Source-Code Information Capture

Robert Douglas

2015-04-08

Document Number:	N4417 followup to N4129
Date:	2015-04-08
Project:	Programming Language C++

1 Introduction

Logging, testing, and invariant checking each produce messages containing information such as file names, line numbers, and function names. Currently, the only way to get at this information, while avoiding code duplication at each invocation, is through the use of function macros. Function macros expand to the location of use, thus allowing __LINE__, __FILE__, and __func__ to be interpreted in the context of the callers site. If this is where the issue ended, it may not be such a big deal. Unfortunately, for each of these domains, we end up with hundreds of macros, used widely across most code bases.

1.1 Changes from N4129

- Reverted information capture to be implementation-defined in its values
- Changed the default constructor behavior to be non-magical, while adding a free-function that contains the source-capture information.
- Removed offset_from_start_of_file()
- Updated names, given feedback

1.2 Straw Poll Results

- Do we want the unified source_location class? 7/3/0/2/0
- Do we want the unified source_location class if function_name() is removed? 3/3/1/4/1
- Do we want to see a paper exploring the separate source_location_line(), etc. functions? 3/2/7/1/0

- Do we want the default argument magic to come from a free function? (vs the constructor) 2/1/7/1/1
- After mention of time::now()... Do we want the default argument magic to come from a free function? (vs the constructor) 3/3/3/1/1

  Followup session:

- "source_location has most support for the type"
- "source_location::current() has most support the function name"
- With the above names, forward to LWG, for Fundamental TS 2? 8/4/1/1/0
- With the above names, forward to LWG, for C++17? 3/5/6/1/0

2 Design Examples

The following examples illustrate some of the expected usage of source information being passed to library functions. These are purely examples for the reader unfamiliar with previous versions and have been updated for recent changes in design.

2.1 Test-Assertions/Invariant-Checks

Updated usage of default argument

```
template<typename T>
void assert_equal(
        T const& l, T const& r,
        source_location sc = source_location::current()) {
        if (!(l == r)) {
                std::ostringstream os;
                os << sc.file_name() << ":" << sc.line_number()
                        << ":" << sc.column()
                        << ":" << sc.function_name()
                        << " Error: " << l << " != " << r;
                throw std::runtime_error(os.str());
        }
}
template<template<class, class> class C, typename T, typename Allocator>
void assert_equal(
        C<T, Allocator> const& l,
        C<T, Allocator> const& r,
        source_location sc = source_location::current()) {
        if (l.size() != r.size()) {
                std::ostringstream os;
                os << sc.file_name() << ":" << sc.line_number()
```

```
                        << ":" << sc.column()
                        << ":" << sc.function_name()
                        << " Error: container sizes mismatch: ("
                        << l.size() << ", " << r.size() << ")";
                throw std::runtime_error(os.str());
        }

        for (typename C<T, Allocator>::const_iterator
                        lit = begin(l), lEndIt = end(l), rit = begin(r);
                lit != lEndIt; ++lit, ++rit) {
                // Explicit about source information, so that
                // the information of the caller of this site is used
                assert_equal(*lit, *rit, sc);
        }
}
```

2.2 Logging

```
template<typename LoggerT, typename MessageT>
void log(
        Logger & l,
        LogLevel level,
        MessageT const& message,
        source_location sc = source_location::current()) {
        if (logger.level() >= level) {
                l << sc.file_name() << ":" << sc.line_number()
                        << ":" << sc.column()
                        << ":" << sc.function_name()
                        << ":" << message << std::endl;
        }
}

template<typename LoggerT, typename MessageT>
void log_debug(
        Logger & l,
        MessageT const& message,
        source_location sc = source_location::current()) {
        log(l, LogLevel::Debug, message, sc);
}
```

3 Proposal

3.1 Library Additions

3.1.1 Create an object `std::source_location`

```
namespace std {
        struct source_location {
                constexpr source_location() noexcept;

                constexpr size_t line_number() const noexcept;
                constexpr size_t column() const noexcept;
                constexpr char const* file_name() const noexcept;
                constexpr char const* function_name() const noexcept;

                constexpr static source_location current() noexcept;
        };
}
```

`constexpr source_location::source_location() noexcept;`

1 *Effects:* Constructs and object of class source_location.

2 *Postconditions:* Values are implementation-defined.

`constexpr int source_location::line_number() const noexcept;`

3 *Returns:* Presumed line number

`constexpr int source_location::column() const noexcept;`

4 *Returns:* Presumed column number

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

5 *Returns:* Presumed file name as non-null NTBS

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

6 *Returns:* If inside the body of a function, returns the presumed function name as a non-null NTBS. Otherwise, returns an empty string.

`std::source_location source_location::current();`

7 *Returns:* std::source_location with implementation-defined fields. Values should be modified by #line in the same manner as for __LINE__ and __FILE__. When used as a default argument (8.3.6), the value of the std::source_location shall be as if it were instantiated at the call site. The address of source_location::current shall not be taken. The value of source_location when used in the context of a brace-or-equal-initializer of a member-declarator, shall be the information corresponding to the start of the brace-or-equal-initializer.

[*Examples:*

```
void f(source_location a = source_location::current()) {
        source_location b = source_location::current(); // values for "b" represent
this line of code
}

void g() {
        f(); // f's "a" represents this line of code

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

– *end example*]