# P1208R6Adopt source_location for C++20

Robert Douglas, Corentin Jabot, Daniel Krügler, Peter Sommerlad

2019-07-19

| | |
|---|---|
| Document Number: | P1208R6 |
| Audience: | LWG |
| Date: | 2019-07-19 |
| Project: | Programming Language C++ |

## 1 Changes and Discussions made in Cologne 2019

A summary of changes made in Cologne to the Latex version (this) by Robert, Daniel, and Peter. This is also based on feedback given by Casey Carter.

— Base the text on N4820.

— Introduce exposition-only member variables to be able to name the return values of the functions.

— Reorder members and descriptions as in LFTS V3. But we got rid of the separate code representation of the header and class synopsis.

— specify more properly the concepts/qualities of the type `source_location`.

— We internally discussed if `source_location` should be trivially copyable or `nothrow_copyable`, but only specified the obvious *Cpp17xxx* and swappable requirements for the type, because we do not want to close the design space for implementors.

— While preparing the update for the paper we discussed if the functions in `source_location` are signal safe as with `initializer_list`, but did not dare to specify it at this point.

— For the default constructor of `source_location` we reduced the guarantees from "implementation-defined" values to "unspecified but valid" values, because we want to keep the door open for a possible future where these values could potentially be defined in a more concrete manner.

— provide explicit description of the intended represented values in the remarks section of `current()`.

## 2 Instructions to the Editor

Introduce a new header `<source_location>` in subclause ([headers]): Table 19 ([tab:headers.cpp]), and subclause ([compliance]) Table 22 ([tab:headers.cpp.fs]) between 17.7 and 17.8 add a new line:

| 17.x Source Location `<source_location>` |

Add the feature test macro `__cpp_lib_source_location` to Table 17 ([tab:cpp.predefined.ft]) with the corresponding value for header `<source_location>`.

Create a new subclause 17.x ([reflection.src_loc]) in section 17 ([language.support]) before 17.8 ([support.contract]) with the following content:

## 2.1 Class `source_location` [reflection.src_loc]

The header `<source_location>` defines the class `source_location` that provides a means to obtain source location information.

### 2.1.1 Header `<source_location>` Synopsis [reflection.src_loc.synop]

```
namespace std {
  struct source_location {
    // source location construction
    static consteval source_location current() noexcept;
    constexpr source_location() noexcept;

    // source location field access
    constexpr uint_least32_t line() const noexcept;
    constexpr uint_least32_t column() const noexcept;
    constexpr const char* file_name() const noexcept;
    constexpr const char* function_name() const noexcept;

  private:
    uint_least32_t line_;        // exposition only
    uint_least32_t column_;      // exposition only
    const char* file_name_;      // exposition only
    const char* function_name_; // exposition only
  };
}
```

1 The type `source_location` meets the *Cpp17DefaultConstructible*, *Cpp17CopyConstructible*, *Cpp17CopyAssignable*, and *Cpp17Destructible* requirements.

2 Lvalues of type `source_location` are swappable ([swappable.requirements]).

3 All of the following conditions are true:

(3.1) — `is_nothrow_move_constructible_v<source_location>`

(3.2) — `is_nothrow_move_assignable_v<source_location>`

(3.3) — `is_nothrow_swappable_v<source_location>`

4 [*Note:* The intent of `source_location` is to have a small size and efficient copying.– *end note*]

5 The data members `file_name_` and `function_name_` always each refer to an NTBS.

6 The copy/move constructors and the copy/move assignment operators of `source_location` meet the following postconditions: Given two objects `lhs` and `rhs` of type `source_location`, where `lhs`

is a copy/move result of `rhs`, and where `rhs_p` is a value denoting the state of `rhs` before the corresponding copy/move operation, then each of the following conditions is true:

(6.1) — `strcmp(lhs.file_name(), rhs_p.file_name()) == 0`

(6.2) — `strcmp(lhs.function_name(), rhs_p.function_name()) == 0`

(6.3) — `lhs.line() == rhs_p.line()`

(6.4) — `lhs.column() == rhs_p.column()`

### 2.1.2 `source_location` creation [reflection.src_loc.creation]

`static consteval source_location current() noexcept;`

1 *Returns:*

(1.1) — When invoked by a function call whose *postfix-expression* is a (possibly parenthesized) *id-expression* naming `current`, returns a `source_location` with an implementation-defined value. The value should be affected by `#line` ([cpp.line]) in the same manner as for `__LINE__` and `__FILE__`. The values of the exposition-only data members of the returned `source_location` object denote the following information:

`line_` a presumed line number ([cpp.predefined]). Line numbers are presumed to be 1-indexed; however, an implementation is encouraged to use 0 when the line number is unknown.

`column_` an implementation-defined value denoting some offset from the start of the line denoted by `line_`. Column numbers are presumed to be 1-indexed; however, an implementation is encouraged to use 0 when the column number is unknown.

`file_name_` a presumed name of the current source file ([cpp.predefined]) as an NTBS.

`function_name_` a name of the current function such as in `__func__`([dcl.fct.def.general]) if any, an empty string otherwise.

(1.2) — Otherwise, that is, when invoked in some other way, returns a `source_location` whose data members are initialized with valid but unspecified values.

2 *Remarks:* When a *brace-or-equal-initializer* is used to initialize a non-static data member, any calls to `current` should correspond to the location of the constructor or aggregate initialization that initializes the member.

3 [*Note*: When used as a default argument ([dcl.fct.default]), the value of the `source_location` will be the location of the call to `current` at the call site. —*end note*]

4 [*Example*:

```
struct s {
  source_location member = source_location::current();
  int other_member;
  s(source_location loc = source_location::current())
    : member(loc) // values of member will be from call-site
  {}
  s(int blather) : // values of member should be hereabouts
    other_member(blather)
  {}
```

```
    s(double) // values of member should be hereabouts
    {}
  };
  void f(source_location a = source_location::current()) {
    source_location b = source_location::current(); // values in b represent this line
  }

  void g() {
    f(); // f's first argument corresponds to this line of code

    source_location c = source_location::current();
    f(c); // f's first argument gets the same values as c, above
  }
```

*— end example*]

`constexpr source_location() noexcept;`

5 *Effects:* The data members are initialized with valid but unspecified values.

### 2.1.3 `source_location` field access [reflection.src_loc.fields]

`constexpr uint_least32_t line() const noexcept;`

1 *Returns:* `line_`.

`constexpr uint_least32_t column() const noexcept;`

2 *Returns:* `column_`.

`constexpr const char* file_name() const noexcept;`

3 *Returns:* `file_name_`.

`constexpr const char* function_name() const noexcept;`

4 *Returns:* `function_name_`.