

3.4.2 Additional content in [P2348R3]

All whitespace is specified by the C++ grammar.

— comments are fully described by the C++ grammar
— *line-break* is introduced to handle multiple line-breaking conventions

Comments are retained, rather than replaced by a single space character. This change might support future language extensions that directly address the use of comments in a similar way that many implementations use them to disable diagnostics today.

`U+000D CARRIAGE RETURN` is promoted to a *line-break* character. This change needs reworking to accommodate changes applied by [P2295R6] to the C++23 Standard; the source mapping specified in translation phase 1 now maps all use of that code point into the new-line character, `U+000A LINE FEED`, so is no longer present in the source seen by translation phases 2 and later.

4 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-character*s 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

> Blanks, horizontal and vertical tabs, newlines, formfeeds, 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 formfeeds. 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 *pp-token*:

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

Non-whitespace characters are relevant, as whitespace is what separates tokens. Comments are not listed here, as they are transformed into spaces in phase 3 of translation, and that is where new-line characters are also removed under the more general guise of whitespace, in the proposed wording. If new-line characters were deemed to survive to this point then they would become a *pp-token* matching this last term in the current grammar. However as new-line continues to be whitespace but not a whitespace character it continues to serve as a separator of tokens. This additional use as a *pp-token* becomes useful when *new-line* becomes a grammar term in 15.1 [cpp.pre]. A future paper, such as a revision of [P2348R3], might want to move the *new-line* grammar into 5 [lex].

Next, all uses of "space" to mean "the space character" are replaced with the Unicode code point U+0020 SPACE, and similarly all " horizontal tab"s, "vertical tab"s, and "form feed" 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] that has different ideas of how to integrate new-lines and more general whitespace more closely into the grammar.

5 Interaction with Other Papers

Several papers being concurrently processed with this paper may appear to have an interaction, in addition to [P2348R3]. All such interactions are addressed within this paper.

[P2996] Reflection effectively merges phases 7 and 8 of translation. However, all the wording changes this paper applies to the phases of translation come before that point, so there is no interaction. Note that while both papers touch phase 7, this paper touches the very beginning of phase 7 while P2996 starts making changes towards the end of the wording for this phase.

[P2843] changes the restrictions on which whitespace characters are allowed in comments. This paper applies the same change (which is an EWG-approved normative change) to resolve any merge conflicts.

[P3556] replaces several uses to the term "source file" with the new term "source text". Where these edits apply to paragraphs touched by this paper, they are never changing the same words so it should be simple for the editors to merge parallel changes. Both papers agrees on the semantics following any merging of their changes.

6 Wording

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

5.2 [lex.phases] Phases of translation

2 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-character*s 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.

3 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.[1] Each comment (5.4.2 [lex.comment]) is replaced by one ~~space~~U+0020 SPACE character. New-line characters are retained. Whether each nonempty sequence of ~~whitespace characters other than new-line~~ *whitespace-character*s is retained or replaced by one ~~space~~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-name*s 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*]

7 Each preprocessing token is converted into a token (5.10 [lex.token]). Whitespace ~~characters~~ separating tokens ~~are~~is no longer significant. The resulting tokens constitute a *translation unit* and are syntactically and semantically analyzed and translated.

5.4 Whitespace [lex.whitespace]

5.4.1 General [lex.whitespace.general]

1 Sequences of *whitespace-character*s (5.4.3 [lex.whitespace.char]), new-line characters, and comments (5.4 [lex.comment]) form *whitespace*, which carries no semantic significance other than to separate tokens (5.10 [lex.token]) and preprocessing tokens (5.5 [lex.pptoken]).

2 [*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. —*end note*]

5.4.2 Comments [lex.comment]

1 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. ~~If there is a form-feed or a vertical-tab character in such a comment, only whitespace characters shall appear between it and the new-line that terminates the comment; no diagnostic is required.~~

[*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*]

[1] A partial preprocessing token would arise from …

5.4.3 Whitespace Characters [lex.whitespace.char]

whitespace-character:
U+0009 CHARACTER TABULATION
U+000B LINE TABULATION
U+000C FORM FEED
U+0020 SPACE

1 Preprocessing tokens (5.5 [lex.pptoken]) are separated by sequences of *whitespace-character*s.

2 [*Note 1:* Comments are turned into U+0020 SPACE characters in phase 3 of translation as part of decomposing a source file into preprocessor tokens and whitespace. —*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

1 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). 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-character*s that do not lexically match the other preprocessing token categories. If a U+0027 APOSTROPHE or a U+0022 QUOTATION MARK 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 (5.4 [lex.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.

2 Each preprocessing token that is converted to a token (5.10) …

5.10 [lex.token] Tokens

token:
identifier
keyword
literal
operator-or-punctuator

1 There are five kinds of tokens: identifiers, keywords, literals,[2] operators, and other separators. ~~Blanks, horizontal and vertical tabs, newlines, formfeeds, and comments (collectively, "whitespace"), as described below, are ignored~~

[2] Literals include strings and character and numeric literals.

~~except as they serve to separate tokens.~~ Whitespace (5.4) is ignored except 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$_{opt}$ " *s-char-sequence*$_{opt}$ "
encoding-prefix$_{opt}$ `R` *raw-string*

s-char-sequence:
s-char s-char-sequence$_{opt}$

…

d-char-sequence:
d-char d-char-sequence$_{opt}$

d-char:
any member of the basic character set except:
~~U+0020 SPACE,~~ U+0028 LEFT PARENTHESIS, U+0029 RIGHT PARENTHESIS, U+005C REVERSE SOLIDUS, ~~U+0009 CHARACTER TABULATION, U+000C FORM FEED, U+000B LINE TABULATION,~~ *whitespace-character*s, and new-line

15 [cpp] Preprocessing directives

15.1 [cpp.pre] Preamble

6 The only 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 (including spaces that have replaced comments or possibly other whitespace characters in translation phase 3).

15.6 [cpp.replace] Macro replacement

15.6.1 [cpp.replace.general] General

8 The identifier immediately following the `define` is called the *macro name*. There is one name space for macro names. Any ~~whitespace character~~*whitespace-character*s preceding or following the replacement list of preprocessing tokens are not considered part of the replacement list for either form of macro.

12 A preprocessing directive of the form …

… The replaced sequence of preprocessing tokens is terminated by the matching) preprocessing token, skipping intervening matched pairs of left and right parenthesis preprocessing tokens. 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*.

13 The sequence of preprocessing tokens bounded by …

7 Acknowledgements

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

8 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. Working Draft, Programming Languages — C++.
https://wg21.link/n5008

[P2295R6] Corentin Jabot, Peter Brett. 2022-07-01. Support for UTF-8 as a portable source file encoding.
https://wg21.link/p2295r6

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

[P2843] Alisdair Meredith. Preprocessing is never undefined".
https://wg21.link/p2843

[P2996] Wyatt Childers, Peter Dimov, Dan Katz, Barry Revzin, Andrew Sutton, Faisal Vali, Daveed Vandevoorde. Reflection for C++26.
https://wg21.link/p2996

[P3556] Alisdair Meredith. Input Files Are Source Files.
https://wg21.link/p3556