user-generated `static_assert` messages

Document #:	D2741R0
Date:	2022-12-09
Programming Language C++	
Audience:	EWG

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

Abstract

We propose that `static_assert` should accept user-generated string-like objects as their diagnostic message.

Revisions

R0

Initial revision

Motivation

We propose that the message of a `static_assert` could be an arbitrary constant expression producing a sequence of characters, rather than be limited to string literals. This would allow libraries doing work at compile time to be able to better diagnose the exact problem. This could be used, for example, to

- Explain why a formatting string in format is invalid
- Explain why as compile time regex in CTRE is invalid. In general, this would be helpful for any library that generates code at compile time, such as DSL generators (lexy), Unicode table generators, unit test frameworks, or any other library or DSL who needs to diagnose complex constraints.
- Better explain the constraints of compile-times APIs
- Reduce the reliance on the preprocessor and stringification of identifiers
- Avoid duplication in static assert messages

Without this proposal

```
template <typename T, auto Expected, unsigned long Size = sizeof(T)>
constexpr bool ensure_size() {
    static_assert(sizeof(T) == Expected, "Unexpected sizeof");
    return true;
}
static_assert(ensure_size<S, 1>());
```

```
error: static assertion failed due to requirement 'sizeof(S) == 1':
    Unexpected sizeof
static_assert(sizeof(T) == Expected, "Unexpected sizeof");
^             ~~~~~~~~~~~~~~~~~~~~~
note: in instantiation of function template specialization
    'ensure_size<S, 1, 4ULL>' requested here
static_assert(ensure_size<S, 1>());
^
note: expression evaluates to '4 == 1'
static_assert(sizeof(T) == Expected, "Unexpected sizeof");
~~~~~~~~~~^~~~~~~~~~
1 error generated.
Compiler returned: 1
```

With this proposal

```
static_assert(sizeof(S) == 1,
    std::format("Unexpected sizeof: expected 1, got {}", sizeof(S))); // *
```

```
error: static assertion failed due to requirement 'sizeof(S) == 1':
    Unexpected sizeof: expected 1, got 4
static_assert(sizeof(S) == 1,
^             ~~~~~~~~~~~~~~
note: expression evaluates to '4 == 1'
static_assert(sizeof(S) == 1,
~~~~~~~~~~^~~~
1 error generated.
Compiler returned: 1
```

* `constexpr std::format` is not proposed in this proposal (and very much intentionally so this is but one building block). We should note however that `libfmt` has supported constexpr formatting since version 8.0.0, mid-2021.

This both simplifies the code and makes the diagnostic clearer.

Interaction with reflection

The capabilities presented here would be made more useful by reflection (P1240 [**?**]), notably for the ability to use the name of an instantiated template parameters in diagnostic messages, however, both features are independently useful and do not overlap and there is no need to tie these features together.

Community use cases

Here is a sampling of discussions of this facility online

- Stackoverflow - How to combine static_assert with sizeof and stringify?

  > Memory usage is quite critical in my application. Therefore I have specific asserts that check for the memory size at compile time and give a static_assert if the size is different from what we considered correct before. [..] The problem is that when this static_assert goes off, it might be quite difficult to find out what the new size should be. [...] It would be much handier if I could include the actual size.

  People replying suggest instead injecting a template parameter in the function enclosing the static_assert, which would be outputted by most compilers. The generated error message is not user-friendly.

  ```
  In instantiation of ‘void check_size() [with ToCheck = foo; long
      unsigned int ExpectedSize = 8ul; long unsigned int RealSize = 16ul’]:

  bla.cpp:15:22:   required from here
  bla.cpp:5:1: error: static assertion failed: Size is off!
  ```

- Stackoverflow - Better Message For ‘static_assert‘ on Object Size

  Similar use case.

- Display integer at compile time in static_assert()

  The user would like to express the following code:

  ```
  enum First
  {
      a,
      b,
      c,
      nbElementFirstEnum,
  };
  enum Second
  {
      a,
  ```

```
    b,
    c,
    nbElementSecondEnum,
};

static_assert(
First::nbElementFirstEnum == Second::nbElementSecondEnum,
"Not the same number of element in the enums." + First::
    nbElementFirstEnum + " " + Second::nbElementSecondEnum);
```

There again, the suggestion is to surface these values as a template parameter, hoping the compiler would show enough content to surface them.

- Stack overflow - How to pass a not explicitly string literal error message to a static_assert?

  In this question, the user would like to reuse the same message in multiple static_assert and is reluctant to either copy-paste their code or use a macro. Alas, there is no better solution.

- Many other questions in stack overflow would require reflection to be fully solved: They are all more or less identical to this one: C++11 static_assert: Parameterized error messages

- That feature was previously requested and discussed on std-proposals here, and here.

