Extending `static_assert`

Document #: WG21 N3846
Date: 2014-01-01
Revises: None
Project: JTC1.22.32 Programming Language C++
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

Abstract

This paper provides proposed wording for several variants of an oft-requested feature: a default string literal for `static_assert`.

1 Introduction

In reflector message [c++std-core-18466], Daniel Krügler writes, in part:

> [P]rogrammers with Boost experience have long lived with the reduced functionality of `BOOST_STATIC_ASSERT` which does not provide the luxury of the message text. For convenience, if `static_assert` is available for the corresponding compiler, it is just mapped to
>
> `#define BOOST_STATIC_ASSERT(B) static_assert(B, #B)`
>
> This looks IMO pretty like an ideal default and is well understood for everyone who has seen a runtime assert ;-)
>
> It looks like a shame to me that programmers would favour to use `BOOST_STATIC_ASSERT` or their own home-grown macro with similar capability.

This feature, a default text message for `static_assert`, has been requested and suggested many times over the past several years; see, for example, reflector messages c++std-core-18466 and c++std-ext-11896, as well as numerous C++ newsgroup messages. Further, near-identical macro-based versions (such as `BOOST_STATIC_ASSERT`, shown above) of the feature have been independently invented and reinvented in several code bases, under several different names.

While there appears to be significant support, there are also some different viewpoints, including competing approaches. Here, in no particular order, are some representative opinions from several perspectives:

- "I . . . believe usability of `static_assert` could be improved by providing a default message" [Peter Sommerlad, c++std-core-24257].
- "This would indeed be convenient" [Faisal Vali, c++std-core-24259].
- "I can't imagine why we wouldn't support it" [Ville Voutilainen, c++std-core-24260].

- "While I sympathize with the proposal, this seems more like something that should be done by the preprocessor, through a macro like [**`BOOST_STATIC_ASSERT`**]. Being a core feature, **`static_assert`** should not mess with stringizing tokens, an operation that should occur in an early phase of translation" [Alberto Ganesh Barbati, c++std-core-18476].
- "It sounds like special pleading. And too late" [Bjarne Stroustrup, c++std-core-18485].
- "I would not oppose a version of this proposal that made the text of the diagnostic output implementation-defined, rather than specifying that it must contain the tokens of the expression. I also would not oppose introduction of a standard macro whose expansion used the stringized version of the expression as the string argument" [Mike Miller, c++std-ext-14628].
- "I would strongly object to macro-based alternatives/solutions.... Let those who deeply care about the preprocessor tokens handle those through macros" [Gabriel dos Reis, c++std-ext-14629].
- "'Let those who deeply care about a single-argument **`static_assert`** handle it through macros.' But I'd be willing to accept a single-argument version with an implementation-defined diagnostic" [Mike Miller, c++std-ext-14630].
- "I prefer to see higher-level, human language messages in **`static_assert`**ions. Can anyone provide specific examples in which

  `#define STATIC_ASSERT(B) static_assert(B, #B)`

  is genuinely more significant or useful in practice than

  `#define STATIC_ASSERT(B) static_assert(B, "ouch")`?

  [...] I would support a form of **`static_assert`** that tests the assertion and has no default message. I regret that this form was not a part of the original proposal" [Robert Klarer, c++std-ext-14657].
- "The main reason I want [the proposed feature] is to apply DRY (Don't Repeat Yourself). [...] I'd like to see what was being tested instead of hoping that the message never gets out of sync with the assertion, especially given that these kinds of things are long enough to get copied and pasted instead of retyped" [Nevin Liber, c++std-ext-14658].
- "Ideally **`static_assert`** should print not one optional message but a variadic list of constant expressions, strings, and typenames. All implementations already have facilities to print types and values in diagnostics, and the lack of this feature in **`static_assert`** often requires falling back on older template tricks (performing time-consuming manual source code adjustment and re-running a potentially long compiler job). No message at all is merely the case of an empty list, and considered as such, the aesthetic desire to specify something in place of the 'missing' string goes away" [David Krauss, personal communication, 2013-12-30].

We have also privately heard that a **`printf`**-style **`static_assert`** message extension would be most welcome.

In the interest of bringing the issue before WG21 for formal consideration and disposition, we propose core language wording not only to achieve the requested behavior per the original suggestion, but also alternative wording for some of the competing ideas. We have not explored any library-based wording.

2 Message guidelines

To provide some context for evaluating the competing proposals, it may be helpful to review some of J. Nielsen's excellent online guidelines for programmers to follow in crafting diagnostic messages for users. The following excerpts from [Niels01] (paraphrased, rearranged, and reformatted to fit onto a slide from one of our talks) seem most applicable:

- Provide explicit indication that something has gone wrong:
  - Users are lost when they make mistakes with no feedback.
  - Be specific in describing the exact problem;
    avoid vague generalities or generic messages (e.g., `"syntax error"`).
- Use human-readable (i.e., comprehensible) language:
  - Avoid abbr's and codes (e.g., `"type 2 error"`).
  - Use polite and grammatically correct phrasing.
  - Neither blame users nor imply they are stupid
    or doing something wrong (e.g., `"illegal command"`).
  - Be affirmative.
- Above all, be helpful; provide constructive advice:
  - Tell users how to address the problem.
  - Use error messages as an educational resource
    to impart a small amount of knowledge to users.

