# `#line` is not in line with existing implementation


| | |
|---|---|
| Document #: | P4136R0 |
| Date: | 2026-03-22 |
| Programming Language C++ | |
| Audience: | EWG |


Reply-to: Corentin Jabot <corentin.jabot@gmail.com>


## Abstract


The restrictions on `#line` do not match existing practices. This addresses FR-009-108 15.8


## Motivation

P2843R3 [1] made the following change.

> **Line control** **[cpp.line]**
>
> *line-directive:*
> `# line` *pp-tokens new-line*
>
> The *string-literal* of a `#line` directive, if present, shall be a character string literal.
>
> The *line number* of the current source line is the line number of the current physical source line, i.e., it is one greater than the number of new-line characters read or introduced in translation phase 1[lex.phases] while processing the source file to the current preprocessing token.
>
> A preprocessing directive of the form `# line` *digit-sequence new-line* causes the implementation to behave as if the following sequence of source lines begins with a source line that has a line number as specified by the digit sequence (interpreted as a decimal integer). If the digit sequence specifies zero or a number greater than 2147483647, the ~~behavior is undefined~~ program is ill-formed.

This is great because we want to avoid talking about UB during translation. And generally avoid unnecessary UBs. However, it forces an implementation to diagnose on the following directives:

```
#line 0
#line 2147483648
```

Yet, only EDG diagnoses the first directives, and all implementations accept the second. So the standard is overly restrictive.

In fact, testing Clang, EDG, GCC, and MSVC,

```
#line 0 // edg rejects
#line 2147483647
#line 2147483648 // all accept
#line 4294967295 // all accept
#line 4294967296 // GCC warns, edg and clang reject
#line 9223372036854775807 // GCC warns, edg, and clang reject
#line 9223372036854775808 // GCC warns, edg, and clang reject
```

So the UB that existed in the standard was used as an extension point by implementations, extension point that we accidentally took away, forcing implementations to emit a warning.

## Solution

Given that all implementations use different strategies to represent source locations, which have a significant impact on compiler performance, we cannot reasonably mandate widening the requirements.

We should also observe that if a value is accepted by `#line`, the extent to which it is actually preserved in diagnostics varies.

We should also not force a diagnostic in these situation, as lots od existing code, often generated, use these extensions. For example, I've found a few thousands instances of `#line 0`.

This leaves us with two solutions:

- Make the values outside of [1, 2147483647] conditionally supported.
- Make the whole thing implementation supported.

I prefer the second, as the number of physical lines supported is already a QoI. This diverges from C - where values outside of [1, 2147483647] are still UB. But we should realize that no implementation is going to change as a result of this paper, or of the status quo.

Note that in all cases, negative numbers are precluded by the grammar.

## Proposed wording

[*Editor's note:* Modify [cpp.line] as follows]

If the digit sequence specifies ~~zero or a number greater than 2147483647~~ a number outside of an implementation-defined range of values, the program is ill-formed.

## Alternative wording, not favored

[*Editor's note:* Modify [cpp.line] as follows]

~~If the digit sequence specifies zero or a number greater than 2147483647, the program is ill-formed.~~ Digit sequences representing a number outside of the range [0,2147483647] are conditionally supported.

## Acknowledgnments

Thanks to Alisdair Meredith for his work on P2843R3!

## References

[1] Alisdair Meredith. P2843R3: Preprocessing is never undefined. https://wg21.link/p2843r3, 7 2025.

[N5008] Thomas Köppe *Working Draft, Standard for Programming Language C++* https://wg21.link/N5008