Design

We proposed to allow a constant expression string as the second parameter of `static_assert`. That's it. In particular, we propose no way to construct a string, as these are orthogonal concerns that can be handled by reflection, making `std::format` `constexpr`, or by string interpolation (P1819r0 [**?**]), or simply by concatenating `std::string`s or using third-party libraries, or, a combination of some or all of the above. The question we are answering in this paper is: I have a string, can I use it as my `static_assert` message?

What is a string?

We do not think this core-language feature should be tied to a specific type or header like `std::string_view`. Indeed many user-defined types can be used to form and store a diagnostic message, and relying on details of the standard libraries are likely to be more complicated than note for implementers and users alike. Instead, we propose a definition of a string-like type that allows the support of `string` and `string_view`, as well as similar user-defined types. A compatible string-like type is a type that:

- Has a `size()` method that produces an integer
- Has a `data()` method that produces a pointer of character type such that
- The elements in the range `[data(), data()+size())` are valid.

This is consistent with how structured bindings and range for loops work.

Encodings

Constant evaluation deals with literal encoding, which may not be UTF-8. As such, `static_assert` should allow both `char` and `char8_t` as messages and will need to convert both to the encoding of diagnostic messages. This is different from string literals in static assert which are not evaluated and converted directly from Unicode (likely UTF-8) to the encoding of diagnostic messages.

Support for `wchar_t`, `char16_t`, `char32_t` is not proposed, but would not be an issue.

Note however that, if an implementation did not have the ability to convert from the literal encoding to the diagnostic encoding, properly rendering non-basic characters in a sequence of `char` might be the most involved part of this proposal. same if we were to support `wchar_t`. Supporting any of the UTF encodings is however a non-issue.

What about null-terminated strings?

Null-terminated strings are mostly useful to communicate with C libraries and systems and are rarely useful at compile time. While it would not be a huge effort to support them, we think it's best to keep the design simple.

Alternatives

In the status quo, depending on the specific use case, macros can be used to stringify some arguments, or the `static_assert` can be lifted in a function template such that most compilers should print the value of this template parameters as part of the diagnostic message.

Neither of these solutions is really satisfying or complete.

Previous work

A similar capability was previously proposed in 2014 by N4433 [1]. At the time, the consensus was that it required too many pieces that did not exist then. As string and string_view are constexpr, `std::format` could be made constexpr (and the fmt lib already can create messages at compile times), and reflection is upon us, we think this feature could be immediately useful.

Future work

As `static_assert` is constantly evaluated, it cannot be used to diagnose, for example, unsatisfied preconditions on parameters and local variables. For that, we will need an additional facility composed of `consteval` functions, as proposed by P0596R1 [2].

Implementation

This feature was implemented in Circle and prototyped in Clang, with no difficulties.

Wording

[Editor's note: This wording assumes P2361R5 has been applied to the working draft] .

🔰 Preamble [dcl.pre]

simple-declaration:
decl-specifier-seq init-declarator-list$_{opt}$;
attribute-specifier-seq decl-specifier-seq init-declarator-list ;
attribute-specifier-seq$_{opt}$ *decl-specifier-seq ref-qualifier*$_{opt}$ `[` *identifier-list* `]` *initializer* ;

static_assert-declaration:
`static_assert` `(` *constant-expression* `)` ;
`static_assert` `(` *constant-expression* `,` *unevaluated-string* `)` ;
`static_assert` `(` *constant-expression* `,` *constant-expression* `)` ;

empty-declaration:
;

attribute-declaration:
attribute-specifier-seq ;

In a *static_assert-declaration*, the first *constant-expression* E is contextually converted to `bool` and the converted expression shall be a constant expression.

If a second *constant-expression* Msg is provided:

- Msg`.size()` shall be a well-formed integral constant expression,
- Msg`.data()` shall be a well-formed constant expression whose type is *cv* `char*` or *cv* `char8_t*`, and
- `[`Msg`.data(), `Msg`.data() + E.size())` shall denote a valid range.

If the value of the expression E when so converted is `true`, the declaration has no effect. Otherwise, the program is ill-formed and the resulting diagnostic message should include ~~the text of the *string-literal*, if one is supplied.~~

- the text formed by the sequence `[`Msg`.data(), `Msg`.data() + E.size())` if Msg is supplied, or
- the text of the *string-literal*, if one is supplied.

[*Example:*

```
static_assert(sizeof(int) == sizeof(void*), "wrong pointer size");
static_assert(sizeof(int[2]));          // OK, narrowing allowed
```

— *end example*]

Feature test macro

[Editor's note: In `[tab:cpp.predefined.ft]`, bump the value of `__cpp_static_assert` to the date of adoption] .

References

[1] Michael Price. N4433: Flexible static_assert messages. https://wg21.link/n4433, 4 2015.

[2] Daveed Vandevoorde. P0596R1: Side-effects in constant evaluation: Output and consteval variables. https://wg21.link/p0596r1, 10 2019.

[N4892] Thomas Köppe *Working Draft, Standard for Programming Language C++* https://wg21.link/N4892