Unlike ordinary diagnostic messages written by programmers typically for end-user consumption, the diagnostic message emitted from a **`static_assert`** is intended specifically for a client programmer. Often, that client is the author of the **`static_assert`**, especially when writing compile-time test code. It would seem that those in this latter situation would most benefit from seeing the exact text of a failing assertion. It further seems that clients in other circumstances might well benefit more from a more carefully crafted diagnostic.

Alas, we seem not likely to be able to serve both contexts with a single extension. We believe that we could come closest, however, by reformulating **`static_assert`** so as to "print not one optional message but a variadic list of constant expressions, strings, and typenames."

3 Proposed wording[1]

3.1 Common wording

There are several alternative wording proposals, but most start out the same:

Augment [dcl.dcl] (Clause 7) paragraph 1 as indicated:

1 Declarations generally specify how names are to be interpreted. Declarations have the form

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

3.2 Alternative 1 (Daniel Krügler's suggestion)

In addition to the common wording from §3.1 above, augment [dcl.dcl] (Clause 7) paragraph 4 as indicated:

4 In a *static_assert-declaration* the *constant-expression* shall be a constant expression (5.19) that can be contextually converted to **`bool`** (Clause 4). If the value of the expression when so converted

[1] All proposed additions and ~~deletions~~ are relative to the post-Chicago Working Draft [N3797]. Editorial notes are displayed against a gray background.

is **`true`**, the declaration has no effect. Otherwise, the program is ill-formed, and the resulting diagnostic message (1.4) shall include the text of the *string-literal*, if one is supplied, except that characters not in the basic source character set (2.3) are not required to appear in the diagnostic message. If no *string-literal* is supplied, the resulting diagnostic message shall consist of the text of the *constant-expression*. [*Example:* . . . — *end example*]

3.3 Alternative 2 (Mike Miller's suggestion)

In addition to the common wording from §3.1 above, augment [dcl.dcl] (Clause 7) paragraph 4 as indicated:

4 In a *static_assert-declaration* the *constant-expression* shall be a constant expression (5.19) that can be contextually converted to **`bool`** (Clause 4). If the value of the expression when so converted is **`true`**, the declaration has no effect. Otherwise, the program is ill-formed, and the resulting diagnostic message (1.4) shall include the text of the *string-literal*, if one is supplied, except that characters not in the basic source character set (2.3) are not required to appear in the diagnostic message. If no *string-literal* is supplied, the resulting diagnostic message shall be implementation-defined. [*Example:* . . . — *end example*]

3.4 Alternative 3 (Robert Klarer's suggestion)

In addition to the common wording from §3.1 above, augment [dcl.dcl] (Clause 7) paragraph 4 as indicated:

4 In a *static_assert-declaration* the *constant-expression* shall be a constant expression (5.19) that can be contextually converted to **`bool`** (Clause 4). If the value of the expression when so converted is **`true`**, the declaration has no effect. Otherwise, the program is ill-formed, and the resulting diagnostic message (1.4) shall include the text of the *string-literal*, if one is supplied, except that characters not in the basic source character set (2.3) are not required to appear in the diagnostic message. [*Example:* . . . — *end example*]

3.5 Alternative 4

Augment [dcl.dcl] (Clause 7) paragraph 1 as indicated:

1 Declarations generally specify how names are to be interpreted. Declarations have the form

. . .
static_assert-declaration:
`static_assert` **`(`** *constant-expression* ~~**`,`** *string-literal*~~ constant-expression-list$_{opt}$ **`)`** **`;`**
constant-expression-list:
constant-expression-list$_{opt}$ **`,`** *constant-expression*
. . .

Augment [dcl.dcl] (Clause 7) paragraph 4 as indicated:

4 In a *static_assert-declaration* the *constant-expression* shall be a constant expression (5.19) that can be contextually converted to **`bool`** (Clause 4). If the value of the expression when so converted is **`true`**, the declaration has no effect. Otherwise, the program is ill-formed, and the resulting diagnostic message (1.4) shall include the ~~text of the *string-literal*~~ value (suitably converted to text) of each *constant-expression*, if any, in the *constant-expression-list*, except that characters not in the basic source character set (2.3) are not required to appear in the diagnostic message. [*Example:* . . . — *end example*]

3.6 Feature-testing macro

For the purposes of SG10, we recommend the macro name **`__cpp_static_assert_extended`**.

4 Bibliography

[N3797] Stefanus Du Toit: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N3797 (post-Chicago mailing), 2013-10-13. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3797.pdf.

[Niels01] Jakob Nielsen: "Error Message Guidelines." Nielsen Norman Group, 2001-06-24. http://www.nngroup.com/articles/error-message-guidelines/.

5 Document history

Version	Date	Changes
1	2014-01-01	• Published as N3846.